Git Error: Too many reference update commands ................................................................. 244
Git Error: push via HTTPS is broken in some versions of Git ........................................... 244
Git Error: 'gnutls_handshake() failed' .............................................................................. 244
Git Error: Git cannot find the AWS CodeCommit repository or does not have permission to access the repository ........................................................................ 244
Git on Windows: No Supported Authentication Methods Available (publickey) .............. 245
Troubleshooting Access Errors ........................................................................................ 245
Access Error: Prompted for AWS User Name When Connecting to an AWS CodeCommit Repository .................................................................................................................. 245
Access Error: Prompted for User Name and Password When Connecting to an AWS CodeCommit Repository from Windows ......................................................................................... 246
Access Error: Public Key Denied When Connecting to an AWS CodeCommit Repository .... 246
Troubleshooting Configuration Errors ................................................................................ 246
Configuration Error: Cannot Configure AWS CLI Credentials on macOS ....................... 247
Troubleshooting Console Errors ........................................................................................ 247
Access Error: Encryption Key Access Denied for an AWS CodeCommit Repository from the Console or the AWS CLI ................................................................. 247
Console Error: Cannot Browse the Code in an AWS CodeCommit Repository from the Console ......................................................................................................................... 247
Troubleshooting Triggers .................................................................................................... 248
Trigger Error: A Repository Trigger Does Not Run When Expected ................................ 248
Troubleshooting E2E Connectivity ..................................................................................... 248
Turn on Debugging ........................................................................................................... 248
Authentication and Access Control .................................................................................... 250
Authentication .................................................................................................................. 250
Access Control .................................................................................................................. 251
Overview of Managing Access ......................................................................................... 251
Resources and Operations ................................................................................................. 252
Understanding Resource Ownership ................................................................................. 253
Managing Access to Resources ........................................................................................ 253
Resource Scoping in AWS CodeCommit ......................................................................... 254
Specifying Policy Elements: Resources, Actions, Effects, and Principals ....................... 255
Specifying Conditions in a Policy .................................................................................... 255
Using Identity-Based Policies (IAM Policies) ..................................................................... 256
Permissions Required to Use the AWS CodeCommit Console ........................................ 256
AWS Managed (Predefined) Policies for AWS CodeCommit ............................................. 258
Customer Managed Policy Examples ................................................................................ 263
AWS CodeCommit Permissions Reference ........................................................................ 267
Required Permissions for Git Client Commands .............................................................. 268
Permissions for Actions on Branches ................................................................................ 269
Permissions for Actions on Pull Requests ........................................................................ 270
Permissions for Actions on Individual Files ..................................................................... 271
Permissions for Actions on Comments ............................................................................. 272
Permissions for Actions on Committed Code .................................................................... 273
Permissions for Actions on Repositories ......................................................................... 274
Permissions for Actions on Triggers .................................................................................. 275
Permissions for Actions on AWS CodePipeline Integration ............................................. 276
AWS CodeCommit Reference ......................................................................................... 278
Regions and Git Connection Endpoints ............................................................................ 278
Supported Regions for AWS CodeCommit ....................................................................... 278
Git Connection Endpoints ............................................................................................... 279
Server Fingerprints for AWS CodeCommit ....................................................................... 281
Limits ................................................................................................................................. 282
Temporary Access ............................................................................................................ 285
Step 1: Complete the Prerequisites ................................................................................... 286
Step 2: Get Temporary Access Credentials ...................................................................... 286
Step 3: Configure the AWS CLI with Your Temporary Access Credentials ....................... 287
Step 4: Access the AWS CodeCommit Repositories .......................................................... 288
AWS KMS and Encryption ............................................................................................... 288
| Encryption Context                                                                 | 289 |
| Logging AWS CodeCommit API Calls with AWS CloudTrail                              | 289 |
| AWS CodeCommit Information in CloudTrail                                           | 289 |
| Understanding AWS CodeCommit Log File Entries                                      | 290 |
| Command Line Reference                                                            | 295 |
| Basic Git Commands                                                                | 297 |
| Configuration Variables                                                           | 297 |
| Remote Repositories                                                               | 298 |
| Commits                                                                           | 299 |
| Branches                                                                          | 299 |
| Tags                                                                              | 300 |
| Document History                                                                  | 302 |
| AWS Glossary                                                                      | 307 |
What Is AWS CodeCommit?

AWS CodeCommit is a version control service hosted by Amazon Web Services that you can use to privately store and manage assets (such as documents, source code, and binary files) in the cloud. For information about pricing for AWS CodeCommit, see Pricing.

Topics
- Introducing AWS CodeCommit (p. 1)
- How Does AWS CodeCommit Work? (p. 1)
- How Is AWS CodeCommit Different from File Versioning in Amazon S3? (p. 2)
- How Do I Get Started with AWS CodeCommit? (p. 3)
- Where Can I Learn More About Git? (p. 3)

Introducing AWS CodeCommit

AWS CodeCommit is a secure, highly scalable, managed source control service that hosts private Git repositories. AWS CodeCommit eliminates the need for you to manage your own source control system or worry about scaling its infrastructure. You can use AWS CodeCommit to store anything from code to binaries. It supports the standard functionality of Git, so it works seamlessly with your existing Git-based tools.

With AWS CodeCommit, you can:
- Benefit from a fully managed service hosted by AWS. AWS CodeCommit provides high service availability and durability and eliminates the administrative overhead of managing your own hardware and software. There is no hardware to provision and scale and no server software to install, configure, and update.
- Store your code securely. AWS CodeCommit repositories are encrypted at rest as well as in transit.
- Work collaboratively on code. AWS CodeCommit repositories support pull requests, where users can review and comment on each other's code changes before merging them to branches; notifications that automatically send emails to users about pull requests and comments; and more.
- Easily scale your version control projects. AWS CodeCommit repositories can scale up to meet your development needs. The service can handle repositories with large numbers of files or branches, large file sizes, and lengthy revision histories.
- Store anything, anytime. AWS CodeCommit has no limit on the size of your repositories or on the file types you can store.
- Integrate with other AWS and third-party services. AWS CodeCommit keeps your repositories close to your other production resources in the AWS Cloud, which helps increase the speed and frequency of your development lifecycle. It is integrated with IAM and can be used with other AWS services and in parallel with other repositories. For more information, see Product and Service Integrations with AWS CodeCommit (p. 68).
- Easily migrate files from other remote repositories. You can migrate to AWS CodeCommit from any Git-based repository.
- Use the Git tools you already know. AWS CodeCommit supports Git commands as well as its own AWS CLI commands and APIs.

How Does AWS CodeCommit Work?

AWS CodeCommit will seem familiar to users of Git-based repositories, but even those unfamiliar should find the transition to AWS CodeCommit relatively simple. AWS CodeCommit provides a console for the
easy creation of repositories and the listing of existing repositories and branches. In a few simple steps, users can find information about a repository and clone it to their computer, creating a local repo where they can make changes and then push them to the AWS CodeCommit repository. Users can work from the command line on their local machines or use a GUI-based editor.

The following figure shows how you use your development machine, the AWS CLI or AWS CodeCommit console, and the AWS CodeCommit service to create and manage repositories:

1. Use the AWS CLI or the AWS CodeCommit console to create an AWS CodeCommit repository.
2. From your development machine, use Git to run `git clone`, specifying the name of the AWS CodeCommit repository. This creates a local repo that connects to the AWS CodeCommit repository.
3. Use the local repo on your development machine to modify (add, edit, and delete) files, and then run `git add` to stage the modified files locally. Run `git commit` to commit the files locally, and then run `git push` to send the files to the AWS CodeCommit repository.
4. Download changes from other users. Run `git pull` to synchronize the files in the AWS CodeCommit repository with your local repo. This ensures you’re working with the latest version of the files.

You can use the AWS CLI or the AWS CodeCommit console to track and manage your repositories.

How Is AWS CodeCommit Different from File Versioning in Amazon S3?

AWS CodeCommit is designed for team software development. It manages batches of changes across multiple files, which can occur in parallel with changes made by other developers. Amazon S3 versioning supports the recovery of past versions of files, but it’s not focused on collaborative file tracking features that software development teams need.
How Do I Get Started with AWS CodeCommit?

To get started with AWS CodeCommit:

1. Follow the steps in Setting Up (p. 4) to prepare your development machines.
2. Follow the steps in one or more of the tutorials in Getting Started (p. 44).
3. Create (p. 76) version control projects in AWS CodeCommit or migrate (p. 211) version control projects to AWS CodeCommit.

Where Can I Learn More About Git?

If you don't know it already, you should learn how to use Git (p. 297). Here are some helpful resources:

- Pro Git, an online version of the Pro Git book. Written by Scott Chacon. Published by Apress.
- Git Immersion, a try-it-yourself guided tour that walks you through the fundamentals of using Git. Published by Neo Innovation, Inc.
- Git Reference, an online quick reference that can also be used as a more in-depth Git tutorial. Published by the GitHub team.
- Git Cheat Sheet with basic Git command syntax. Published by the GitHub team.
- Git Pocket Guide. Written by Richard E. Silverman. Published by O'Reilly Media, Inc.
Setting Up for AWS CodeCommit

You can sign in to the AWS Management Console and upload, add, or edit a file (p. 130) to a repository directly from the AWS CodeCommit console. This is a quick way to make a change. However, if you want to work with multiple files, files across branches, and so on, consider setting up your local computer to work with repositories. The easiest way to set up AWS CodeCommit is to configure HTTPS Git credentials for AWS CodeCommit. This HTTPS authentication method:

- Uses a static user name and password.
- Works with all operating systems supported by AWS CodeCommit.
- Is also compatible with integrated development environments (IDEs) and other development tools that support Git credentials.

You can use other methods if you do not want to or cannot use Git credentials for operational reasons. Read through these other options carefully, to decide which alternate method will work best for you.

- Setting Up Using Git Credentials (p. 4)
- Setting Up Using Other Methods (p. 4)
- Compatibility for AWS CodeCommit, Git, and Other Components (p. 5)

Setting Up Using Git Credentials

With HTTPS connections and Git credentials, you generate a static user name and password in IAM. You then use these credentials with Git and any third-party tool that supports Git user name and password authentication. This method is supported by most IDEs and development tools. It is the simplest and easiest connection method to use with AWS CodeCommit.

- For HTTPS Users Using Git Credentials (p. 5): Follow these instructions to set up connections between your local computer and AWS CodeCommit repositories using Git credentials.
- For Connections from Development Tools (p. 9): Follow these guidelines to set up connections between your IDE or other development tools and AWS CodeCommit repositories using Git credentials. IDEs that support Git credentials include (but are not limited to) Visual Studio, Eclipse, Xcode, and IntelliJ.

Setting Up Using Other Methods

You can use the SSH protocol instead of HTTPS to connect to your AWS CodeCommit repository. With SSH connections, you create public and private key files on your local machine that Git and AWS CodeCommit use for SSH authentication. You associate the public key with your IAM user. You store the private key on your local machine. Because SSH requires manual creation and management of public and private key files, you might find Git credentials simpler and easier to use with AWS CodeCommit.

Unlike Git credentials, SSH connection setup will vary, depending on the operating system on your local computer.

- For SSH Users Not Using the AWS CLI (p. 24): Follow these abbreviated instructions if you already have a public-private key pair and are familiar with SSH connections on your local computer.
- For SSH Connections on Linux, macOS, or Unix (p. 26): Follow these instructions for a step-by-step walkthrough of creating a public-private key pair and setting up connections on Linux, macOS, or Unix operating systems.
Compatibility for AWS CodeCommit, Git, and Other Components

When working with AWS CodeCommit, you will use Git. You might use other programs, too. The following table provides the latest guidance for version compatibility.

**Version Compatibility Information for AWS CodeCommit**

<table>
<thead>
<tr>
<th>Component</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Git</td>
<td>AWS CodeCommit supports Git versions 1.7.9 and later.</td>
</tr>
<tr>
<td>Curl</td>
<td>AWS CodeCommit requires curl 7.33 and later. However, there is a known issue with HTTPS and curl update 7.41.0. For more information, see Troubleshooting (p. 235).</td>
</tr>
</tbody>
</table>

Setup for HTTPS Users Using Git Credentials

The simplest way to set up connections to AWS CodeCommit repositories is to configure Git credentials for AWS CodeCommit in the IAM console, and then use those credentials for HTTPS connections. You can also use these same credentials with any third-party tool or individual development environment...
Step 1: Initial Configuration for AWS CodeCommit

Follow these steps to set up an AWS account, create an IAM user, and configure access to AWS CodeCommit.

To create and configure an IAM user for accessing AWS CodeCommit

2. Create an IAM user, or use an existing one, in your AWS account. Make sure you have an access key ID and a secret access key associated with that IAM user. For more information, see Creating an IAM User in Your AWS Account.
   
   Note
   AWS CodeCommit requires AWS Key Management Service. If you are using an existing IAM user, make sure there are no policies attached to the user that expressly deny the AWS KMS actions required by AWS CodeCommit. For more information, see AWS KMS and Encryption (p. 288).

3. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
4. In the IAM console, in the navigation pane, choose Users, and then choose the IAM user you want to configure for AWS CodeCommit access.
5. On the Permissions tab, choose Add Permissions.
6. In Grant permissions, choose Attach existing policies directly.
7. Select AWSCodeCommitFullAccess from the list of policies, or another managed policy for AWS CodeCommit access. For more information about managed policies for AWS CodeCommit, see AWS Managed (Predefined) Policies for AWS CodeCommit (p. 258).

   After you have selected the policy you want to attach, choose Next: Review to review the list of policies that will be attached to the IAM user. If the list is correct, choose Add permissions.

   For more information about AWS CodeCommit managed policies and sharing access to repositories with other groups and users, see Share a Repository (p. 80) and Authentication and Access Control for AWS CodeCommit (p. 250).

If you want to use AWS CLI commands with AWS CodeCommit, install the AWS CLI. For more information, see Command Line Reference (p. 295).

Step 2: Install Git

To work with files, commits, and other information in AWS CodeCommit repositories, you must install Git on your local machine. AWS CodeCommit supports Git versions 1.7.9 and later.

To install Git, we recommend websites such as Git Downloads.

   Note
   Git is an evolving, regularly updated platform. Occasionally, a feature change might affect the way it works with AWS CodeCommit. If you encounter issues with a specific version of Git and AWS CodeCommit, review the information in Troubleshooting (p. 235).
Step 3: Create Git Credentials for HTTPS Connections to AWS CodeCommit

After you have installed Git, create Git credentials for your IAM user in IAM. For more information, see Use Git Credentials and HTTPS with AWS CodeCommit in the IAM User Guide.

To set up HTTPS Git Credentials for AWS CodeCommit

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.

   Make sure to sign in as the IAM user who will create and use the Git credentials for connections to AWS CodeCommit.

2. In the IAM console, in the navigation pane, choose Users, and from the list of users, choose your IAM user.

3. On the user details page, choose the Security Credentials tab, and in HTTPS Git credentials for AWS CodeCommit, choose Generate.

   ![HTTPS Git credentials for AWS CodeCommit](image)

   **Note**
   You cannot choose your own user name or password for Git credentials. For more information, see Use Git Credentials and HTTPS with AWS CodeCommit.

4. Copy the user name and password that IAM generated for you, either by showing, copying, and pasting this information into a secure file on your local computer, or by choosing Download credentials to download this information as a .CSV file. You will need this information to connect to AWS CodeCommit.
After you have saved your credentials, choose Close.

**Important**
This is your only chance to save the user name and password. If you do not save them, you can copy the user name from the IAM console, but you cannot look up the password. You must reset the password and then save it.

**Step 4: Connect to the AWS CodeCommit Console and Clone the Repository**

If an administrator has already sent you the name and connection details for the AWS CodeCommit repository, you can skip this step and clone the repository directly.

**To connect to an AWS CodeCommit repository**

2. In the region selector, choose the region where the repository was created. Repositories are specific to an AWS region. For more information, see Regions and Git Connection Endpoints (p. 278).
3. Choose the repository you want to connect to from the list. This opens the Code page for that repository.

   If you see a Welcome page instead of a list of repositories, there are no repositories associated with your AWS account. To create a repository, see the section called “Create a Repository” (p. 76) or follow the steps in the Git with AWS CodeCommit Tutorial (p. 58) tutorial.

4. Choose Connect. Review the instructions and copy the URL to use when connecting to the repository.
5. Open a terminal, command line, or Git shell. Using the HTTPS URL you copied, run the `git clone` command to clone the repository. For example, to clone a repository named `MyDemoRepo` to a local repo named `my-demo-repo` in the US East (Ohio) region:

   ```bash
   ```
The first time you connect, you will be prompted to provide the user name and password for the repository. Depending on the configuration of your local computer, this prompt will either originate from a credential management system for the operating system (for example, Keychain Access for macOS), a credential manager utility for your version of Git (for example, the Git Credential Manager included in Git for Windows), your IDE, or Git itself. Provide the user name and password generated for Git credentials in IAM (the ones you created in Step 3: Create Git Credentials for HTTPS Connections to AWS CodeCommit (p. 7)). Depending on your operating system and other software, this information might be saved for you in a credential store or credential management utility. If so, you should not be prompted again unless you change the password, inactivate the Git credentials, or delete the Git credentials in IAM.

If you do not have a credential store or credential management utility configured on your local computer, you can install one. For more information about Git and how it manages credentials, see Credential Storage in the Git documentation.

For more information, see Connect to the AWS CodeCommit Repository by Cloning the Repository (p. 79) and Create a Commit (p. 164).

Next Steps

You have completed the prerequisites. Follow the steps in AWS CodeCommit Tutorial (p. 44) to start using AWS CodeCommit.

Set Up Connections from Development Tools Using Git Credentials

After you have configured Git credentials for AWS CodeCommit in the IAM console, you can use those credentials with any development tool that supports Git credentials. For example, you can configure access to your AWS CodeCommit repository in AWS Cloud9, Visual Studio, Eclipse, Xcode, IntelliJ, or any integrated development environment (IDE) that integrates Git credentials. After configuring access, you can edit your code, commit your changes, and push directly from the IDE or other development tool.

Topics

- Integrate AWS Cloud9 with AWS CodeCommit (p. 12)
- Integrate Visual Studio with AWS CodeCommit (p. 16)
- Integrate Eclipse with AWS CodeCommit (p. 19)

When prompted by your IDE or development tool for the user name and password used to connect to the AWS CodeCommit repository, provide the Git credentials for User name and Password you created in IAM. For example, if you are prompted for a user name and password in Eclipse, you would provide your Git credentials as follows:
For more information about regions and endpoints for AWS CodeCommit, see Regions and Git Connection Endpoints (p. 278).

You might also see a prompt from your operating system to store your user name and password. For example, in Windows, you would provide your Git credentials as follows:

![Windows Security dialog](image)

For more information about configuring Git credentials for a particular software program or development tool, consult the product documentation.

The following is not a comprehensive list of IDEs. AWS is not responsible for the content of any of these topics. The links are provided solely to help you learn more about these tools.

- AWS Cloud9 (p. 12)
• Visual Studio

Alternatively, install the AWS Toolkit for Visual Studio. For more information, see Integrate Visual Studio with AWS CodeCommit (p. 16).

• EGit with Eclipse
Alternatively, install the AWS Toolkit for Eclipse. For more information, see Integrate Eclipse with AWS CodeCommit (p. 19).

• IntelliJ

• XCode

**Integrate AWS Cloud9 with AWS CodeCommit**

You can use AWS Cloud9 to make code changes in an AWS CodeCommit repository. AWS Cloud9 contains a collection of tools that you can use to write code, as well as build, run, test, debug, and release software. You can clone existing repositories, create repositories, commit and push code changes to a repository, and more, all from your AWS Cloud9 EC2 development environment. The AWS Cloud9 EC2 development environment is generally preconfigured with the AWS CLI, an Amazon EC2 role, and Git, so in most cases, you can run a few simple commands and start interacting with your repository.

To use AWS Cloud9 with AWS CodeCommit, you need the following:

• An AWS Cloud9 EC2 development environment running on Amazon Linux.
• The AWS Cloud9 IDE open in a web browser.
• An IAM user with one of the AWS CodeCommit managed policies and one of the AWS Cloud9 managed policies applied to it.

For more information, see AWS Managed (Predefined) Policies for AWS CodeCommit (p. 258) and Understanding and Getting Your Security Credentials.
Topics

- Step 1: Create an AWS Cloud9 Development Environment (p. 13)
- Step 2: Configure the AWS CLI Credential Helper On Your AWS Cloud9 EC2 Development Environment (p. 14)
- Step 3: Clone an AWS CodeCommit Repository Into Your AWS Cloud9 EC2 Development Environment (p. 15)
- Next Steps (p. 16)

Step 1: Create an AWS Cloud9 Development Environment

AWS Cloud9 will host your development environment on an Amazon EC2 instance. This is the easiest way to integrate, as you can use the AWS managed temporary credentials for the instance to connect to your AWS CodeCommit repository. If you want to use your own server instead, see the AWS Cloud9 User Guide.

To create an AWS Cloud9 environment

1. Sign in to AWS as the IAM user you've configured and open the AWS Cloud9 console.
2. In the AWS Cloud9 console, choose Create environment.
3. In Step 1: Name environment, type a name for your development environment in Name. Optionally add a description for the environment, and then choose Next step.
4. In Step 2: Configure Settings, configure your environment as follows:
   - In Environment type, choose Create a new instance for environment (EC2).
   - In Instance type, choose the appropriate instance type for your development environment. For example, if you're just exploring the service, you might choose the default of t2.micro. If you intend to use this environment for development work, choose a larger instance type.
   - Accept the other default settings unless you have specific reasons to choose otherwise (for example, your organization uses a specific VPC, or your AWS account does not have any VPCs configured), and then choose Next step.
5. In Step 3: Review, review your settings. Choose Previous step if you want to make any changes. If not, choose Create environment.

Creating an environment and connecting to it for the first time takes several minutes. If it seems to take an unusually long time, see Troubleshooting in the AWS Cloud9 User Guide.

6. Once you are connected to your environment, check to see if Git is already installed and is a supported version by running the git --version command in the terminal window.
If Git is not installed, or if it is not a supported version, install a supported version. AWS CodeCommit supports Git versions 1.7.9 and later. To install Git, we recommend websites such as Git Downloads.

**Tip**
Depending on the operating system of your environment, you might be able to use the `yum` command with the `sudo` option to install updates, including Git. For example, an administrative command sequence might resemble the following three commands:

```bash
sudo yum -y update
sudo yum -y install git
git --version
```

7. Configure a user name and email to be associated with your Git commits by running the `git config` command. For example:

```bash
git config --global user.name "Mary Major"
git config --global user.email mary.major@example.com
```

**Step 2: Configure the AWS CLI Credential Helper On Your AWS Cloud9 EC2 Development Environment**

After you’ve created an AWS Cloud9 environment, you can configure the AWS CLI Credential Helper to manage the credentials for connections to your AWS CodeCommit repository. The AWS Cloud9 development environment comes with AWS managed temporary credentials that are associated with your IAM user. You will use these credentials with the AWS CLI credential helper.

1. Open the terminal window and type the following command to verify that the AWS CLI is installed:

```bash
aws --version
```

If successful, this command returns the currently-installed version of the AWS CLI. To upgrade an older version of the AWS CLI to the latest version, see [Installing the AWS Command Line Interface](#).

2. At the terminal, type the following commands to configure the AWS CLI Credential Helper for HTTPS connections:
git config --global credential.helper '!aws codecommit credential-helper $@'
git config --global credential.UseHttpPath true

**Tip**
The credential helper will use the default Amazon EC2 instance role for your development environment. If you intend to use the development environment to connect to repositories that are not hosted in AWS CodeCommit, either configure SSH connections to those repositories, or configure a local `.gitconfig` file to use an alternative credential management system when connecting to those other repositories. For more information, see [Git Tools - Credential Storage](https://docs.aws.amazon.com/codecommit/latest/userguide/git-credential.html) on the Git website.

**Step 3: Clone an AWS CodeCommit Repository Into Your AWS Cloud9 EC2 Development Environment**

Once you’ve configured the AWS CLI credential helper, you can clone your AWS CodeCommit repository onto it. Then you can start working with the code.

1. In the terminal, run the `git clone` command, specifying the HTTPS clone URL of the repository you want to clone. For example, if you want to clone a repository named MyDemoRepo in the US East (Ohio) region, you would type:

   ```bash
   git clone https://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo
   ```

   **Tip**
   You can find the Clone URL for your repository in the AWS CodeCommit console, both on the **Dashboard**, and in the **Connect** information on the **Code** page of the repository itself.
2. When the cloning is complete, expand the folder for your repository in the side navigation, and choose the file you want to open for editing. Alternatively, choose File and then choose New File to create a new file.

3. When you have finished editing or creating files, in the terminal window, change directories to your cloned repository and then commit and push your changes. For example, if you added a new file named `MyFile.py`:

   ```bash
   cd MyDemoRepo
   git commit -a MyFile.py
   git commit -m "Added a new file with some code improvements"
   git push
   ```

**Next Steps**

To learn more about using AWS Cloud9, see the AWS Cloud9 User Guide. To learn more about using Git with AWS CodeCommit, see Git with AWS CodeCommit Tutorial (p. 58).

**Integrate Visual Studio with AWS CodeCommit**

You can use Visual Studio to make code changes in an AWS CodeCommit repository. The AWS Toolkit for Visual Studio now includes features that make working with AWS CodeCommit easier and more convenient when working from within Visual Studio Team Explorer. The Toolkit for Visual Studio integration is designed to work with Git credentials and an IAM user. You can clone existing repositories, create repositories, commit and push code changes to a repository, and more.

**Important**

The Toolkit for Visual Studio is available for installation on Windows operating systems only.

If you've used the Toolkit for Visual Studio before, you're probably already familiar with setting up AWS credential profiles that contain an access key and secret key. Credential profiles are used in the Toolkit for Visual Studio to enable calls to AWS service APIs (for example, to Amazon S3 to list buckets or to AWS CodeCommit to list repositories). To pull and push code to an AWS CodeCommit repository, you also need Git credentials. If you don't have Git credentials, the Toolkit for Visual Studio can generate and apply those credentials for you. This can save you a great deal of time.

To use Visual Studio with AWS CodeCommit, you need the following:

- An IAM user with a valid set of credentials (an access key and secret key) configured for it. This IAM user should also have:
  
  One of the AWS CodeCommit managed policies and the IAMSelfManageServiceSpecificCredentials managed policy applied to it.

  OR

  If the IAM user already has Git credentials configured, one of the AWS CodeCommit managed policies or equivalent permissions.

  For more information, see AWS Managed (Predefined) Policies for AWS CodeCommit (p. 258) and Understanding and Getting Your Security Credentials.

- The AWS Toolkit for Visual Studio installed on the computer where you've installed Visual Studio. For more information, see Setting Up the AWS Toolkit for Visual Studio.

**Topics**

- Step 1: Get an Access Key and Secret Key for Your IAM User (p. 17)
Step 1: Get an Access Key and Secret Key for Your IAM User

If you do not already have a credential profile set up on the computer where Visual Studio is installed, you can configure one with the AWS CLI and the aws configure command. Alternatively, you can follow this procedure to create and download your credentials. Provide them to the Toolkit for Visual Studio when prompted.

To get the access key ID and secret access key for an IAM user

Access keys consist of an access key ID and secret access key, which are used to sign programmatic requests that you make to AWS. If you don't have access keys, you can create them from the AWS Management Console. We recommend that you use IAM access keys instead of AWS account root user access keys. IAM lets you securely control access to AWS services and resources in your AWS account.

The only time that you can view or download the secret access keys is when you create the keys. You cannot recover them later. However, you can create new access keys at any time. You must also have permissions to perform the required IAM actions. For more information, see Permissions Required to Access IAM Resources in the IAM User Guide.

1. Open the IAM console.
2. In the navigation pane of the console, choose Users.
3. Choose your IAM user name (not the check box).
4. Choose the Security credentials tab and then choose Create access key.
5. To see the new access key, choose Show. Your credentials will look something like this:
   - Access key ID: AKIAIOSFODNN7EXAMPLE
   - Secret access key: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
6. To download the key pair, choose Download .csv file. Store the keys in a secure location.
   - Keep the keys confidential in order to protect your AWS account, and never email them. Do not share them outside your organization, even if an inquiry appears to come from AWS or Amazon.com. No one who legitimately represents Amazon will ever ask you for your secret key.

Related topics
- What Is IAM? in the IAM User Guide
- AWS Security Credentials in AWS General Reference

Step 2: Install AWS Toolkit for Visual Studio and Connect to AWS CodeCommit

The Toolkit for Visual Studio is a software package you can add to Visual Studio. After you've installed it, you can connect to AWS CodeCommit from Team Explorer in Visual Studio.

To install the Toolkit for Visual Studio with the AWS CodeCommit module and configure access to your project repository

1. Install Visual Studio on your local computer if you don't have a supported version already installed.
2. **Download and install the Toolkit for Visual Studio** and save the file to a local folder or directory. Launch the installation wizard by opening the file. When prompted on the **Getting Started with the AWS Toolkit for Visual Studio** page, type or import your AWS credentials (your access key and secret key), and then choose **Save and Close**.

3. In Visual Studio, open **Team Explorer**. In **Hosted Service Providers**, find **AWS CodeCommit**, and choose **Connect**.

4. Do one of the following:

   - If you have a single credential profile already configured on your computer, the Toolkit for Visual Studio will apply it automatically. No action is required. The AWS CodeCommit connection panel appears in Team Explorer.
   - If you have more than one credential profile configured on your computer, you are prompted to choose the one you want to use. Choose the profile associated with the IAM user you'll use for connecting to AWS CodeCommit repositories, and then choose **OK**.
   - If you do not have a profile configured, a dialog box appears and asks for your AWS security credentials (your access key and secret key). Type or import them, and then choose **OK**.

   After you are signed in with a profile, the AWS CodeCommit connection panel appears in Team Explorer with options to clone, create, or sign out. Choosing **Clone** clones an existing AWS CodeCommit repository to your local computer, so you can start working on code. This is the most frequently used option.

   If you don't have repositories, or want to create a repository, choose **Create**. For more information, see **Create an AWS CodeCommit Repository from Visual Studio** (p. 19).

**Clone an AWS CodeCommit Repository from Visual Studio**

After you're connected to AWS CodeCommit, you can clone a repository to a local repo on your computer. Then you can start working with the code.

1. In **Manage Connections**, choose **Clone**. In **Region**, choose the AWS region where the repository was created in AWS CodeCommit. Choose your project's repository and the folder on your local computer you want to clone the repository into, and then choose **OK**.

2. If you are prompted to create Git credentials, choose **Yes**. The toolkit attempts to create credentials on your behalf. You must have the IAMSelfManageServiceSpecificCredentials applied to your IAM user, or the equivalent permissions. When prompted, save the credentials file in a secure location. This is the only opportunity you will have to save these Git credentials.

   If the toolkit cannot create Git credentials on your behalf, or if you chose **No**, you must create and provide your own Git credentials. For more information, see **For HTTPS Users Using Git Credentials** (p. 5), or follow the online directions.

3. When you have finished cloning the project, you're ready to start editing your code in Visual Studio and committing and pushing your changes to your project's repository in AWS CodeCommit.
Create an AWS CodeCommit Repository from Visual Studio

You can create AWS CodeCommit repositories from Visual Studio with the Toolkit for Visual Studio. As part of creating the repository, you also clone it to a local repo on your computer, so you can start working with it right away.

1. In Manage Connections, choose Create.
2. In Region, choose the AWS region where you want to create the repository. AWS CodeCommit repositories are organized by region.
3. In Name, type a name for this repository. Repository names must be unique within an AWS account. There are character and length limits. For more information, see Limits (p. 282). Optionally, in Description, type a description for this repository. This helps others understand what the repository is for, and helps distinguish it from other repositories in the region.
4. In Clone into, type or browse to the folder or directory where you want to clone this repository on your local computer. Visual Studio automatically clones the repository after it's created and creates the local repo in the location you choose.
5. When you are satisfied with your choices, choose OK.
6. If prompted to create Git credentials, choose Yes. The toolkit attempts to create credentials on your behalf. You must have the IAMSelfManageServiceSpecificCredentials applied to your IAM user, or the equivalent permissions. When prompted, save the credentials file in a secure location. This is the only opportunity you will have to save these Git credentials.

   If the toolkit cannot create Git credentials on your behalf, or if you chose No, you must create and provide your own Git credentials. For more information, see For HTTPS Users Using Git Credentials (p. 5), or follow the online directions.

Working with AWS CodeCommit Repositories

After you have connected to AWS CodeCommit, you can see a list of repositories associated with your AWS account. You can browse the contents of these repositories in Visual Studio. Open the context menu for the repository you're interested in, and choose Browse in Console.

Git operations in Visual Studio for AWS CodeCommit repositories work exactly as they do for any other Git-based repository. You can make changes to code, add files, and create local commits. When you are ready to share, you use the Sync option in Team Explorer to push your commits to the AWS CodeCommit repository. Because your Git credentials for your IAM user are already stored locally and associated with your connected AWS credential profile, you won't be prompted to supply them again when you push to AWS CodeCommit.

For more information about working with Toolkit for Visual Studio, see the AWS Toolkit for Visual Studio User Guide.

Integrate Eclipse with AWS CodeCommit

You can use Eclipse to make code changes in an AWS CodeCommit repository. The Toolkit for Eclipse integration is designed to work with Git credentials and an IAM user. You can clone existing repositories, create repositories, commit and push code changes to a repository, and more.

To use Toolkit for Eclipse with AWS CodeCommit, you need the following:

- Eclipse installed on your local computer.
- An IAM user with a valid set of credentials (an access key and secret key) configured for it. This IAM user should also have:
One of the AWS CodeCommit managed policies and the IAMSelfManageServiceSpecificCredentials
managed policy applied to it.

OR

If the IAM user already has Git credentials configured, one of the AWS CodeCommit managed policies
or equivalent permissions.

For more information, see AWS Managed (Predefined) Policies for AWS CodeCommit (p. 258) and
Understanding and Getting Your Security Credentials.

- An active set of Git credentials configured for the user in IAM. For more information, see Step 3: Create
  Git Credentials for HTTPS Connections to AWS CodeCommit (p. 7).

Topics

- Step 1: Get an Access Key and Secret Key for Your IAM User (p. 20)
- Step 2: Install AWS Toolkit for Eclipse and Connect to AWS CodeCommit (p. 21)
- Clone an AWS CodeCommit Repository from Eclipse (p. 22)
- Create an AWS CodeCommit Repository from Eclipse (p. 23)
- Working with AWS CodeCommit Repositories (p. 24)

Step 1: Get an Access Key and Secret Key for Your IAM User

If you do not already have a credential profile set up on the computer where Eclipse is installed, you
can configure one with the AWS CLI and the aws configure command. Alternatively, you can follow
this procedure to create and download your credentials. Provide them to the Toolkit for Eclipse when
prompted.

To get the access key ID and secret access key for an IAM user

Access keys consist of an access key ID and secret access key, which are used to sign programmatic
requests that you make to AWS. If you don't have access keys, you can create them from the AWS
Management Console. We recommend that you use IAM access keys instead of AWS account root user
access keys. IAM lets you securely control access to AWS services and resources in your AWS account.

The only time that you can view or download the secret access keys is when you create the keys. You
cannot recover them later. However, you can create new access keys at any time. You must also have
permissions to perform the required IAM actions. For more information, see Permissions Required to
Access IAM Resources in the IAM User Guide.

1. Open the IAM console.
2. In the navigation pane of the console, choose Users.
3. Choose your IAM user name (not the check box).
4. Choose the Security credentials tab and then choose Create access key.
5. To see the new access key, choose Show. Your credentials will look something like this:
   - Access key ID: AKIAIOSFODNN7EXAMPLE
   - Secret access key: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
6. To download the key pair, choose Download .csv file. Store the keys in a secure location.

Keep the keys confidential in order to protect your AWS account, and never email them. Do not share
them outside your organization, even if an inquiry appears to come from AWS or Amazon.com. No one
who legitimately represents Amazon will ever ask you for your secret key.
Step 2: Install AWS Toolkit for Eclipse and Connect to AWS CodeCommit

The Toolkit for Eclipse is a software package you can add to Eclipse. After you've installed it and configured it with your AWS credential profile, you can connect to AWS CodeCommit from the AWS Explorer in Eclipse.

To install the Toolkit for Eclipse with the AWS CodeCommit module and configure access to your project repository

1. Install Toolkit for Eclipse on your local computer if you don't have a supported version already installed. If you need to update your version of Toolkit for Eclipse, follow the instructions in Set Up the Toolkit.
2. In Eclipse, either follow the firstrun experience, or open Preferences from the Eclipse menu system (the precise location will vary depending on your version and operating system) and choose AWS Toolkit.
3. Do one of the following:
   - If you are following the firstrun experience, provide your AWS security credentials when prompted to set up your credential profile.
   - If you are configuring in Preferences and have a credential profile already configured on your computer, choose it from the list in Default Profile.
   - If you are configuring in Preferences and you do not see the profile you want to use, or if the list is empty, choose Add profile. In Profile Details, provide a name for the profile and the credentials for the IAM user (access key and secret key), or alternatively provide the location of the credentials file.
   - If you are configuring in Preferences and you do not have a profile configured, use the links for signing up for a new account or managing your existing AWS security credentials.
4. In Eclipse, expand the AWS Toolkit menu and choose AWS CodeCommit. Choose your credential profile, and then type the user name and password for your Git credentials or import them from the .csv file. Choose Apply, and then choose OK.

Related topics

- What Is IAM? in the IAM User Guide
- AWS Security Credentials in AWS General Reference
After you are signed in with a profile, the AWS CodeCommit connection panel appears in Team Explorer with options to clone, create, or sign out. Choosing **Clone** clones an existing AWS CodeCommit repository to your local computer, so you can start working on code. This is the most frequently used option.

If you don't have any repositories, or want to create a repository, choose **Create**.

### Clone an AWS CodeCommit Repository from Eclipse

After you've configured your credentials, you can clone a repository to a local repo on your computer by checking it out in Eclipse. Then you can start working with the code.

1. In Eclipse, open **AWS Explorer**. For more information about where to find it, see [How to Access AWS Explorer](#). Expand **AWS CodeCommit**, and choose the AWS CodeCommit repository you want to work in. You can view the commit history and other details of the repository, which can help you determine if this is the repository and branch you want to clone.

   **Note**
   
   If you do not see your repository, choose the flag icon to open the AWS regions menu, and choose the region where the repository was created.
2. Choose **Check out**, and follow the instructions to clone the repository to your local computer.

3. When you have finished cloning the project, you’re ready to start editing your code in Eclipse and staging, committing, and pushing your changes to your project’s repository in AWS CodeCommit.

### Create an AWS CodeCommit Repository from Eclipse

You can create AWS CodeCommit repositories from Eclipse with the Toolkit for Eclipse. As part of creating the repository, you’ll also clone it to a local repo on your computer, so you can start working with it right away.

1. In AWS Explorer, right-click **AWS CodeCommit**, and then choose **Create repository**.

   **Note**
   Repositories are region-specific. Before you create the repository, make sure you have selected the correct AWS region. You cannot choose the region after you have started the repository creation process.
2. In **Repository Name**, type a name for this repository. Repository names must be unique within an AWS account. There are character and length limits. For more information, see Limits (p. 282). Optionally, in **Repository Description**, type a description for this repository. This helps others understand what this repository is for, and helps distinguish it from other repositories in the region. Choose OK.

3. In AWS Explorer, expand **AWS CodeCommit**, and then choose the AWS CodeCommit repository you just created. You will see that this repository has no commit history. Choose **Check out**, and follow the instructions to clone the repository to your local computer.

### Working with AWS CodeCommit Repositories

After you have connected to AWS CodeCommit, you can see a list of repositories associated with your AWS account, by region, in AWS Explorer. Choose the flag menu to change the region.

**Note**

AWS CodeCommit might not be available in all regions supported by Toolkit for Eclipse.

In Toolkit for Eclipse, you can browse the contents of these repositories from the **Navigation** and **Package Explorer** views. To open a file, choose it from the list.

Git operations in Toolkit for Eclipse for AWS CodeCommit repositories work exactly as they do for any other Git-based repository. You can make changes to code, add files, and create local commits. When you are ready to share, you use the **Git Staging** option to push your commits to the AWS CodeCommit repository. If you haven’t configured your author and committer information in a Git profile, you can do this before you commit and push. Because your Git credentials for your IAM user are already stored locally and associated with your connected AWS credential profile, you won’t be prompted to supply them again when you push to AWS CodeCommit.

For more information about working with Toolkit for Eclipse, see the AWS Toolkit for Eclipse Getting Started Guide.

### Setup for SSH Users Not Using the AWS CLI

If you want to use SSH connections for your repository, you can connect to AWS CodeCommit without installing the AWS CLI. The AWS CLI includes commands that will be useful later when using and managing AWS CodeCommit repositories, but it is not required for initial setup.

This topic assumes:

- You have set up an IAM user with the policies or permissions required for AWS CodeCommit as well as the **IAMUserSSHKeys** managed policy or equivalent permissions required for uploading keys. For more information, see Using Identity-Based Policies (IAM Policies) for AWS CodeCommit (p. 256).
- You already have, or know how to create, a public/private key pair. We strongly recommend you use a secure passphrase for your SSH key.
- You are familiar with SSH, your Git client, and its configuration files.
- If you are using Windows, you have installed a command-line utility, such as Git Bash, that emulates the bash shell.

If you need more guidance, follow the detailed instructions in For SSH Connections on Linux, macOS, or Unix (p. 26) or For SSH Connections on Windows (p. 30).

**Topics**

- Step 1: Associate Your Public Key with Your IAM User (p. 25)
Step 1: Associate Your Public Key with Your IAM User

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the IAM console, in the navigation pane, choose Users, and from the list of users, choose your IAM user.
4. Paste the contents of your SSH public key into the field, and then choose Upload SSH Key.

   **Tip**
   The public/private key pair must be SSH-2 RSA, in OpenSSH format, and contain 2048 bits. The key will look similar to this:

   ```
   ssh-rsa EXAMPLE-AfIFCCQD6e8oRw0uXOjANBkghkiG9w0BAQUDACBDiCDEMAkGAEiUEBbMCVVMXcAZJ
   gNVAgTAdBMRAwDgYDVKQHEwdTZWF0dGx1MQ8wDQYDVQQKKeZbbWF6b24xFDASBgNVBAoSC0lBTSDBd2
   5b2x1MRtwEADYDVQQDEw1UZ1N0Q21aYWMxHAdBgkqhkiG9w0BCQEMEG5vb251QGFhYXpviSjb20wHhc
   NNTEwNDI1MyA0T1wIRcHcNNT1wIDiomja0NTlXWGcIDEMkGAUEhMCVVMXcAZJBeNVBAgTAldBMRAw
   DgYDVKQHEwdTZWF0dGx1MQ8wDQYDVQQKEZbbWF6b24xFDASEXAMPLE user-
   name@ip-192-0-2-137
   
   IAM accepts public keys in the OpenSSH format only. If you provide your public key in another format, you will see an error message stating the key format is not valid.

5. Copy the SSH key ID (for example, APKAEIBAERJR2EXAMPLE) and close the console.

Step 2: Add AWS CodeCommit to Your SSH Configuration

1. At the terminal (Linux, macOS, or Unix) or bash emulator (Windows), edit your SSH configuration file by typing `cat>> ~/.ssh/config`:

   ```
   Host git-codecommit.*.amazonaws.com
   User Your-SSH-Key-ID, such as APKAEIBAERJR2EXAMPLE
   IdentityFile Your-Private-Key-File, such as ~/.ssh/codecommit_rsa or ~/.ssh/id_rsa
   
   **Tip**
   If you have more than one SSH configuration, make sure you include the blank lines before and after the content. Save the file by pressing the `Ctrl` and `a` keys simultaneously.

2. Run the following command to test your SSH configuration:

   ```
   ssh git-codecommit.us-east-2.amazonaws.com
   ```
Type the passphrase for your SSH key file when prompted. If everything is configured correctly, you should see the following success message:

You have successfully authenticated over SSH. You can use Git to interact with AWS CodeCommit.
Interactive shells are not supported. Connection to git-codecommit.us-east-2.amazonaws.com closed by remote host.

Next Steps

You have completed the prerequisites. Follow the steps in AWS CodeCommit Tutorial (p. 44) to start using AWS CodeCommit.

To connect to an existing repository, follow the steps in Connect to a Repository (p. 78). To create a repository, follow the steps in Create a Repository (p. 76).

Setup Steps for SSH Connections to AWS CodeCommit Repositories on Linux, macOS, or Unix

Before you can connect to AWS CodeCommit for the first time, you must complete the initial configuration steps. This topic walks you through the steps for setting up your computer and AWS profile, connecting to an AWS CodeCommit repository, and cloning that repository to your computer (also known as creating a local repo). If you're new to Git, you might also want to review the information in Where Can I Learn More About Git? (p. 3).

Topics

- Step 1: Initial Configuration for AWS CodeCommit (p. 26)
- Step 2: Install Git (p. 27)
- Step 3: Configure Credentials on Linux, macOS, or Unix (p. 27)
- Step 4: Connect to the AWS CodeCommit Console and Clone the Repository (p. 29)
- Next Steps (p. 30)

Step 1: Initial Configuration for AWS CodeCommit

Follow these steps to set up an AWS account, create an IAM user, and configure access to AWS CodeCommit.

To create and configure an IAM user for accessing AWS CodeCommit

2. Create an IAM user, or use an existing one, in your AWS account. Make sure you have an access key ID and a secret access key associated with that IAM user. For more information, see Creating an IAM User in Your AWS Account.

   Note
   AWS CodeCommit requires AWS Key Management Service. If you are using an existing IAM user, make sure there are no policies attached to the user that expressly deny the
AWS KMS actions required by AWS CodeCommit. For more information, see AWS KMS and Encryption (p. 288).

3. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
4. In the IAM console, in the navigation pane, choose Users, and then choose the IAM user you want to configure for AWS CodeCommit access.
5. On the Permissions tab, choose Add Permissions.
6. In Grant permissions, choose Attach existing policies directly.
7. Select AWSCodeCommitFullAccess from the list of policies, or another managed policy for AWS CodeCommit access. For more information about managed policies for AWS CodeCommit, see AWS Managed (Predefined) Policies for AWS CodeCommit (p. 258).

After you have selected the policy you want to attach, choose Next: Review to review the list of policies that will be attached to the IAM user. If the list is correct, choose Add permissions.

For more information about AWS CodeCommit managed policies and sharing access to repositories with other groups and users, see Share a Repository (p. 80) and Authentication and Access Control for AWS CodeCommit (p. 250).

Note
If you want to use AWS CLI commands with AWS CodeCommit, install the AWS CLI. For more information, see Command Line Reference (p. 295).

Step 2: Install Git

To work with files, commits, and other information in AWS CodeCommit repositories, you must install Git on your local machine. AWS CodeCommit supports Git versions 1.7.9 and later.

To install Git, we recommend websites such as Git Downloads.

Note
Git is an evolving, regularly updated platform. Occasionally, a feature change might affect the way it works with AWS CodeCommit. If you encounter issues with a specific version of Git and AWS CodeCommit, review the information in Troubleshooting (p. 235).

Step 3: Configure Credentials on Linux, macOS, or Unix

SSH and Linux, macOS, or Unix: Set Up the Public and Private Keys for Git and AWS CodeCommit

1. From the terminal on your local machine, run the ssh-keygen command, and follow the directions to save the file to the .ssh directory for your profile.

   Note
   Be sure to check with your system administrator about where key files should be stored and which file naming pattern should be used.

   For example:

   ```
   # ssh-keygen
   Generating public/private rsa key pair.
   ```
Enter file in which to save the key (/home/user-name/.ssh/id_rsa): Type /home/your-user-name/.ssh/ and a file name here, for example /home/your-user-name/.ssh/codecommit_rsa

Enter passphrase (empty for no passphrase): <Type a passphrase, and then press Enter>
Enter same passphrase again: <Type the passphrase again, and then press Enter>

Your identification has been saved in /home/user-name/.ssh/codecommit_rsa.
Your public key has been saved in /home/user-name/.ssh/codecommit_rsa.pub.
The key fingerprint is:
The key's randomart image is:
```plaintext
+---[ RSA 2048]----+
| E+...E+*+++|     |
| .o,=.=o.  |     |
| .. *   +    |     |
| .. +   .    |     |
| So      ... |     |
|        .    |     |
|          .  |     |
|               |     |
+-----------------+
```

This generates:
- The codecommit_rsa file, which is the private key file.
- The codecommit_rsa.pub file, which is the public key file.

2. Run the following command to display the value of the public key file (codecommit_rsa.pub):

```
cat ~/.ssh/codecommit_rsa.pub
```

Copy this value. It will look similar to the following:
```
ssh-rsa EXAMPLE-AfICCCQD6m7o8W0uXOjJANBkgkhhkiG9w0BAQUFADCBiDELMAkGA1UEBhMCVMxMzAIAgNGBgTAd28c=
```

3. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.

4. In the IAM console, in the navigation pane, choose Users, and from the list of users, choose your IAM user.

5. On the user details page, choose the Security Credentials tab, and then choose Upload SSH public key.

6. Paste the contents of your SSH public key into the field, and then choose Upload SSH public key.

7. Copy or save the information in SSH Key ID (for example, APKAEIBAERJR2EXAMPLE).

8. On your local machine, use a text editor to create a config file in the ~/.ssh directory, and then add the following lines to the file, where the value for User is the SSH key ID you copied earlier:

```
Host git-codecommit.*.amazonaws.com
```

API Version 2015-04-13
Step 4: Connect to the AWS CodeCommit Console and Clone the Repository

If an administrator has already sent you the name and connection details for the AWS CodeCommit repository, you can skip this step and clone the repository directly.

To connect to an AWS CodeCommit repository

2. In the region selector, choose the region where the repository was created. Repositories are specific to an AWS region. For more information, see Regions and Git Connection Endpoints (p. 278).
3. Choose the repository you want to connect to from the list. This opens the Code page for that repository.
If you see a Welcome page instead of a list of repositories, there are no repositories associated with your AWS account. To create a repository, see the section called “Create a Repository” (p. 76) or follow the steps in the Git with AWS CodeCommit Tutorial (p. 58) tutorial.

4. Copy the SSH URL to use when connecting to the repository.

5. Open a terminal. From the /tmp directory, using the SSH URL you copied, run the git clone command to clone the repository. For example, to clone a repository named MyDemoRepo to a local repo named my-demo-repo in the US East (Ohio) region:

   ```bash
   git clone ssh://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo my-demo-repo
   ```

   **Note**
   If you successfully tested your connection, but the clone command fails, you might not have the necessary access to your config file, or another setting might be in conflict with your config file. Try connecting again, this time including the SSH key ID in the command. For example:

   ```bash
   git clone ssh://Your-SSH-Key-ID@git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo my-demo-repo
   ```

   For more information, see Access Error: Public Key Is Uploaded Successfully to IAM but Connection Fails on Linux, macOS, or Unix Systems (p. 236).

   For more information about how to connect to repositories, see Connect to the AWS CodeCommit Repository by Cloning the Repository (p. 79).

**Next Steps**

You have completed the prerequisites. Follow the steps in AWS CodeCommit Tutorial (p. 44) to start using AWS CodeCommit.

**Setup Steps for SSH Connections to AWS CodeCommit Repositories on Windows**

Before you can connect to AWS CodeCommit for the first time, you must complete the initial configuration steps. This topic walks you through the steps for setting up your computer and AWS profile, connecting to an AWS CodeCommit repository, and cloning that repository to your computer (also known as creating a local repo). If you’re new to Git, you might also want to review the information in Where Can I Learn More About Git? (p. 3).

**Topics**

- Step 1: Initial Configuration for AWS CodeCommit (p. 31)
- Step 2: Install Git (p. 31)
- SSH and Windows: Set Up the Public and Private Keys for Git and AWS CodeCommit (p. 32)
- Step 4: Connect to the AWS CodeCommit Console and Clone the Repository (p. 34)
- Next Steps (p. 34)
Step 1: Initial Configuration for AWS CodeCommit

Follow these steps to set up an AWS account, create an IAM user, and configure access to AWS CodeCommit.

To create and configure an IAM user for accessing AWS CodeCommit

2. Create an IAM user, or use an existing one, in your AWS account. Make sure you have an access key ID and a secret access key associated with that IAM user. For more information, see Creating an IAM User in Your AWS Account.
   
   **Note**
   AWS CodeCommit requires AWS Key Management Service. If you are using an existing IAM user, make sure there are no policies attached to the user that expressly deny the AWS KMS actions required by AWS CodeCommit. For more information, see AWS KMS and Encryption (p. 288).
3. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
4. In the IAM console, in the navigation pane, choose Users, and then choose the IAM user you want to configure for AWS CodeCommit access.
5. On the Permissions tab, choose Add Permissions.
6. In Grant permissions, choose Attach existing policies directly.
7. Select AWSCodeCommitFullAccess from the list of policies, or another managed policy for AWS CodeCommit access. For more information about managed policies for AWS CodeCommit, see AWS Managed (Predefined) Policies for AWS CodeCommit (p. 258).

   After you have selected the policy you want to attach, choose Next: Review to review the list of policies that will be attached to the IAM user. If the list is correct, choose Add permissions.

   For more information about AWS CodeCommit managed policies and sharing access to repositories with other groups and users, see Share a Repository (p. 80) and Authentication and Access Control for AWS CodeCommit (p. 250).

   **Note**
   If you want to use AWS CLI commands with AWS CodeCommit, install the AWS CLI. For more information, see Command Line Reference (p. 295).

Step 2: Install Git

To work with files, commits, and other information in AWS CodeCommit repositories, you must install Git on your local machine. AWS CodeCommit supports Git versions 1.7.9 and later.

To install Git, we recommend websites such as Git Downloads.

   **Note**
   Git is an evolving, regularly updated platform. Occasionally, a feature change might affect the way it works with AWS CodeCommit. If you encounter issues with a specific version of Git and AWS CodeCommit, review the information in Troubleshooting (p. 235).

   If the version of Git you installed does not include a Bash emulator, such as Git Bash, install one. You will use this emulator instead of the Windows command line when you configure SSH connections.
SSH and Windows: Set Up the Public and Private Keys for Git and AWS CodeCommit

1. Open the Bash emulator.

   **Note**
   You might need to run the emulator with administrative permissions.

   From the emulator, run the `ssh-keygen` command, and follow the directions to save the file to the `.ssh` directory for your profile.

   For example:

   ```bash
   $ ssh-keygen
   Generating public/private rsa key pair.
   Enter file in which to save the key (/drive/Users/user-name/.ssh/id_rsa): Type a file name here, for example /c/Users/user-name/.ssh/codecommit_rsa
   Enter passphrase (empty for no passphrase): Type a passphrase, and then press Enter
   Enter same passphrase again: Type the passphrase again, and then press Enter
   
   Your identification has been saved in /drive/Users/user-name/.ssh/codecommit_rsa.
   Your public key has been saved in /drive/Users/user-name/.ssh/codecommit_rsa.pub.
   The key fingerprint is:
   
   The key's randomart image is:
   +--[ RSA 2048]----+
   |        E.+.o*.++|
   |        .o .=.=o.|
   |       . ..  *. +|
   |        ..o . +..|
   |        So . . . |
   |          .      |
   |                 |
   |                 |
   +-----------------+
   
   This generates:
   
   - The `codecommit_rsa` file, which is the private key file.
   - The `codecommit_rsa.pub` file, which is the public key file.

2. Run the following commands to display the value of the public key file (`codecommit_rsa.pub`):

   ```bash
   cd .ssh
   notepad codecommit_rsa.pub
   
   Copy the contents of the file, and then close Notepad without saving. The contents of the file will look similar to the following:
   
   ssh-rsa EXAMPLExAfICCQD6m7oRw0uXOjANBgpqkhkiG9w0BAQUFACDBiDELMAkGA1UEBhMCVVMg
   gNVBAgTAlidBMRAwDgYDVQQHEwtdTWF0dGx1MQ8wDQYDVQQKEwZBBWF6b24xFDASBgNVBAEw
   TC0lBTSDBb25zb2xldBMRAwDgYDVQQHEwZBBWF6b24xFDASBgNVBAMwEwZBBWF6b24xFDAS=
   EXAMPLE user-name@computer-name
   ```

API Version 2015-04-13

32
3. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.

4. In the IAM console, in the navigation pane, choose Users, and from the list of users, choose your IAM user.

5. On the user details page, choose the Security Credentials tab, and then choose Upload SSH public key.

6. Paste the contents of your SSH public key into the field, and then choose Upload SSH public key.

7. Copy or save the information in SSH Key ID (for example, APKAEIBAERJR2EXAMPLE).

8. In the Bash emulator, type the following commands to create a config file in the ~/.ssh directory, or edit it if one already exists:

   ```bash
   notepad ~/.ssh/config
   ```

9. Add the following lines to the file, where the value for User is the SSH key ID you copied earlier, and the value for IdentityFile is the path to and name of the private key file:

   ```
   Host git-codecommit.*.amazonaws.com
   User APKAEIBAERJR2EXAMPLE
   IdentityFile ~/.ssh/codecommit_rsa
   ```

   **Note**
   If you gave your private key file a name other than codecommit_rsa, be sure to use it here.

   Save the file as config (not config.txt), and then close Notepad.

   **Important**
   The name of the file must be config with no file extension, or the SSH connections will fail.

10. Run the following command to test your SSH configuration:

    ```bash
    ssh git-codecommit.us-east-2.amazonaws.com
    ```

    You will be asked to confirm the connection because git-codecommit.us-east-2.amazonaws.com is not yet included in your known hosts file. The AWS CodeCommit server fingerprint is displayed as part of the verification (a9:6d:03:ed:08:42:21:be:06:e1:e0:2a:d1:75:31:5e for MD5 or 3lBlW2g5xn/NA2CK6dyeJrQ0Wvn7n8UEs56f62ZIzQ for SHA256).

    **Note**
    AWS CodeCommit server fingerprints are unique for every region. To view the server fingerprints for a specific region, see Server Fingerprints for AWS CodeCommit (p. 281).

    After you have confirmed the connection, you should see confirmation that you have added the server to your known hosts file and a successful connection message. If you do not see a success message, double-check that you saved the config file in the ~/.ssh directory of the IAM user you configured for access to AWS CodeCommit, that the config file has no file extension (for example, it must not be named config.txt), and that you specified the correct private key file (codecommit_rsa, not codecommit_rsa.pub).
Step 4: Connect to the AWS CodeCommit Console and Clone the Repository

If an administrator has already sent you the name and connection details for the AWS CodeCommit repository, you can skip this step and clone the repository directly.

To connect to an AWS CodeCommit repository

2. In the region selector, choose the region where the repository was created. Repositories are specific to an AWS region. For more information, see Regions and Git Connection Endpoints (p. 278).
3. Choose the repository you want to connect to from the list. This opens the Code page for that repository.

   If you see a Welcome page instead of a list of repositories, there are no repositories associated with your AWS account. To create a repository, see the section called “Create a Repository” (p. 76) or follow the steps in the Git with AWS CodeCommit Tutorial (p. 58) tutorial.

4. Choose Clone URL, and then copy the SSH URL.
5. In the Bash emulator, using the SSH URL you just copied, run the git clone command to clone the repository. This command will create the local repo in a subdirectory of the directory where you run the command. For example, to clone a repository named MyDemoRepo to a local repo named my-demo-repo in the US East (Ohio) region:

   ```
   git clone ssh://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo my-demo-repo
   ```

   Alternatively, open a command prompt, and using the URL and the SSH key ID for the public key you uploaded to IAM, run the git clone command. The local repo will be created in a subdirectory of the directory where you run the command. For example, to clone a repository named MyDemoRepo to a local repo named my-demo-repo:

   ```
   git clone ssh://Your-SSH-Key-ID@git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo my-demo-repo
   ```

   For more information, see Connect to the AWS CodeCommit Repository by Cloning the Repository (p. 79) and Create a Commit (p. 164).

Next Steps

You have completed the prerequisites. Follow the steps in AWS CodeCommit Tutorial (p. 44) to start using AWS CodeCommit.
Setup Steps for HTTPS Connections to AWS CodeCommit Repositories on Linux, macOS, or Unix with the AWS CLI Credential Helper

Before you can connect to AWS CodeCommit for the first time, you must complete the initial configuration steps. This topic walks you through the steps to set up your computer and AWS profile, connect to an AWS CodeCommit repository, and clone that repository to your computer, also known as creating a local repo. If you’re new to Git, you might also want to review the information in Where Can I Learn More About Git? (p. 3).

Topics
- Step 1: Initial Configuration for AWS CodeCommit (p. 35)
- Step 2: Install Git (p. 37)
- Step 3: Set Up the Credential Helper (p. 37)
- Step 4: Connect to the AWS CodeCommit Console and Clone the Repository (p. 38)
- Next Steps (p. 39)

Step 1: Initial Configuration for AWS CodeCommit

Follow these steps to set up an AWS account, create and configure an IAM user, and install the AWS CLI.

To create and configure an IAM user for accessing AWS CodeCommit

2. Create an IAM user, or use an existing one, in your AWS account. Make sure you have an access key ID and a secret access key associated with that IAM user. For more information, see Creating an IAM User in Your AWS Account.

   Note
   AWS CodeCommit requires AWS Key Management Service. If you are using an existing IAM user, make sure there are no policies attached to the user that expressly deny the AWS KMS actions required by AWS CodeCommit. For more information, see AWS KMS and Encryption (p. 288).
3. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
4. In the IAM console, in the navigation pane, choose Users, and then choose the IAM user you want to configure for AWS CodeCommit access.
5. On the Permissions tab, choose Add Permissions.
6. In Grant permissions, choose Attach existing policies directly.
7. Select AWSCodeCommitFullAccess from the list of policies, or another managed policy for AWS CodeCommit access. For more information about managed policies for AWS CodeCommit, see AWS Managed (Predefined) Policies for AWS CodeCommit (p. 258).

   After you have selected the policy you want to attach, choose Next: Review to review the list of policies that will be attached to the IAM user. If the list is correct, choose Add permissions.

   For more information about AWS CodeCommit managed policies and sharing access to repositories with other groups and users, see Share a Repository (p. 80) and Authentication and Access Control for AWS CodeCommit (p. 250).
To install and configure the AWS CLI

1. On your local machine, download and install the AWS CLI. This is a prerequisite for interacting with AWS CodeCommit from the command line. For more information, see Getting Set Up with the AWS Command Line Interface.

   **Note**
   AWS CodeCommit works only with AWS CLI versions 1.7.38 and later. To determine which version of the AWS CLI you have installed, run the `aws --version` command.
   To upgrade an older version of the AWS CLI to the latest version, see Installing the AWS Command Line Interface.

2. Run this command to verify the AWS CodeCommit commands for the AWS CLI are installed:
   
   ```
   aws codecommit help
   ```

   This command should return a list of AWS CodeCommit commands.

3. Configure the AWS CLI with the `configure` command, as follows:

   ```
   aws configure
   ```

   When prompted, specify the AWS access key and AWS secret access key of the IAM user you will use with AWS CodeCommit. Also, be sure to specify the region where the repository exists, such as `us-east-2`. When prompted for the default output format, specify `json`. For example:

   | AWS Access Key ID [None]: Type your target AWS access key ID here, and then press Enter |
   | AWS Secret Access Key [None]: Type your target AWS secret access key here, and then press Enter |
   | Default region name [None]: Type a supported region for AWS CodeCommit here, and then press Enter |
   | Default output format [None]: Type json here, and then press Enter |

   To connect to a repository or a resource in another region, you must re-configure the AWS CLI with the default region name for that region. Supported default region names for AWS CodeCommit include:

   - `us-east-2`
   - `us-east-1`
   - `eu-west-1`
   - `us-west-2`
   - `ap-northeast-1`
   - `ap-southeast-1`
   - `ap-southeast-2`
   - `eu-central-1`
   - `ap-northeast-2`
   - `sa-east-1`
   - `us-west-1`
   - `eu-west-2`
   - `ap-south-1`
   - `ca-central-1`
Step 2: Install Git

To work with files, commits, and other information in AWS CodeCommit repositories, you must install Git on your local machine. AWS CodeCommit supports Git versions 1.7.9 and later.

To install Git, we recommend websites such as Git Downloads.

**Note**
Git is an evolving, regularly updated platform. Occasionally, a feature change might affect the way it works with AWS CodeCommit. If you encounter issues with a specific version of Git and AWS CodeCommit, review the information in Troubleshooting (p. 235).

Step 3: Set Up the Credential Helper

1. From the terminal, use Git to run `git config`, specifying the use of the Git credential helper with the AWS credential profile, and enabling the Git credential helper to send the path to repositories:

   ```
   git config --global credential.helper '!aws codecommit credential-helper $@'
   git config --global credential.UseHttpPath true
   ```

   **Tip**
   The credential helper will use the default AWS credential profile or the Amazon EC2 instance role. You can specify a profile to use, such as CodeCommitProfile, if you have created a specific AWS credential profile to use with AWS CodeCommit:

   ```
   git config --global credential.helper '!aws --profile CodeCommitProfile codecommit credential-helper $@'
   ```

   If your profile name contains spaces, make sure you enclose the name in quotation marks (`"`).

   You can configure profiles per repository instead of globally by using `--local` instead of `--global`.

   The Git credential helper writes the following value to `~/.gitconfig`:

   ```
   [credential]
   helper = !aws --profile CodeCommitProfile codecommit credential-helper @@
   UseHttpPath = true
   ```

   **Important**
   If you want to use a different IAM user on the same local machine for AWS CodeCommit, you must run the `git config` command again and specify a different AWS credential profile.

2. Run `git config --global --edit` to verify the preceding value has been written to `~/.gitconfig`. If successful, you should see the preceding value (in addition to values that may already exist in the Git global configuration file). To exit, typically you would type `:q`, and then press Enter.

   If you experience problems after you configure your credential helper, see Troubleshooting (p. 235).
Important
If you are using macOS, use the following steps to ensure the credential helper is configured correctly.

3. If you are using macOS, use HTTPS to connect to an AWS CodeCommit repository (p. 78). After you connect to an AWS CodeCommit repository with HTTPS for the first time, subsequent access will fail after about fifteen minutes. The default Git version on macOS uses the Keychain Access utility to store credentials. For security measures, the password generated for access to your AWS CodeCommit repository is temporary, so the credentials stored in the keychain will stop working after about 15 minutes. To prevent these expired credentials from being used, you must either:

- Install a version of Git that does not use the keychain by default.
- Configure the Keychain Access utility to not provide credentials for AWS CodeCommit repositories.

1. Open the Keychain Access utility. (You can use Finder to locate it.)
2. Search for git-codecommit.us-east-2.amazonaws.com. Highlight the row, open the context menu or right-click it, and then choose Get Info.
3. Choose the Access Control tab.
4. In Confirm before allowing access, choose git-credential-osxkeychain, and then choose the minus sign to remove it from the list.

Note
After removing git-credential-osxkeychain from the list, you will see a pop-up dialog whenever you run a Git command. Choose Deny to continue. If you find the pop-ups too disruptive, here are some alternate options:

- Connect to AWS CodeCommit using SSH instead of HTTPS. For more information, see For SSH Connections on Linux, macOS, or Unix (p. 26).
- In the Keychain Access utility, on the Access Control tab for git-codecommit.us-east-2.amazonaws.com, choose the Allow all applications to access this item (access to this item is not restricted) option. This will prevent the pop-ups, but the credentials will eventually expire (on average, this takes about 15 minutes) and you will see a 403 error message. When this happens, you must delete the keychain item in order to restore functionality.
- Install a version of Git that does not use the keychain by default.

Step 4: Connect to the AWS CodeCommit Console and Clone the Repository

If an administrator has already sent you the name and connection details for the AWS CodeCommit repository, you can skip this step and clone the repository directly.

To connect to an AWS CodeCommit repository

2. In the region selector, choose the region where the repository was created. Repositories are specific to an AWS region. For more information, see Regions and Git Connection Endpoints (p. 278).
3. Choose the repository you want to connect to from the list. This opens the Code page for that repository.

   If you see a Welcome page instead of a list of repositories, there are no repositories associated with your AWS account. To create a repository, see the section called “Create a Repository” (p. 76) or follow the steps in the Git with AWS CodeCommit Tutorial (p. 58) tutorial.
4. Copy the HTTPS URL to use when connecting to the repository.
5. Open a terminal and from the /tmp directory, use the URL to clone the repository with the `git clone` command. For example, to clone a repository named `MyDemoRepo` to a local repo named `my-demo-repo` in the US East (Ohio) region:

```
```

For more information, see Connect to the AWS CodeCommit Repository by Cloning the Repository (p. 79) and Create a Commit (p. 164).

Next Steps

You have completed the prerequisites. Follow the steps in AWS CodeCommit Tutorial (p. 44) to start using AWS CodeCommit.

Setup Steps for HTTPS Connections to AWS CodeCommit Repositories on Windows with the AWS CLI Credential Helper

Before you can connect to AWS CodeCommit for the first time, you must complete the initial configuration steps. For most users, this can be done most easily by following the steps in For HTTPS Users Using Git Credentials (p. 5). However, if you want to connect to AWS CodeCommit using a root account, federated access, or temporary credentials, you must use the credential helper that is included in the AWS CLI.

This topic walks you through the steps to install the AWS CLI, set up your computer and AWS profile, connect to an AWS CodeCommit repository, and clone that repository to your computer, also known as creating a local repo. If you’re new to Git, you might also want to review the information in Where Can I Learn More About Git? (p. 3).

Topics

- Step 1: Initial Configuration for AWS CodeCommit (p. 39)
- Step 2: Install Git (p. 41)
- Step 3: Set Up the Credential Helper (p. 41)
- Step 4: Connect to the AWS CodeCommit Console and Clone the Repository (p. 43)
- Next Steps (p. 43)

Step 1: Initial Configuration for AWS CodeCommit

Follow these steps to set up an AWS account, create and configure an IAM user, and install the AWS CLI. The AWS CLI includes a credential helper that you will configure for HTTPS connections to your AWS CodeCommit repositories.

To create and configure an IAM user for accessing AWS CodeCommit

2. Create an IAM user, or use an existing one, in your AWS account. Make sure you have an access key ID and a secret access key associated with that IAM user. For more information, see Creating an IAM User in Your AWS Account.
Step 1: Initial Configuration for AWS CodeCommit

Note
AWS CodeCommit requires AWS Key Management Service. If you are using an existing
IAM user, make sure there are no policies attached to the user that expressly deny the
AWS KMS actions required by AWS CodeCommit. For more information, see AWS KMS and
Encryption (p. 288).

3. Sign in to the AWS Management Console and open the IAM console at https://
console.aws.amazon.com/iam/.

4. In the IAM console, in the navigation pane, choose Users, and then choose the IAM user you want to
configure for AWS CodeCommit access.

5. On the Permissions tab, choose Add Permissions.

6. In Grant permissions, choose Attach existing policies directly.

7. Select AWSCodeCommitFullAccess from the list of policies, or another managed policy for AWS
CodeCommit access. For more information about managed policies for AWS CodeCommit, see AWS
Managed (Predefined) Policies for AWS CodeCommit (p. 258).

After you have selected the policy you want to attach, choose Next: Review to review the list of
policies that will be attached to the IAM user. If the list is correct, choose Add permissions.

For more information about AWS CodeCommit managed policies and sharing access to repositories
with other groups and users, see Share a Repository (p. 80) and Authentication and Access
Control for AWS CodeCommit (p. 250).

To install and configure the AWS CLI

1. On your local machine, download and install the AWS CLI. This is a prerequisite for interacting with
AWS CodeCommit from the command line. For more information, see Getting Set Up with the AWS
Command Line Interface.

   Note
AWS CodeCommit works only with AWS CLI versions 1.7.38 and later. To determine which
version of the AWS CLI you have installed, run the aws --version command.
To upgrade an older version of the AWS CLI to the latest version, see Installing the AWS
Command Line Interface.

2. Run this command to verify the AWS CodeCommit commands for the AWS CLI are installed:

   ```
   aws codecommit help
   ```

   This command should return a list of AWS CodeCommit commands.

3. Configure the AWS CLI with the configure command, as follows:

   ```
   aws configure
   ```

   When prompted, specify the AWS access key and AWS secret access key of the IAM user you will use
with AWS CodeCommit. Also, be sure to specify the region where the repository exists, such as us-
east-2. When prompted for the default output format, specify json. For example:

   ```
   AWS Access Key ID [None]: Type your target AWS access key ID here, and then press Enter
   AWS Secret Access Key [None]: Type your target AWS secret access key here, and then press Enter
   Default region name [None]: Type a supported region for AWS CodeCommit here, and then press Enter
   Default output format [None]: Type json here, and then press Enter
   ```
To connect to a repository or a resource in another region, you must re-configure the AWS CLI with the default region name for that region. Supported default region names for AWS CodeCommit include:

- us-east-2
- us-east-1
- eu-west-1
- us-west-2
- ap-northeast-1
- ap-southeast-1
- ap-southeast-2
- eu-central-1
- ap-northeast-2
- sa-east-1
- us-west-1
- eu-west-2
- ap-south-1
- ca-central-1

For more information about AWS CodeCommit and regions, see Regions and Git Connection Endpoints (p. 278). For more information about IAM, access keys, and secret keys, see How Do I Get Credentials? and Managing Access Keys for IAM Users.

### Step 2: Install Git

To work with files, commits, and other information in AWS CodeCommit repositories, you must install Git on your local machine. AWS CodeCommit supports Git versions 1.7.9 and later.

To install Git, we recommend websites such as Git for Windows. If you use this link to install Git, you can accept all of the installation default settings except for the following:

- When prompted during the Adjusting your PATH environment step, select the Use Git from the Windows Command Prompt option.
- (Optional) If you intend to use HTTPS with the credential helper that is included in the AWS CLI instead of configuring Git credentials for AWS CodeCommit, on the Configuring extra options page, make sure the Enable Git Credential Manager option is cleared. The Git Credential Manager is only compatible with AWS CodeCommit if IAM users configure Git credentials. For more information, see For HTTPS Users Using Git Credentials (p. 5) and Git for Windows: I Installed Git for Windows, but I Am Denied Access to My Repository (403) (p. 242).

**Note**

Git is an evolving, regularly updated platform. Occasionally, a feature change might affect the way it works with AWS CodeCommit. If you encounter issues with a specific version of Git and AWS CodeCommit, review the information in Troubleshooting (p. 235).

### Step 3: Set Up the Credential Helper

The AWS CLI includes a Git credential helper you can use with AWS CodeCommit. The Git credential helper requires an AWS credential profile, which stores a copy of an IAM user’s AWS access key ID and AWS secret access key (along with a default region name and default output format). The Git credential
helper uses this information to automatically authenticate with AWS CodeCommit so you don't need to type this information every time you use Git to interact with AWS CodeCommit.

1. Open a command prompt and use Git to run `git config`, specifying the use of the Git credential helper with the AWS credential profile, which enables the Git credential helper to send the path to repositories:

```bash
    git config --global credential.helper "!aws codecommit credential-helper $@"
    git config --global credential.UseHttpPath true
```

The Git credential helper writes the following to the .gitconfig file:

```
    [credential]
    helper = !aws codecommit credential-helper $@
    UseHttpPath = true
```

**Important**

- If you are using a Bash emulator instead of the Windows command line, you must use single quotes instead of double quotes.

- The credential helper will use the default AWS profile or the Amazon EC2 instance role. If you have created an AWS credential profile to use, such as `CodeCommitProfile`, you can modify the command as follows to use it instead:

```bash
    git config --global credential.helper "!aws codecommit credential-helper --profile CodeCommitProfile $@"
```

This will write the following to the .gitconfig file:

```
    [credential]
    helper = !aws codecommit credential-helper --profile=CodeCommitProfile $@
    UseHttpPath = true
```

- If your profile name contains spaces, you must edit your .gitconfig file after you run this command to enclose it in single quotes (''); otherwise, the credential helper will not work.

- If your installation of Git for Windows included the Git Credential Manager utility, you will see 403 errors or prompts to provide credentials into the Credential Manager utility after the first few connection attempts. The most reliable way to solve this problem is to uninstall and then reinstall Git for Windows without the option for the Git Credential Manager utility, as it is not compatible with AWS CodeCommit. If you want to keep the Git Credential Manager utility, you must perform additional configuration steps to also use AWS CodeCommit, including manually modifying the .gitconfig file to specify the use of the credential helper for AWS CodeCommit when connecting to AWS CodeCommit. Remove any stored credentials from the Credential Manager utility (you can find this utility in Control Panel). Once you have removed any stored credentials, add the following to your .gitconfig file, save it, and then try connecting again from a new command prompt window:

```bash
    [credential "https://git-codecommit.us-east-2.amazonaws.com"]
      helper = !aws codecommit credential-helper $@
      UseHttpPath = true

    [credential "https://git-codecommit.us-east-1.amazonaws.com"]
      helper = !aws codecommit credential-helper $@
      UseHttpPath = true
```
Step 4: Connect to the AWS CodeCommit Console and Clone the Repository

If an administrator has already sent you the name and connection details for the AWS CodeCommit repository, you can skip this step and clone the repository directly.

To connect to an AWS CodeCommit repository

2. In the region selector, choose the region where the repository was created. Repositories are specific to an AWS region. For more information, see Regions and Git Connection Endpoints (p. 278).
3. Choose the repository you want to connect to from the list. This opens the Code page for that repository.

   If you see a Welcome page instead of a list of repositories, there are no repositories associated with your AWS account. To create a repository, see the section called “Create a Repository” (p. 76) or follow the steps in the Git with AWS CodeCommit Tutorial (p. 58) tutorial.

4. Copy the HTTPS URL to use when connecting to the repository.
5. Open a command prompt and use the URL to clone the repository with the git clone command. The local repo will be created in a subdirectory of the directory where you run the command. For example, to clone a repository named MyDemoRepo to a local repo named my-demo-repo in the US East (Ohio) region:

   ```bash
   ```

   On some versions of Windows, you might see a pop-up dialog box asking for your user name and password. This is the built-in credential management system for Windows, but it is not compatible with the credential helper for AWS CodeCommit. Choose Cancel.

   For more information about how to connect to repositories, see Connect to the AWS CodeCommit Repository by Cloning the Repository (p. 79).

Next Steps

You have completed the prerequisites. Follow the steps in AWS CodeCommit Tutorial (p. 44) to start using AWS CodeCommit.
Getting Started with AWS CodeCommit

The easiest way to get started with AWS CodeCommit is to follow the steps in AWS CodeCommit Tutorial (p. 44). If you are new to Git as well as AWS CodeCommit, you should also consider following the steps in Git with AWS CodeCommit Tutorial (p. 58). This will help you familiarize yourself with AWS CodeCommit as well as the basics of using Git when interacting with your AWS CodeCommit repositories.

You can also follow the tutorial in Simple Pipeline Walkthrough with AWS CodePipeline and AWS CodeCommit to learn how to use your AWS CodeCommit repository as part of a continuous delivery pipeline.

The tutorials in this section assume you have completed the prerequisites and setup (p. 4), including:

- Assigning permissions to the IAM user.
- Setting up credential management for HTTPS or SSH connections on the local machine you will use for this tutorial.
- Configuring the AWS CLI if you want to use the command line or terminal for all operations, including creating the repository.

Topics

- Getting Started with AWS CodeCommit Tutorial (p. 44)
- Git with AWS CodeCommit Tutorial (p. 58)

Getting Started with AWS CodeCommit Tutorial

If you’re new to AWS CodeCommit, this tutorial helps you learn how to use its features. In this tutorial, you create a repository in AWS CodeCommit. After you create a local copy of that repository (a local repo) and push some changes to the AWS CodeCommit repository, you browse the files and view the changes. You can also create a pull request for others to review and comment on changes to your code.

If you are not familiar with Git, you might want to complete the Git with AWS CodeCommit Tutorial (p. 58) in addition to this tutorial. After you finish this tutorial, you should have enough practice to start using AWS CodeCommit for your own projects and in team environments.

Important
Before you begin, you must complete the prerequisites and setup (p. 4), including:

- Assigning permissions to the IAM user.
- Setting up credential management for HTTPS or SSH connections on the local machine you use for this tutorial.
- Configuring the AWS CLI if you want to use the command line or terminal for all operations, including creating the repository.
Step 1: Create an AWS CodeCommit Repository

You can use the AWS CodeCommit console to create an AWS CodeCommit repository. If you already have a repository you want to use for this tutorial, you can skip this step.

**Note**
Depending on your usage, you might be charged for creating or accessing a repository. For more information, see Pricing on the AWS CodeCommit product information page.

To create the AWS CodeCommit repository (console)

2. In the region selector, choose the region where you will create the repository. For more information, see Regions and Git Connection Endpoints (p. 278).
3. On the Welcome page, choose Get Started Now. (If a Dashboard page appears instead, choose Create repository.)
4. On the Create repository page, in the Repository name box, type a name (for example, MyDemoRepo).
5. In Description, type a description (for example, My demonstration repository).
6. Choose Create repository.
7. In Configure email notifications, configure notifications so that repository users receive emails about important repository events. This step is optional, but recommended. You can choose the event types (for example, comments on code) and whether to use an existing Amazon SNS topic or create one specifically for this purpose. You can choose to skip this step and configure notifications at a later time. For more information, see Configuring Notifications for Events in an AWS CodeCommit Repository (p. 84).
Step 2: Add Files to Your Repository

You can add files to your repository in the following ways:

- Creating a file directly from the AWS CodeCommit console.
- Uploading a file from your local computer using the AWS CodeCommit console.
- Using a Git client to clone the repository to your local computer, and then adding, committing, and pushing files to the AWS CodeCommit repository.

The simplest way to get started is to add a file from the AWS CodeCommit console.

1. In the navigation bar for the repository, choose Code.
2. Choose **Add file**, and then choose to create a file or upload a file from your computer.

3. In **Code: Create a file** or **Code: Upload a file**, in **Author name**, type the name you want displayed to other repository users. In **Commit message**, type a brief message. (Optional) Provide an email address in **Email address**. If you're uploading a file, choose the file you want to upload. If you're creating a file, in the code editor, type a name for the file and the code you want to add.

4. Choose **Commit file**.

For more information, see Working with Files in AWS CodeCommit Repositories (p. 130).

To use a Git client to clone the repository, install Git on your local computer, and then clone the AWS CodeCommit repository. Add some files to the local repo and push them to the AWS CodeCommit repository. For an in-depth introduction, try the Git with AWS CodeCommit Tutorial (p. 58). If you are familiar with Git but are not sure how to do this with an AWS CodeCommit repository, you can view examples and instructions in Create a Commit (p. 164), Step 2: Create a Local Repo (p. 60), or Connect to a Repository (p. 78).

After you have added some files to the AWS CodeCommit repository, you can view them from the console.
Step 3: Browse the Contents of Your Repository

You can use the AWS CodeCommit console to review the files in a repository or to quickly read the contents of a file. This helps you determine which branch to check out or whether to create a local copy of a repository.

1. From the AWS CodeCommit console, choose MyDemoRepo from the list of repositories.

2. The contents of the repository are displayed in the default branch for your repository. To change the view to another branch, choose the view selector button (here the view is set to Branch: master), and then choose the branch you want to view from the list.

3. To view the contents of a file in your repository, choose the file from the list.
Step 3: Browse the Contents of Your Repository

For more information, see Browse Files in a Repository (p. 131).

You can also browse the commit history of a repository. This helps you identify changes made in a repository, including when and by whom those changes were made.

1. In the navigation pane for a repository, choose **Commits**. In the commit history view, a history of commits for the repository in the default branch is displayed, in reverse chronological order.
2. Review the commit history by branch (p. 194) or by tag (p. 190), and get details about commits by author, date, and more.

3. To view the differences between a commit and its parent, choose the abbreviated commit ID. You can choose how the changes are displayed, including showing or hiding white space changes, and whether to view changes inline (Unified view) or side by side (Split view).

   **Note**
   If you are signed in as an IAM user, you can configure and save your preferences for viewing code and other console settings. For more information, see Working with User Preferences (p. 208).

4. To view all comments on a commit, choose **Comments**, or scroll through the changes to view them inline. You can also add your own comments and reply to the comments made by others.
For more information, see Comment on a Commit (p. 179).

5. To view the differences between any two commits specifiers, including tags, branches, and commit IDs, in the navigation pane, choose Compare.
For more information, see Browse the Commit History of a Repository (p. 166) and Compare Commits (p. 173).

6. In the navigation pane, choose **Visualizer**.

The commit graph is displayed, with the subject line for each commit shown next to its point in the graph. The subject line display is limited to 80 characters.

7. To see more details about a commit, choose its point in the graph.
Step 4: Create and Collaborate on a Pull Request

When you work with other repository users, you might want to collaborate on code and review changes. You can create a pull request so that other users can review and comment on your code changes in a branch before you merge those changes into another branch.

**Important**
Before you can create a pull request, you must create a branch that contains the code changes you want to review. For more information, see Create a Branch (p. 195).

1. In the navigation pane, choose **Branches**.
2. In **Branches**, find the branch that contains the changes you want reviewed, and then choose **Create pull request**.
3. In Create pull request, in Source, choose the branch that contains the changes you want reviewed. By default, the destination branch is preconfigured for you with the default branch of the repository. (Optional) Choose a different branch if you want to merge code into a different branch when the pull request is complete, and then choose Compare.

4. Review the merge details and changes to confirm that the pull request contains the changes and commits you want reviewed. If so, in Details, provide a title for this review. This is the title that appears in the list of pull requests for the repository. In Description, provide details about what this review is about and any other useful information for reviewers. Choose Create.
5. Your pull request appears in the list of pull requests for the repository. You can filter the view to show only open requests, closed requests, requests that you created, and more.
Tip
You can create pull requests directly from the Pull requests view.

6. If you configured notifications for your repository and chose to notify users of pull request events, users receive email about your new pull request. Users can view the changes and comment on specific lines of code, files, and the pull request itself. They can also reply to comments. If necessary, you can push changes to the pull request branch, which updates the pull request.
7. When you are satisfied that all the code changes have been reviewed and agreed to, from the pull request, do one of the following:

- If you want to automatically merge the branches as part of closing the pull request using the fast-forward merge option, choose **Merge**.
- If you want to close the pull request without using the automatic fast-forward merge option to merge branches, or if there are merge conflicts in the branches that cannot be automatically resolved, choose **Close pull request**.

**Note**
You can always manually merge branches, including pull request branches, by using the `git merge` command in your local repo and pushing your changes. AWS CodeCommit closes the pull request when you push the merged code.
For more information, see Working with Pull Requests (p. 138).

**Step 5: Next Steps**

Now that you have familiarized yourself with AWS CodeCommit and some of its features, consider doing the following:

- If you are new to Git and AWS CodeCommit or want to review examples of using Git with AWS CodeCommit, continue to the Git with AWS CodeCommit Tutorial (p. 58) tutorial.
- If you want to work with others in an AWS CodeCommit repository, see Share a Repository (p. 80). (If you want to share your repository with users in another AWS account, see Configure Cross-Account Access to an AWS CodeCommit Repository (p. 120).)
- If you want to migrate a repository to AWS CodeCommit, follow the steps in Migrate to AWS CodeCommit (p. 211).
- If you want to add your repository to a continuous delivery pipeline, follow the steps in Simple Pipeline Walkthrough.
- If you want to learn more about products and services that integrate with AWS CodeCommit, including examples from the community, see Product and Service Integrations (p. 68).

**Step 6: Clean Up**

If you are done with exploring AWS CodeCommit and no longer need the repository, you should delete the AWS CodeCommit repository and other resources you used in this tutorial so you won’t continue to be charged for the storage space.

**Important**

After you delete this repository, you can no longer clone it to any local repo or shared repo. You also can no longer pull data from it, push data to it, or perform any Git operations, from any local repo or shared repo. This action cannot be undone.

If you configured notifications for your repository, deleting the repository also deletes the Amazon CloudWatch Events rule created for the repository. It does not delete the Amazon SNS topic used as a target for that rule.

If you configured triggers for your repository, deleting the repository does not delete the Amazon SNS topics or Lambda functions you configured as the targets of those triggers. Be sure to delete those resources if you don't need them. For more information, see Delete Triggers from a Repository (p. 108).

To delete the AWS CodeCommit repository

2. On the Dashboard page, in the list of repositories, choose MyDemoRepo.
3. In the navigation pane, choose Settings.
4. On the Settings page, in General, in Delete repository, choose Delete repository.
5. In the box next to Type the name of the repository to confirm deletion, type MyDemoRepo, and then choose Delete.

**Git with AWS CodeCommit Tutorial**

If you are new to Git and AWS CodeCommit, this tutorial helps you learn some simple commands to get you started. If you are already familiar with Git, you can skip this tutorial and go to AWS CodeCommit Tutorial (p. 44).
In this tutorial, you create a repository that represents a local copy of the AWS CodeCommit repository, which we refer to here and in the rest of the documentation as a local repo.

After you create the local repo, you make some changes to it. Then you send (push) your changes to the AWS CodeCommit repository.

You also simulate a team environment where two users independently commit changes to their local repo and push those changes to the AWS CodeCommit repository. The users then pull the changes from the AWS CodeCommit repository to their own local repo to see the changes the other user made.

You also create branches and tags and manage some access permissions in the AWS CodeCommit repository.

After you complete this tutorial, you should have enough practice with the core Git and AWS CodeCommit concepts to use them for your own projects.

Complete the prerequisites and setup (p. 4), including:

- Assign permissions to the IAM user.
- Set up credential management for HTTPS or SSH connections on the local machine you use for this tutorial.
- Configure the AWS CLI if you want to use the command line or terminal for all operations, including creating the repository.

Topics

- Step 1: Create an AWS CodeCommit Repository (p. 59)
- Step 2: Create a Local Repo (p. 60)
- Step 3: Create Your First Commit (p. 60)
- Step 4: Push Your First Commit (p. 61)
- Step 5: Share the AWS CodeCommit Repository and Push and Pull Another Commit (p. 61)
- Step 6: Create and Share a Branch (p. 63)
- Step 7: Create and Share a Tag (p. 64)
- Step 8: Set Up Access Permissions (p. 65)
- Step 9: Clean Up (p. 67)

Step 1: Create an AWS CodeCommit Repository

In this step, you use the AWS CodeCommit console to create the repository.

You can skip this step if you already have an AWS CodeCommit repository you want to use.

Note
Depending on your usage, you might be charged for creating or accessing a repository. For more information, see Pricing on the AWS CodeCommit product information page.

To create the AWS CodeCommit repository (console)

2. In the region selector, choose the region where you will create the repository. For more information, see Regions and Git Connection Endpoints (p. 278).
3. On the Welcome page, choose Get Started Now. (If a Dashboard page appears instead, choose Create repository.)
4. On the **Create repository** page, in the **Repository name** box, type a name (for example, **MyDemoRepo**).

5. In **Description**, type a description (for example, **My demonstration repository**).

6. Choose **Create repository**.

7. In **Configure email notifications**, configure notifications so that repository users receive emails about important repository events. This step is optional, but recommended. You can choose the event types (for example, comments on code) and whether to use an existing Amazon SNS topic or create one specifically for this purpose. You can choose to skip this step and configure notifications at a later time. For more information, see **Configuring Notifications for Events in an AWS CodeCommit Repository** (p. 84).

**Note**
The remaining steps in this tutorial use **MyDemoRepo** for the name of your AWS CodeCommit repository. If you choose a different name, be sure to use it throughout this tutorial.

For more information about creating repositories, including how to create a repository from the terminal or command line, see **Create a Repository** (p. 76).

### Step 2: Create a Local Repo

In this step, you set up a local repo on your local machine to connect to your repository. To do this, you select a directory on your local machine that represents the local repo. You use Git to clone and initialize a copy of your empty AWS CodeCommit repository inside of that directory. Then you specify the user name and email address used to annotate your commits.


2. In the region selector, choose the region where the repository was created. Repositories are specific to an AWS region. For more information, see **Regions and Git Connection Endpoints** (p. 278).

3. On the **Dashboard** page, choose the name of the repository you want to share.

4. On the **Code** page, choose **Clone URL**, and then choose the protocol you want your users to use.

5. Copy the displayed URL for the connection protocol your users will use when connecting to your AWS CodeCommit repository.

6. Send your users the connection information along with any other instructions, such as installing the AWS CLI, configuring a profile, or installing Git. Make sure to include the configuration information for the connection protocol (for example, for HTTPS, configuring the credential helper for Git).

### Step 3: Create Your First Commit

In this step, you create your first commit in your local repo. To do this, you create two example files in your local repo. You use Git to stage the change to, and then commit the change to, your local repo.

1. Use a text editor to create the following two example text files in your directory. Name the files **cat.txt** and **dog.txt**:

   ```
   cat.txt
   -------
   The domestic cat (Felis catus or Felis silvestris catus) is a small, usually furry, domesticated, and carnivorous mammal.
   
   dog.txt
   -------
   
   API Version 2015-04-13
   60
   ```
Step 4: Push Your First Commit

In this step, you push the commit from your local repo to your AWS CodeCommit repository.

Run `git push` to push your commit through the default remote name Git uses for your AWS CodeCommit repository (origin), from the default branch in your local repo (master):

```
$ git push -u origin master
```

**Tip**
To see details about the commit you just made, run `git log`.

Step 5: Share the AWS CodeCommit Repository and Push and Pull Another Commit

In this step, you share information about the AWS CodeCommit repository with a fellow team member. The team member uses this information to get a local copy, make some changes to it, and then push the modified local copy to your AWS CodeCommit repository. You then pull the changes from the AWS CodeCommit repository to your local repo.

In this tutorial, you simulate the fellow user by having Git create a directory separate from the one you created in step 2 (p. 60). (Typically, this directory would be on a different machine.) This new directory is a copy of your AWS CodeCommit repository. Any changes you make to the existing directory or this new directory are made independently. The only way to identify changes to these directories is to pull from the AWS CodeCommit repository.

Even though they're on the same local machine, we call the existing directory your `local repo` and the new directory the `shared repo`.

From the new directory, you get a separate copy of the AWS CodeCommit repository. You then add a new example file, commit the changes to the shared repo, and then push the commit from the shared repo to your AWS CodeCommit repository.

Lastly, you pull the changes from your repository to your local repo and then browse it to see the changes committed by the other user.

1. Switch to the `/tmp` directory or the `c:\temp` directory.
2. Run `git clone` to pull down a copy of the repository into the shared repo:
For HTTPS:

```bash
```

For SSH:

```bash
git clone ssh://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo shared-demo-repo
```

**Note**
When you clone a repository using SSH on Windows operating systems, you must add the SSH key ID to the connection string as follows:

```bash
git clone ssh://Your-SSH-Key-ID@git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo my-demo-repo
```

For more information, see For SSH Connections on Windows (p. 30).

In this command, `MyDemoRepo` is the name of your AWS CodeCommit repository. `shared-demo-repo` is the name of the directory Git creates in the `/tmp` directory or the `c:\temp` directory. After Git creates the directory, Git pulls down a copy of your repository into the `shared-demo-repo` directory.

3. Switch to the `shared-demo-repo` directory:

   (For Linux, macOS, or Unix) `cd /tmp/shared-demo-repo`
   (For Windows) `cd c:\temp\shared-demo-repo`

4. Run `git config` to add another user name and email address represented by placeholders `other-user-name` and `other-email-address` (for example, John Doe and johndoe@example.com). This makes it easier to identify the commits the other user made:

   ```bash
git config --local user.name "other-user-name"
git config --local user.email other-email-address
```

5. Use a text editor to create the following example text file in the `shared-demo-repo` directory. Name the file `horse.txt`:

   ```
horse.txt
-------
The horse (Equus ferus caballus) is one of two extant subspecies of Equus ferus.
```

6. Run `git add` to stage the change to the shared repo:

   ```bash
git add horse.txt
```

7. Run `git commit` to commit the change to the shared repo:

   ```bash
git commit -m "Added horse.txt"
```

8. Run `git push` to push your initial commit through the default remote name Git uses for your AWS CodeCommit repository (origin), from the default branch in your local repo (master):

   ```bash
git push -u origin master
```
9. Switch to your local repo and run `git pull` to pull into your local repo the commit the shared repo made to the AWS CodeCommit repository. Then run `git log` to see the commit that was initiated from the shared repo.

**Step 6: Create and Share a Branch**

In this step, you create a branch in your local repo, make a few changes, and then push the branch to your AWS CodeCommit repository. You then pull the branch to the shared repo from your AWS CodeCommit repository.

A *branch* allows you to independently develop a different version of the repository's contents (for example, to work on a new software feature without affecting the work of your team members). When that feature is stable, you merge the branch into a more stable branch of the software.

You use Git to create the branch and then point it to the first commit you made. You use Git to push the branch to the AWS CodeCommit repository. You then switch to your shared repo and use Git to pull the new branch into your shared local repo and explore the branch.

1. From your local repo, run `git checkout`, specifying the name of the branch (for example, `MyNewBranch`) and the ID of the first commit you made in the local repo.

   If you don't know the commit ID, run `git log` to get it. Make sure the commit has your user name and email address, not the user name and email address of the other user. This is to simulate that `master` is a stable version of the AWS CodeCommit repository and the `MyNewBranch` branch is for some new, relatively unstable feature:

   ```
   git checkout -b MyNewBranch commit-ID
   ```

2. Run `git push` to send the new branch from the local repo to the AWS CodeCommit repository:

   ```
   git push origin MyNewBranch
   ```

3. Now, pull the branch into the shared repo and check your results:

   1. Switch to the shared repo directory (shared-demo-repo).
   2. Pull in the new branch (`git fetch origin`).
   3. Confirm that the branch has been pulled in (`git branch --all` displays a list of all branches for the repository).
   4. Switch to the new branch (`git checkout MyNewBranch`).
   5. Confirm that you have switched to the `MyNewBranch` branch by running `git status` or `git branch`. The output shows which branch you are on. In this case, it should be `MyNewBranch`.
   6. View the list of commits in the branch (`git log`).

   Here's the list of Git commands to call:

   ```
   git fetch origin
   git branch --all
   git checkout MyNewBranch
   git branch or git status
   git log
   ```

4. Switch back to the `master` branch and view its list of commits. The Git commands should look like this:

   ```
   git checkout master
   ```
Step 7: Create and Share a Tag

In this step, you create two tags in your local repo, associate the tags with commits, and then push the tags to your AWS CodeCommit repository. You then pull the changes from the AWS CodeCommit repository to the shared repo.

A tag is used to give a human-readable name to a commit (or branch or even another tag). You would do this, for example, if you want to tag a commit as "v2.1." A commit, branch, or tag can have any number of tags associated with it, but an individual tag can be associated with only one commit, branch, or tag. In this tutorial, you tag one commit as release and one as beta.

You use Git to create the new tags, pointing the release tag to the first commit you made and the beta tag to the commit made by the other user. You then use Git to push the tags to the AWS CodeCommit repository. Then you switch to your shared repo and use Git to pull the tags into your shared local repo and explore the tags.

1. From your local repo, run `git tag`, specifying the name of the new tag (release) and the ID of the first commit you made in the local repo.

   If you don't know the commit ID, run `git log` to get it. Make sure the commit has your user name and email address, not the user name and email address of the other user. This is to simulate that your commit is a stable version of the AWS CodeCommit repository:

   ```
git tag release commit-ID
```

   Run `git tag` again to tag the commit from the other user with the beta tag. This is to simulate that the commit is for some new, relatively unstable feature:

   ```
git tag beta commit-ID
```

2. Run `git push --tags` to send the tags to the AWS CodeCommit repository.

3. Now pull the tags into the shared repo and check your results:

   1. Switch to the shared repo directory (shared-demo-repo).
   2. Pull in the new tags (`git fetch origin`).
   3. Confirm that the tags have been pulled in (`git tag` displays a list of tags for the repository).
   4. View information about each tag (`git log release` and `git log beta`).

Here's the list of Git commands to call:

```
git fetch origin
git tag
git log release
git log beta
```
4. Try this out in the local repo, too:

```
git log release
git log beta
```

### Step 8: Set Up Access Permissions

In this step, you give a user permission to synchronize the shared repo with the AWS CodeCommit repository. This is an optional step. It's recommended for users who are interested in learning about how to control access to AWS CodeCommit repositories.

To do this, you use the IAM console to create an IAM user, who, by default, does not have permissions to synchronize the shared repo with the AWS CodeCommit repository. You can run `git pull` to verify this. If the new user doesn't have permission to synchronize, the command doesn't work. Then you go back to the IAM console and apply a policy that allows the user to use `git pull`. Again, you can run `git pull` to verify this.

This step assumes you have permissions to create IAM users in your AWS account. If you don’t have these permissions, then you can't perform the procedures in this step. Skip ahead to Step 9: Clean Up (p. 67) to clean up the resources you used for your tutorial.

1. Sign in to the AWS Management Console and open the IAM console at [https://console.aws.amazon.com/iam/](https://console.aws.amazon.com/iam/).

   Be sure to sign in with the same user name and password you used in Setting Up (p. 4).

2. In the navigation pane, choose **Users**, and then choose **Create New Users**.

3. In the first **Enter User Names** box, type an example user name (for example, **JaneDoe-CodeCommit**). Select the **Generate an access key for each user** box, and choose **Create**.

4. Choose **Show User Security Credentials**. Make a note of the access key ID and secret access key or choose **Download Credentials**.

5. Follow the instructions in For HTTPS Users Using Git Credentials (p. 5) to generate and supply the credentials of the IAM user.

   If you want to use SSH, follow the instructions in [SSH and Linux, macOS, or Unix: Set Up the Public and Private Keys for Git and AWS CodeCommit](p. 27) or [SSH and Windows: Set Up the Public and Private Keys for Git and AWS CodeCommit](p. 32) to set up the user with public and private keys.

6. Run `git pull`. The following error should appear:

   For HTTPS:

   ```
   ```

   For SSH:

   ```
fatal: unable to access 'ssh://git-codecommit.us-east-2.amazonaws.com/v1/repos/repository-name/': The requested URL returned error: 403.
   ```

   The error appears because the new user doesn't have permission to synchronize the shared repo with the AWS CodeCommit repository.

7. Return to the IAM console. In the navigation pane, choose **Policies**, and then choose **Create Policy**.

   (If a **Get Started** button appears, choose it, and then choose **Create Policy**.)

8. Next to **Create Your Own Policy**, choose **Select**.

9. In the **Policy Name** box, type a name (for example, **CodeCommitAccess-GettingStarted**).
10. In the **Policy Document** box, type the following, which allows an IAM user to pull from any repository associated with the IAM user:

```json
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "codecommit:GitPull"
         ],
         "Resource": "*"
      }
   ]
}
```

**Tip**
If you want the IAM user to be able to push commits to any repository associated with the IAM user, type this instead:

```json
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "codecommit:GitPull",
            "codecommit:GitPush"
         ],
         "Resource": "*"
      }
   ]
}
```

For information about other AWS CodeCommit action and resource permissions you can give to users, see [Authentication and Access Control for AWS CodeCommit](p. 250).

11. In the navigation pane, choose **Users**.
12. Choose the example user name (for example, **JaneDoe-CodeCommit**) to which you want to attach the policy.
13. Choose the **Permissions** tab.
15. Select the **CodeCommitAccess-GettingStarted** policy you just created, and then choose **Attach Policy**.
16. Run **git pull**. This time the command should work and an **Already up-to-date** message should appear.
17. If you are using HTTPS, switch to your original credentials. For more information, see the instructions in [Step 3: Set Up the Credential Helper](p. 37) or [Step 3: Set Up the Credential Helper](p. 41).

If you are using SSH, switch to your original keys. For more information, see [SSH and Linux, macOS, or Unix: Set Up the Public and Private Keys for Git and AWS CodeCommit](p. 27) or [SSH and Windows: Set Up the Public and Private Keys for Git and AWS CodeCommit](p. 32).

You've reached the end of this tutorial.
Step 9: Clean Up

In this step, you delete the AWS CodeCommit repository you used in this tutorial, so you won't continue to be charged for the storage space.

You also remove the local repo and shared repo on your local machine because they won't be needed after you delete the AWS CodeCommit repository.

**Important**
After you delete this repository, you won't be able to clone it to any local repo or shared repo. You also won't be able to pull data from it, or push data to it, from any local repo or shared repo. This action cannot be undone.

To delete the AWS CodeCommit repository (console)

2. On the **Dashboard** page, in the list of repositories, choose **MyDemoRepo**.
3. In the navigation pane, choose **Settings**.
4. On the **Settings** page, in **Delete repository**, choose **Delete repository**.
5. In the box next to **Type the name of the repository to confirm deletion**, type **MyDemoRepo**, and then choose **Delete**.

To delete the AWS CodeCommit repository (AWS CLI)

Run the **delete-repository (p. 129)** command:

```
aws codecommit delete-repository --repository-name MyDemoRepo
```

To delete the local repo and shared repo

For Linux, macOS, or Unix:

```
cd /tmp
rm -rf /tmp/my-demo-repo
rm -rf /tmp/shared-demo-repo
```

For Windows:

```
cd c:\temp
rd /s /q c:\temp\my-demo-repo
rd /s /q c:\temp\shared-demo-repo
```
Product and Service Integrations with AWS CodeCommit

By default, AWS CodeCommit is integrated with a number of AWS services. You can also use AWS CodeCommit with products and services outside of AWS. The following information can help you configure AWS CodeCommit to integrate with the products and services you use.

**Note**
You can automatically build and deploy commits to an AWS CodeCommit repository by integrating with AWS CodePipeline. To learn more, follow the steps in the AWS for DevOps Getting Started Guide.

**Topics**
- Integration with Other AWS Services (p. 68)
- Integration Examples from the Community (p. 72)

### Integration with Other AWS Services

AWS CodeCommit is integrated with the following AWS services:

| AWS Cloud9 | AWS Cloud9 contains a collection of tools that you use to code, build, run, test, debug, and release software in the cloud. This collection of tools is referred to as the AWS Cloud9 integrated development environment, or IDE.

You access the AWS Cloud9 IDE through a web browser. The IDE offers a rich code-editing experience with support for several programming languages and runtime debuggers, as well as a built-in terminal.

Learn more:
- AWS Cloud9 User Guide
- AWS CodeCommit Sample for AWS Cloud9
- Integrate AWS Cloud9 with AWS CodeCommit (p. 12) |

| Amazon CloudWatch Events | CloudWatch Events delivers a near real-time stream of system events that describe changes in Amazon Web Services (AWS) resources. Using simple rules that you can quickly set up, you can match events and route them to one or more target functions or streams. CloudWatch Events becomes aware of operational changes as they occur. CloudWatch Events responds to these operational changes and takes action as necessary, by sending messages to respond to |
the environment, activating functions, making changes, and capturing state information.

You can configure CloudWatch Events to monitor AWS CodeCommit repositories and respond to repository events by targeting streams, functions, tasks, or other processes in other AWS services, such as Amazon Simple Queue Service, Amazon Kinesis, AWS Lambda, and many more.

Learn more:
- CloudWatch Events User Guide
- AWS CodeCommit Events
- Blog post: Build Serverless AWS CodeCommit Workflows using Amazon CloudWatch Events and JGit

AWS CodeStar

AWS CodeStar is a cloud-based service for creating, managing, and working with software development projects on AWS. You can quickly develop, build, and deploy applications on AWS with an AWS CodeStar project. An AWS CodeStar project creates and integrates AWS services for your project development toolchain, including an AWS CodeCommit repository for the project. AWS CodeStar also assigns permissions to team members for that project. These permissions are applied automatically, including permissions for accessing AWS CodeCommit, creating and managing Git credentials, and more.

You can configure repositories created for AWS CodeStar projects just as you would any other AWS CodeCommit repository by using the AWS CodeCommit console, AWS CodeCommit commands from the AWS CLI, the local Git client, and from the AWS CodeCommit API.

Learn more:
- Working with Repositories (p. 75)
- Working with AWS CodeStar Projects
- Working with AWS CodeStar Teams
### AWS CloudTrail

CloudTrail captures AWS API calls and related events made by or on behalf of an AWS account and delivers log files to an Amazon S3 bucket that you specify. You can configure CloudTrail to capture API calls from the AWS CodeCommit console, AWS CodeCommit commands from the AWS CLI, the local Git client, and from the AWS CodeCommit API.

Learn more:

- Logging AWS CodeCommit API Calls with AWS CloudTrail (p. 289)

### AWS CodeBuild

AWS CodeBuild is a fully managed build service in the cloud that compiles your source code, runs unit tests, and produces artifacts that are ready to deploy. You can store the source code to be built and the build specification in an AWS CodeCommit repository. You can use AWS CodeBuild directly with AWS CodeCommit, or you can incorporate both AWS CodeBuild and AWS CodeCommit in a continuous delivery pipeline with AWS CodePipeline.

Learn more:

- Plan a Build
- Create a Build Project
- Use AWS CodePipeline with AWS CodeBuild to Run Builds

### AWS Elastic Beanstalk

Elastic Beanstalk is a managed service that makes it easy to deploy and manage applications in the AWS cloud without worrying about the infrastructure that runs those applications. You can use the Elastic Beanstalk command line interface (EB CLI) to deploy your application directly from a new or existing AWS CodeCommit repository.

Learn more:

- Using the EB CLI with AWS CodeCommit
- Using an Existing AWS CodeCommit Repository
- eb codesource (EB CLI command)
<table>
<thead>
<tr>
<th>AWS Service</th>
<th>Description</th>
<th>Learn more:</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AWS CloudFormation</strong></td>
<td>AWS CloudFormation is a service that helps you model and set up your AWS resources so that you can spend less time managing those resources and more time focusing on your applications. You create a template that describes resources, including an AWS CodeCommit repository, and AWS CloudFormation takes care of provisioning and configuring those resources for you.</td>
<td>• AWS CodeCommit Repository resource page</td>
</tr>
</tbody>
</table>
| **AWS CodePipeline**              | AWS CodePipeline is a continuous delivery service you can use to model, visualize, and automate the steps required to release your software. You can configure AWS CodePipeline to use an AWS CodeCommit repository as a source action in a pipeline, and automate building, testing, and deploying your changes. | • Simple Pipeline Walkthrough with AWS CodePipeline and AWS CodeCommit
• Migrate to Amazon CloudWatch Events Change Detection for Pipelines with an AWS CodeCommit Repository
• Change-Detection Methods Used to Start Pipelines Automatically |
| **AWS Key Management Service**    | AWS KMS is a managed service that makes it easy for you to create and control the encryption keys used to encrypt your data. By default, AWS CodeCommit uses AWS KMS to encrypt repositories. | • AWS KMS and Encryption (p. 288)                                                                        |
| **AWS Lambda**                    | Lambda lets you run code without provisioning or managing servers. You can configure triggers for AWS CodeCommit repositories that will invoke Lambda functions in response to repository events. | • Create a Trigger for a Lambda Function (p. 97)                                                            |
|                                   |                                                                                                                                                                                                                                  | • AWS Lambda Developer Guide                                                                           |
Amazon Simple Notification Service

Amazon SNS is a web service that enables applications, end users, and devices to instantly send and receive notifications from the cloud. You can configure triggers for AWS CodeCommit repositories that will send Amazon SNS notifications in response to repository events. You can also use Amazon SNS notifications to integrate with other AWS services. For example, you can use an Amazon SNS notification to send messages to an Amazon Simple Queue Service queue.

Learn more:
- Create a Trigger for an Amazon SNS Topic (p. 92)
- Amazon Simple Notification Service Developer Guide

Integration Examples from the Community

The following sections provide links to blog posts, articles, and community-provided examples.

Note
These links are provided for informational purposes only, and should not be considered either a comprehensive list or an endorsement of the content of the examples. AWS is not responsible for the content or accuracy of external content.

Topics
- Blog Posts (p. 72)
- Code Samples (p. 73)

Blog Posts
- **Build Serverless AWS CodeCommit Workflows using Amazon CloudWatch Events and JGit**
  Learn how to create CloudWatch Events rules that process changes in a repository using AWS CodeCommit repository events and target actions in other AWS services. Examples include AWS Lambda functions that enforce Git commit message policies on commits, replicate an AWS CodeCommit repository, and backing up an AWS CodeCommit repository to Amazon S3.
  Published August 3, 2017
- **Replicating and Automating Sync-Ups for a Repository with AWS CodeCommit**
  Learn how to back up or replicate an AWS CodeCommit repository to another AWS region, and how to back up repositories hosted on other services to AWS CodeCommit.
  Published March 17, 2017
- **Migrating to AWS CodeCommit**
  Learn how to push code to two repositories as part of migrating from using another Git repository to AWS CodeCommit when using SourceTree.
  Published September 6, 2016
• **Set Up Continuous Testing with Appium, AWS CodeCommit, Jenkins, and AWS Device Farm**

Learn how to create a continuous testing process for mobile devices using Appium, AWS CodeCommit, Jenkins, and Device Farm.

Published February 2, 2016

• **Using AWS CodeCommit with Git Repositories in Multiple AWS Accounts**

Learn how to clone your AWS CodeCommit repository and, in one command, configure the credential helper to use a specific IAM role for connections to that repository.

Published November 2015

• **Integrating AWS OpsWorks and AWS CodeCommit**

Learn how AWS OpsWorks can automatically fetch Apps and Chef cookbooks from AWS CodeCommit.

Published August 25 2015

• **Using AWS CodeCommit and GitHub Credential Helpers**

Learn how to configure your gitconfig file to work with both AWS CodeCommit and GitHub credential helpers.

Published September 2015

• **Using AWS CodeCommit from Eclipse**

Learn how to use the EGit tools in Eclipse to work with AWS CodeCommit.

Published August 2015

• **AWS CodeCommit with Amazon EC2 Role Credentials**

Learn how to use an instance profile for Amazon EC2 when configuring automated agent access to an AWS CodeCommit repository.

Published July 2015

• **Integrating AWS CodeCommit with Jenkins**

Learn how to use AWS CodeCommit and Jenkins to support two simple continuous integration (CI) scenarios.

Published July 2015

• **Integrating AWS CodeCommit with Review Board**

Learn how to integrate AWS CodeCommit into a development workflow using the Review Board code review system.

Published July 2015

**Code Samples**

The following are code samples that might be of interest to AWS CodeCommit users.

• **Mac OS X Script to Periodically Delete Cached Credentials in the OS X Certificate Store**

If you use the credential helper for AWS CodeCommit on Mac OS X, you are likely familiar with the problem with cached credentials. This script demonstrate one solution.

**Author:** Nico Coetzee
Published February 2016
Working with Repositories in AWS CodeCommit

A repository is the fundamental version control object in AWS CodeCommit. It's where you securely store code and files for your project. It also stores your project history, from the first commit through the latest changes. You can share your repository with other users so you can work together on a project. You can set up notifications so that repository users receive email about events (for example, another user commenting on their code). You can also change the default settings for your repository, browse its contents, and more. You can create triggers for your repository so that code pushes or other events trigger actions, such as emails or code functions. You can even configure a repository on your local computer (a local repo) to push your changes to more than one repository.

Before you can push changes to an AWS CodeCommit repository, you must configure your IAM user in your AWS account. For more information, see Step 1: Initial Configuration for AWS CodeCommit (p. 6).

For information about working with other aspects of your repository in AWS CodeCommit, see Working with Files (p. 130), Working with Pull Requests (p. 138), Working with Commits (p. 163), Working with Branches (p. 194), and Working with User Preferences (p. 208). To learn about migrating to AWS CodeCommit, see Migrate to AWS CodeCommit (p. 211).

Topics

- Create an AWS CodeCommit Repository (p. 76)
- Connect to an AWS CodeCommit Repository (p. 78)
- Share an AWS CodeCommit Repository (p. 80)
- Configuring Notifications for Events in an AWS CodeCommit Repository (p. 84)
- Manage Triggers for an AWS CodeCommit Repository (p. 91)
- View AWS CodeCommit Repository Details (p. 109)
- Change AWS CodeCommit Repository Settings (p. 113)
- Synchronize Changes Between a Local Repo and an AWS CodeCommit Repository (p. 116)
- Push Commits to an Additional Git Repository (p. 117)
Create an AWS CodeCommit Repository

Use AWS CLI or the AWS CodeCommit console to create a new, empty AWS CodeCommit repository.

These instructions assume you have already completed the steps in Setting Up (p. 4).

Note
 Depending on your usage, you might be charged for creating or accessing a repository. For more information, see Pricing on the AWS CodeCommit product information page.

Topics

• Use the AWS CodeCommit Console to Create a Repository (p. 76)
• Use the AWS CLI to Create an AWS CodeCommit Repository (p. 77)

Use the AWS CodeCommit Console to Create a Repository

To create a new AWS CodeCommit repository (console):

2. In the region selector, choose the region where you will create the repository. For more information, see Regions and Git Connection Endpoints (p. 278).
3. On the Dashboard page, choose Create repository. (If a welcome page appears instead of the Dashboard page, choose Get Started Now.)
4. On the Create repository page, in Repository name, type a name for the repository.
   Note
   This name must be unique in the region for your AWS account.
5. (Optional) In the Description box, type a description for the repository. This can help you and other users identify the purpose of the repository.
   Note
   The description field accepts all HTML characters and all valid Unicode characters. If you are an application developer using the GetRepository or BatchGetRepositories APIs and plan to display the repository description field in a web browser, see the AWS CodeCommit API Reference.
6. Choose Create repository.
7. In Configure email notifications, configure notifications so that repository users receive emails about important repository events. This step is optional, but recommended. You can choose the event types (for example, comments on code) and whether to use an existing Amazon SNS topic or create one specifically for this purpose. You can choose to skip this step and configure notifications at a later time. For more information, see Configuring Notifications for Events in an AWS CodeCommit Repository (p. 84).

After you create a repository, you can connect to it and start adding code. To learn more, see Connect to a Repository (p. 78). You can also add your repository to a continuous delivery pipeline. To learn more, see Simple Pipeline Walkthrough.

To get information about the new AWS CodeCommit repository, such as the URLs to use when cloning the repository, choose the repository’s name from the list.
Use the AWS CLI to Create an AWS CodeCommit Repository

To create a new AWS CodeCommit repository (CLI):

1. Make sure that you have configured the AWS CLI with the region where the repository exists. To verify the region, type the following command at the command line or terminal and review the information for default region name:

   ```bash
   aws configure
   ```

   The default region name must match the region for the repository in AWS CodeCommit. For more information, see Regions and Git Connection Endpoints (p. 278).

2. Run the `create-repository` command, specifying:
   - A name that uniquely identifies the AWS CodeCommit repository (with the `--repository-name` option).
     
     **Note**
     
     This name must be unique across an AWS account.
   - Optionally, a comment about the AWS CodeCommit repository (with the `--repository-description` option).

   For example, to create an AWS CodeCommit repository named `MyDemoRepo` with the description "My demonstration repository":

   ```bash
   aws codecommit create-repository --repository-name MyDemoRepo --repository-description "My demonstration repository"
   ```

   **Note**
   The description field accepts all HTML characters and all valid Unicode characters. If you are an application developer using the GetRepository or BatchGetRepositories APIs and plan to display the repository description field in a web browser, see the AWS CodeCommit API Reference.

3. If successful, this command outputs a `repositoryMetadata` object with the following information:
   - The description (`repositoryDescription`).
   - The unique, system-generated ID (`repositoryId`).
   - The name (`repositoryName`).
   - The ID of the AWS account associated with the AWS CodeCommit repository (`accountId`).

   Here is some example output, based on the preceding example command:

   ```json
   {
     "repositoryMetadata": {
       "repositoryName": "MyDemoRepo",
       "repositoryDescription": "My demonstration repository",
       "repositoryId": "f7579e13-b83e-4027-aaef-650c0EXAMPLE",
   ```
Connect to a Repository

4. Note the AWS CodeCommit repository's name and ID. You will need them to monitor and change information about the AWS CodeCommit repository, especially if you use AWS CLI.

   If you forget the AWS CodeCommit repository's name or ID, follow the instructions in Use the AWS CLI to View AWS CodeCommit Repository Details (p. 111).

   After you create a repository, you can connect to it and start adding code. To learn more, see Connect to a Repository (p. 78). You can also add your repository to a continuous delivery pipeline. To learn more, see Simple Pipeline Walkthrough.

Connect to an AWS CodeCommit Repository

When you connect to an AWS CodeCommit repository for the first time, you typically clone its contents to your local machine. Alternatively, if you already have a local repo, you can add an AWS CodeCommit repository as a remote. This topic provides instructions for connecting to an AWS CodeCommit repository. If you want to migrate an existing repository to AWS CodeCommit, see Migrate to AWS CodeCommit (p. 211).

   Note
   Depending on your usage, you might be charged for creating or accessing a repository. For more information, see Pricing on the AWS CodeCommit product information page.

Topics
- Prerequisites for Connecting to an AWS CodeCommit Repository (p. 78)
- Connect to the AWS CodeCommit Repository by Cloning the Repository (p. 79)
- Connect a Local Repo to the AWS CodeCommit Repository (p. 80)

Prerequisites for Connecting to an AWS CodeCommit Repository

Before you can connect to an AWS CodeCommit repository:

- You must have configured your local computer with the software and settings required to connect to AWS CodeCommit. For more information, see Setting Up (p. 4).
- You must have the clone URL of the AWS CodeCommit repository to which you want to connect. This URL includes the name of the repository as well as its AWS region. For more information, see View Repository Details (p. 109).

   If you have not yet created an AWS CodeCommit repository, follow the instructions in Create a Repository (p. 76), copy the clone URL of the new AWS CodeCommit repository, and return to this page.

   If you have an AWS CodeCommit repository but you do not know its name, follow the instructions in View Repository Details (p. 109).

- You must have a location on your local machine to store a local copy of the AWS CodeCommit repository to which you will be connecting. (This local copy of the AWS CodeCommit repository is known as a local repo.) You then switch to and run Git commands from that location. For example, you could use /tmp (for Linux, macOS, or Unix) or c:\temp (for Windows).
Connect to the AWS CodeCommit Repository by Cloning the Repository

If you do not already have a local repo, follow the steps in this procedure to clone the AWS CodeCommit repository to your local machine.

1. Complete the prerequisites, including Setting Up (p. 4).

   Important
   If you have not completed setup, you will not be able to connect to or clone the repository.

2. From the /tmp directory or the c:\temp directory, use Git to run the clone command, as shown in the following example for cloning a repository named MyDemoRepo in the US East (Ohio) region:

   **For HTTPS:**
   ```
   ```

   **For SSH:**
   ```
   git clone ssh://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo my-demo-repo
   ```

   In this example, git-codecommit.us-east-2.amazonaws.com is the Git connection point for the US East (Ohio) region where the repository exists, MyDemoRepo represents the name of your AWS CodeCommit repository, and my-demo-repo represents the name of the directory Git will create in the /tmp directory or the c:\temp directory. For more information about the regions that support AWS CodeCommit and the Git connections for those regions, see Regions and Git Connection Endpoints (p. 278).

   **Note**
   When you use SSH on Windows operating systems to clone a repository, you might need to add the SSH key ID to the connection string as follows:

   ```
   git clone ssh://Your-SSH-Key-ID@git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo my-demo-repo
   ```

   For more information, see For SSH Connections on Windows (p. 30) and Troubleshooting (p. 235).

After Git creates the directory, it will pull down a copy of your AWS CodeCommit repository into the newly created directory.

If the AWS CodeCommit repository is new or otherwise empty, you will see a message that you are cloning an empty repository. This is expected.

**Note**
If you receive an error that Git can't find the AWS CodeCommit repository or that you don't have permission to connect to the AWS CodeCommit repository, make sure you completed the prerequisites (p. 4), including assigning permissions to the IAM user and setting up your IAM user credentials for Git and AWS CodeCommit on the local machine. Also, make sure you specified the correct repository name.
Connect a Local Repo to the AWS CodeCommit Repository

After you successfully connect your local repo to your AWS CodeCommit repository, you are now ready to start running Git commands from the local repo to create commits, branches, and tags and push to and pull from the AWS CodeCommit repository.

1. Complete the prerequisites (p. 78).
2. From the command prompt or terminal, switch to your local repo directory and run the `git remote add` command to add the AWS CodeCommit repository as a remote repository for your local repo.

For example, the following command adds the remote nicknamed `origin` to https://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo:

For HTTPS:

```
git remote add origin https://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo
```

For SSH:

```
git remote add origin ssh://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo
```

This command returns nothing.
3. To verify you have added the AWS CodeCommit repository as a remote for your local repo, run the `git remote -v` command, which should create output similar to the following:

For HTTPS:

```
origin  https://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo (fetch)
origin  https://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo (push)
```

For SSH:

```
origin  ssh://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo (fetch)
origin  ssh://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo (push)
```

After you successfully connect your local repo to your AWS CodeCommit repository, you are ready to start running Git commands from the local repo to create commits, branches, and tags, and to push to and pull from the AWS CodeCommit repository.

Share an AWS CodeCommit Repository

After you have created an AWS CodeCommit repository, you can share it with other users. First, decide which protocol to recommend to users when connecting to your repository: HTTPS or SSH. Then
send the URL and connection information to the users with whom you want to share the repository. Depending on your security requirements, sharing a repository may also require creating an IAM group, applying managed policies to that group, and editing IAM policies to refine access. This topic will walk you through these steps.

These instructions assume you have already completed the steps in Setting Up (p. 4) and Create a Repository (p. 76).

**Note**
Depending on your usage, you might be charged for creating or accessing a repository. For more information, see Pricing on the AWS CodeCommit product information page.

**Topics**
- Choose the Connection Protocol to Share with Your Users (p. 81)
- Create IAM Policies for Your Repository (p. 81)
- Create an IAM Group for Repository Users (p. 82)
- Share the Connection Information with Your Users (p. 83)

## Choose the Connection Protocol to Share with Your Users

When you create a repository in AWS CodeCommit, two endpoints are generated: one for HTTPS connections and one for SSH connections. Both provide secure connections over a network. Your users can use either protocol. Both endpoints remain active regardless of which protocol you recommend to your users.

HTTPS connections require either Git credentials, which IAM users can generate for themselves in IAM, or an AWS access key, which your repository users must configure in the credential helper included in the AWS CLI but is the only method available for root account or federated users. Git credentials are the easiest method for users of your repository to set up and use. SSH connections require your users to generate a public-private key pair, store the public key, associate the public key with their IAM user, configure their known hosts file on their local computer, and create and maintain a config file on their local computers. Because this is a more complex configuration process, we recommend you choose HTTPS and Git credentials for connections to AWS CodeCommit.

For more information about HTTPS, SSH, Git, and remote repositories, see Setting Up (p. 4) or consult your Git documentation. For a general overview of communication protocols and how each communicates with remote repositories, see Git on the Server - The Protocols.

**Note**
Although Git supports a variety of connection protocols, AWS CodeCommit does not support connections with unsecured protocols, such as the local protocol or generic HTTP.

## Create IAM Policies for Your Repository

AWS provides three managed policies in IAM for AWS CodeCommit. These policies cannot be edited and apply to all repositories associated with your AWS account. However, you can use these policies as templates to create your own custom managed policies that apply only to the repository you want to share. Your customer managed policy can apply specifically to the repository you want to share.

For more information about managed policies and IAM users, see Managed Policies and IAM Users and Groups.

**Tip**
For more fine-grained control over access to your repository, you can create more than one customer managed policy and apply the policies to different IAM users and groups.
Create a customer managed policy for your repository

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the Dashboard navigation area, choose Policies, and then choose Create Policy.
3. On the Create Policy page, next to Copy an AWS Managed Policy, choose Select.
5. On the Review Policy page, in Policy Name, type a new name for the policy (for example, AWSCodeCommitPowerUser-MyDemoRepo).

In the Policy Document text box, replace the "*" portion of the Resource line with the Amazon Resource Name (ARN) of the AWS CodeCommit repository. For example:

"Resource": [
  "arn:aws:codecommit:us-east-2:80398EXAMPLE:MyDemoRepo"
]

Tip
To find the ARN for the AWS CodeCommit repository, go to the AWS CodeCommit console and choose the repository name from the list. For more information, see View Repository Details (p. 109).

If you want this policy to apply to more than one repository, add each repository as a resource by specifying its ARN. Include a comma between each resource statement, as shown in the following example:

"Resource": [
  "arn:aws:codecommit:us-east-2:80398EXAMPLE:MyOtherDemoRepo"
]

6. Choose Validate Policy. After it is validated, choose Create Policy.

   Tip
Creating a managed policy for a repository does not supply additional permissions required for individual users to set up Git credentials or SSH keys in IAM. You must apply these managed policies to individual IAM users.

   • To allow users to use Git credentials to connect to AWS CodeCommit, select the IAMSelfManageServiceSpecificCredentials and IAMReadOnlyAccess managed policies and apply them to your users.
   • To allow users to use SSH to connect to AWS CodeCommit, select the IAMUserSSHKeys and IAMReadOnlyAccess managed policies and apply them to your users.

Create an IAM Group for Repository Users

To manage access to your repository, create an IAM group for its users, add IAM users to that group, and then attach the customer managed policy you created in the previous step.
If you use SSH, you must attach another managed policy to the IAMUserSSHKeys group, the IAM managed policy that allows users to upload their SSH public key and associate it with the IAM user they use to connect to AWS CodeCommit.

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the Dashboard navigation area, choose Groups, and then choose Create New Group.
3. On the Set Group Name page, in the Group Name box, type a name for the group (for example, MyDemoRepoGroup), and then choose Next Step. Consider including the repository name as part of the group name.

   **Note**
   This name must be unique across an AWS account.

4. Select the check box next to the customer managed policy you created in the previous section (for example, AWSCodeCommitPowerUser-MyDemoRepo).
   - If your users will use HTTPS and Git credentials to connect to AWS CodeCommit, select the check boxes next to IAMSelfManageServiceSpecificCredentials and IAMReadOnlyAccess, and then choose Next Step.
   - If your users will use SSH to connect to your repository, select the check boxes next to IAMUserSSHKeys and IAMReadOnlyAccess, and then choose Next Step.

5. On the Review page, choose Create Group. The group will be created in IAM with the specified policies already attached. It will appear in the list of groups associated with your AWS account.
6. Choose your group from the list.
7. On the group summary page, choose the Users tab, and then choose Add Users to Group. On the list that shows all users associated with your AWS account, select the check boxes next to the users to whom you want to allow access to the AWS CodeCommit repository, and then choose Add Users.

   **Tip**
   You can use the Search box to quickly find users by name.

8. When you have added your users, close the IAM console.

**Share the Connection Information with Your Users**

2. In the region selector, choose the region where the repository was created. Repositories are specific to an AWS region. For more information, see Regions and Git Connection Endpoints (p. 278).
3. On the Dashboard page, choose the name of the repository you want to share.
4. On the Code page, choose Clone URL, and then choose the protocol you want your users to use.
5. Copy the displayed URL for the connection protocol your users will use when connecting to your AWS CodeCommit repository.
6. Send your users the connection information along with any other instructions, such as installing the AWS CLI, configuring a profile, or installing Git. Make sure to include the configuration information for the connection protocol (for example, for HTTPS, configuring the credential helper for Git).

The following example email provides information for users connecting to the MyDemoRepo repository with the HTTPS connection protocol and Git credentials in the US East (Ohio) (us-east-2) region. This email assumes the user has already installed Git and is familiar with using it:

I've created an AWS CodeCommit repository for us to use while working on our project. The name of the repository is MyDemoRepo, and it is in the US East (Ohio) (us-east-2) region. Here's what you need to do in order to get started using it:
1. Make sure that your version of Git on your local computer is 1.7.9 or later.
2. Generate Git credentials for your IAM user by signing into the IAM console here: [https://console.aws.amazon.com/iam/](https://console.aws.amazon.com/iam/).
   Switch to the Security credentials tab for your IAM user and choose the Generate button in HTTPS Git credentials for AWS CodeCommit.
   Make sure to save your credentials in a secure location!
3. Switch to a directory of your choice and clone the AWS CodeCommit repository to your local machine by running the following command:
   ```bash
   ```
4. When prompted for username and password, use the Git credentials you just saved.

That's it! If you'd like to learn more about using AWS CodeCommit, you can start with the tutorial [here](https://docs.aws.amazon.com/codecommit/latest/userguide/using-aws-codecommit.html#using-aws-codecommit-tutorial).

You can find complete setup instructions in Setting Up (p. 4).

Configuring Notifications for Events in an AWS CodeCommit Repository

You can set up notifications for a repository so that repository users receive emails about the repository event types you specify. When you configure notifications, AWS CodeCommit creates an Amazon CloudWatch Events rule for your repository. This rule responds to the event types you select from the preconfigured options in the AWS CodeCommit console. Notifications are sent when events match the rule settings. You can create an Amazon SNS topic to use for notifications, or use an existing one in your AWS account.

You use the AWS CodeCommit console to configure notifications.
Using Repository Notifications

Configuring notifications helps your repository users by sending emails to users when someone takes an action that affects another user. For example, you can configure a repository to send notifications when comments are made on commits. In this configuration, when a repository user comments on a line of code in a commit, other repository users receive an email. They can sign in and view the comment. Responses to comments also generate emails, so repository users stay informed.

Notification event types are grouped into the following categories:

- **Pull request update events**: If you select this option, users receive emails when:
  - A pull request is created or closed.
  - A pull request is updated with code changes.
  - The title or description of the pull request changes.
Configure Repository Notifications

- **Pull request comment events**: If you select this option, users receive emails when someone comments or replies to a comment in a pull request.
- **Commit comment events**: If you select this option, users receive emails when someone comments on a commit outside of a pull request. This includes comments on:
  - Lines of code in a commit.
  - Files in a commit.
  - The commit itself.

For more information, see Comment on a Commit (p. 179).

Repository notifications are different from repository triggers. Although you can configure a trigger to use Amazon SNS to send emails about some repository events, those events are limited to operational events such as creating branches and pushing code to a branch. Triggers do not use CloudWatch Events rules to evaluate repository events. They are more limited in scope. For more information about using triggers, see Manage Triggers for a Repository (p. 91).

Configure Repository Notifications

You can keep repository users informed of repository events by configuring notifications. When you configure notifications, subscribed users receive emails about the events that you specify, such as when someone comments on a commit. For more information, see Using Repository Notifications (p. 85).

To use the AWS CodeCommit console to configure notifications for a repository in AWS CodeCommit, you must have the following managed policy or the equivalent permissions attached to your IAM user:

- **CloudWatchEventsFullAccess**
- **AmazonSNSFullAccess**

**Note**
Equivalent permissions are included in the **AWSCodeCommitFullAccess** policy, which is required to configure repository notifications. If you have this policy applied, you do not need the other two policies. If you have a customized policy applied, you might need to modify it to include the required permissions for CloudWatch Events and Amazon SNS.

To configure notifications for a repository

2. In the list of repositories, choose the name of the repository where you want to configure notifications.
3. In the navigation pane, choose **Settings**, and then choose **Notifications**.
4. In **Event types**, select the event types you want included in the CloudWatch Events rule for the repository.
5. In SNS topic, either choose a topic from the list of Amazon SNS topics in your AWS account, or create a topic to use for this repository.

   **Note**
   If you create a topic, you can manage subscriptions for that policy from the AWS CodeCommit console. If you use an existing topic, you cannot manage subscriptions for that topic unless you have permissions to manage subscriptions for all topics in Amazon SNS. For more information, see Amazon Simple Notification Service Developer Guide.

   If you create a topic, in **Topic name**, type a name for the topic. Optionally, in **Display name**, type a short name. Choose **Create**.

6. To add the email addresses of the repository users, choose Manage subscriptions. In **Add email subscriber**, type the email address of a repository user, and then choose **Save**. You can add only one email address at a time.

   **Note**
   A confirmation email is sent to the address as soon as you choose **Save**. However, the status of the subscription is not updated while you remain in Manage subscriptions.

   After you have added all the email addresses to the list of subscribers, choose **Close**.

   **Tip**
   Amazon SNS coordinates and manages the delivery and sending of messages to subscribing endpoints and email addresses. Endpoints include web servers, email addresses, Amazon Simple Queue Service queues, and AWS Lambda functions. For more information, see What Is Amazon Simple Notification Service? and Sending Amazon SNS Messages to HTTP/HTTPS Endpoints.

7. To finish configuring notifications, choose **Save**.
After you have configured notifications for a repository, you can view the CloudWatch Events rule automatically created for the repository.

**Important**
Do not edit or delete this rule. Changing or deleting the rule might cause operational issues. For example, emails might not be sent to subscribers or the inability to change notification settings for a repository in AWS CodeCommit.

**To view the CloudWatch Events rule for a repository**

1. Sign in to the AWS Management Console and open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation bar, under **Events**, choose **Rules**.
3. Choose the rule for your repository from the list. The rule name is displayed on the **Notifications** tab in your repository settings.
4. View the rule summary information.

**Important**
Do not edit, delete, or disable this rule.

**Change, Disable, or Enable Notifications**

You can use the AWS CodeCommit console to change how notifications are configured, including the event types that send emails to users and the Amazon SNS topic used to send emails about the repository, or to manage the list of email addresses and endpoints subscribed to the topic. You can also use the console to temporarily disable notifications.

**To change notification settings (console)**

2. In the list of repositories, choose the name of the repository where you want to configure notifications.
3. In the navigation pane, choose **Settings**, and then choose **Notifications**.
4. Choose **Edit**.
5. Make your changes, and then choose **Save**.

Disabling notifications is an easy way to temporarily prevent users from receiving emails about repository events. For example, you might want to disable notifications while you are performing repository maintenance. Because the configuration is preserved, you can quickly enable notifications when you are ready.

To permanently delete the notification settings, follow the steps in **Delete Notification Settings for a Repository** (p. 89).

**To disable notifications (console)**

2. In the list of repositories, choose the name of the repository where you want to disable notifications.
3. In the navigation pane, choose **Settings**, and then choose **Notifications**.
4. Choose **Disable**.
5. The notification state changes to **Disabled**. No emails about events are sent. When you disable notifications, the CloudWatch Events rule for the repository is disabled automatically. Do not manually change its status in the CloudWatch Events console.
To enable notifications (console)
2. In the list of repositories, choose the name of the repository where notifications are disabled.
3. In the navigation pane, choose Settings, and then choose Notifications.
4. Choose Enable.
5. The notification state changes to Enabled. Emails about events are sent. The CloudWatch Events rule for the repository is enabled automatically.

Delete Notification Settings for a Repository

If you no longer want notifications for your repository, you can delete the settings. Deleting the settings also deletes the CloudWatch Events rule created for notifications for the repository. It does not delete any subscriptions or the Amazon SNS topic used for notifications.

Note
If you change the name of a repository from the console, notifications continue to work without modification. However, if you change the name of your repository from the command line or by using the API, notifications no longer work. The easiest way to restore notifications is to delete the notification settings and then configure them again.

To delete notification settings (console)
2. In the list of repositories, choose the name of the repository where notifications are disabled.
3. In the navigation pane, choose Settings, and then choose Notifications.
4. Choose **Delete**.

5. In **Delete notification settings**, choose **Delete notification settings**.

If you delete these settings, users of this repository will no longer receive emails about events in this repository. However, the SNS topic will not be deleted. Subscribers to that topic might still receive emails if other repositories or other AWS services are configured to use that topic. To delete the topic, go to the **SNS Console**.

6. (Optional) To change or delete the Amazon SNS topic used for notifications after you delete notification settings, go to the Amazon SNS console at [https://console.aws.amazon.com/sns/v2/home](https://console.aws.amazon.com/sns/v2/home). For more information, see **Clean Up** in **Amazon Simple Notification Service Developer Guide**.
Manage Triggers for an AWS CodeCommit Repository

You can configure an AWS CodeCommit repository so that code pushes or other events trigger actions, such as sending a notification from Amazon Simple Notification Service (Amazon SNS) or invoking a function in AWS Lambda. You can create up to ten triggers for each AWS CodeCommit repository.

Triggers are commonly configured to:

• Send emails to subscribed users every time someone pushes to the repository.
• Notify an external build system to start a build after someone pushes to the main branch of the repository.

Scenarios like notifying an external build system require writing a Lambda function to interact with other applications. The email scenario simply requires creating an Amazon SNS topic.

In this topic, you will learn how to set permissions that allow AWS CodeCommit to trigger actions in Amazon SNS and Lambda. You will also find links to examples for creating, editing, testing, and deleting triggers.

Topics

• Create the Resource and Add Permissions for AWS CodeCommit (p. 91)
• Example: Create an AWS CodeCommit Trigger for an Amazon SNS Topic (p. 92)
• Example: Create an AWS CodeCommit Trigger for an AWS Lambda Function (p. 97)
• Example: Create a Trigger in AWS CodeCommit for an Existing AWS Lambda Function (p. 100)
• Edit Triggers for an AWS CodeCommit Repository (p. 105)
• Test Triggers for an AWS CodeCommit Repository (p. 106)
• Delete Triggers from an AWS CodeCommit Repository (p. 108)

Create the Resource and Add Permissions for AWS CodeCommit

You can integrate Amazon SNS topics and Lambda functions with triggers in AWS CodeCommit, but you must first create and then configure resources with a policy that allows AWS CodeCommit the permissions to interact with those resources. You must create the resource in the same region as the AWS CodeCommit repository. For example, if the repository is in US East (Ohio) (us-east-2), the Amazon SNS topic or Lambda function must be in US East (Ohio).

• For Amazon SNS topics, you do not need to configure additional IAM policies or permissions if the Amazon SNS topic is created using the same account as the AWS CodeCommit repository. You can create the AWS CodeCommit trigger as soon as you have created and subscribed to the Amazon SNS topic.
• For more information about creating topics in Amazon SNS, see the Amazon SNS documentation.
• For information about using Amazon SNS to send messages to Amazon SQS queues, see Sending Messages to Amazon SQS Queues.
• For information about using Amazon SNS to invoke a Lambda function, see Invoking Lambda Functions.
• If you want to configure your trigger to use an Amazon SNS topic in another AWS account, you must first configure that topic with a policy that allows AWS CodeCommit to publish to that topic. For more
You can configure Lambda functions by creating the trigger in the Lambda console as part of the function. This is the simplest method, as triggers created in the Lambda console automatically include the permissions required for AWS CodeCommit to invoke the Lambda function. If you create the trigger in AWS CodeCommit, you must include a policy to allow AWS CodeCommit to invoke the function. For more information, see Create a Trigger for an Existing Lambda Function (p. 100) and Example 2: Create a Policy for AWS Lambda Integration (p. 267).

Example: Create an AWS CodeCommit Trigger for an Amazon SNS Topic

You can create a trigger for an AWS CodeCommit repository so that events in that repository trigger notifications from an Amazon Simple Notification Service (Amazon SNS) topic. You might want to create a trigger to an Amazon SNS topic to enable users to subscribe to notifications about repository events, such as the deletion of branches. You can also take advantage of the integration of Amazon SNS topics with other services, such as Amazon Simple Queue Service (Amazon SQS) and AWS Lambda.

Note
You must point the trigger to an existing Amazon SNS topic that will be the action taken in response to repository events. For more information about creating and subscribing to Amazon SNS topics, see Getting Started with Amazon Simple Notification Service.

Topics
• Create a Trigger to an Amazon SNS Topic for an AWS CodeCommit Repository (Console) (p. 92)
• Create a Trigger to an Amazon SNS Topic for an AWS CodeCommit Repository (AWS CLI) (p. 93)

Create a Trigger to an Amazon SNS Topic for an AWS CodeCommit Repository (Console)

2. From the list of repositories, choose the repository where you want to create triggers for repository events.
3. In the navigation pane for the repository, choose Settings. In Settings, choose Triggers.
4. Choose Create trigger.
5. In the Create trigger pane, do the following:
   • In Trigger name, type a name for the trigger, such as MyFirstTrigger.
   • In Events, select the repository events that will trigger the Amazon SNS topic to send notifications.
     
     If you choose All repository events, you cannot choose any other events. To choose a subset of events, remove All repository events, and then choose one or more events from the list. For example, if you want the trigger to run only when a user creates a branch or tag in the AWS CodeCommit repository, remove All repository events, and then choose Create branch or tag.
     
     If you want the trigger to apply to all branches of the repository, in Branches, choose All branches. Otherwise, choose Specific branches. The default branch for the repository will be added by default. You can keep or delete this branch from the list. Choose up to ten branch names from the list of repository branches.
   • In Send to, choose Amazon SNS.
• In **Amazon SNS Topic**, choose a topic name from the list, or choose **Add an Amazon SNS topic ARN** and then type the ARN for the function.

• In **Custom data**, optionally provide any information you want included in the notification sent by the Amazon SNS topic (for example, an IRC channel name developers use when discussing development in this repository). This field is a string. It cannot be used to pass any dynamic parameters.

6. Optionally, choose **Test trigger**. This step will help you confirm have correctly configured access between AWS CodeCommit and the Amazon SNS topic. It will use the Amazon SNS topic to send a test notification using data from your repository, if available. If no real data is available, the test notification will contain sample data.

7. Choose **Create** to finish creating the trigger.

Create a Trigger to an Amazon SNS Topic for an AWS CodeCommit Repository (AWS CLI)

You can also use the command line to create a trigger for an Amazon SNS topic in response to AWS CodeCommit repository events, such as when someone pushes a commit to your repository.

**To create a trigger for an Amazon SNS topic**

1. Open a plain-text editor and create a JSON file that specifies:
   
   • The Amazon SNS topic name.
• The repository and branches you want to monitor with this trigger. (If you do not specify any branches, the trigger will apply to all branches in the repository.)
• The events that will activate this trigger.

Save the file.

For example, to create a trigger for a repository named MyDemoRepo that will publish all repository events to an Amazon SNS topic named MySNSTopic for two branches, master and preprod:

```
{"repositoryName": "MyDemoRepo",
 "triggers": [ {
 "name": "MyFirstTrigger",
 "customData": "",
 "branches": [ "master", "preprod" ],
 "events": [ "all" ]
 } ]
}
```

There must be a trigger block in the JSON for each trigger for a repository. To create more than one trigger for the repository, include more than one trigger block in the JSON. Remember that all triggers created in this file are for the specified repository. You cannot create triggers for multiple repositories in a single JSON file. For example, if you wanted to create two triggers for a repository, you could create a JSON file with two trigger blocks. In the following example, no branches are specified for the second trigger, so that trigger will apply to all branches:

```
{"repositoryName": "MyDemoRepo",
 "triggers": [ {
 "name": "MyFirstTrigger",
 "customData": "",
 "branches": [ "master", "preprod" ],
 "events": [ "all" ]
 },
 { "name": "MySecondTrigger",
 "destinationArn": "arn:aws:sns:us-east-2:80398EXAMPLE:MySNSTopic2",
 "customData": "",
 "branches": [ ],
 "events": [ "updateReference", "deleteReference" ]
 }
 ]
}
```
You can create triggers for events you specify, such as when a commit is pushed to a repository. Event types include:

- **all** for all events in the specified repository and branches.
- **updateReference** for when commits are pushed to the specified repository and branches.
- **createReference** for when a new branch or tag is created in the specified repository.
- **deleteReference** for when a branch or tag is deleted in the specified repository.

**Note**
You can use more than one event type in a trigger. However, if you specify `all`, you cannot specify other events.

To see the full list of valid event types, at the terminal or command prompt, type `aws codecommit put-repository-triggers help`.

In addition, you can include a string in `customData` (for example, an IRC channel name developers use when discussing development in this repository). This field is a string. It cannot be used to pass any dynamic parameters. This string will be appended as an attribute to the AWS CodeCommit JSON returned in response to the trigger.

2. At a terminal or command prompt, optionally run the `test-repository-triggers` command. This test uses sample data from the repository (or generates sample data if no data is available) to send a notification to the subscribers of the Amazon SNS topic. For example, the following is used to test that the JSON in the trigger file named `trigger.json` is valid and that AWS CodeCommit can publish to the Amazon SNS topic:

```
aws codecommit test-repository-triggers --cli-input-json file://trigger.json
```

If successful, this command returns information similar to the following:

```
{
   "successfulExecutions": [ "MyFirstTrigger" ],
   "failedExecutions": []
}
```

3. At a terminal or command prompt, run the `put-repository-triggers` command to create the trigger in AWS CodeCommit. For example, to use a JSON file named `trigger.json` to create the trigger:

```
aws codecommit put-repository-triggers --cli-input-json file://trigger.json
```

This command returns a `configuration ID`, similar to the following:

```
{
   "configurationId": "0123456-I-AM-AN-EXAMPLE"
}
```

4. To view the configuration of the trigger, run the `get-repository-triggers` command, specifying the name of the repository:

```
aws codecommit get-repository-triggers --repository-name MyDemoRepo
```

This command returns the structure of all triggers configured for the repository, similar to the following:
Create a Trigger for an Amazon SNS Topic

5. To test the functionality of the trigger itself, make and push a commit to the repository where you configured the trigger. You should see a response from the Amazon SNS topic. For example, if you configured the Amazon SNS topic to send an email, you should see an email from Amazon SNS in the email account subscribed to the topic.

The following is example output from an email sent from Amazon SNS in response to a push to an AWS CodeCommit repository:

```json
{
  "Records": [  
    {
      "awsRegion": "us-east-2",
      "codecommit": {
        "references": [  
          {
            "commit": "317f8570EXAMPLE",
            "created": true,
            "ref": "refs/heads/NewBranch"
          },
          {
            "commit": "4c925148EXAMPLE",
            "ref": "refs/heads/preprod"
          }
        ]
      },
      "eventId": "11111-EXAMPLE-ID",
      "eventName": "ReferenceChange",
      "eventPartNumber": 1,
      "eventSource": "aws:codecommit",
      "eventTime": "2016-02-09T00:08:11.743+0000",
      "eventTotalParts": 1,
      "eventTriggerConfigId": "0123456-I-AM-AN-EXAMPLE",
      "eventTriggerName": "MyFirstTrigger",
      "eventVersion": "1.0",
      "customData": "Project ID 12345",
      "userIdentityARN": "arn:aws:iam::80398EXAMPLE:user/JaneDoe-CodeCommit"
    }
  ]
}
```
Example: Create an AWS CodeCommit Trigger for an AWS Lambda Function

You can create a trigger for an AWS CodeCommit repository so that events in the repository will invoke a Lambda function. In this example, you will create a Lambda function that returns the URL used to clone the repository to an Amazon CloudWatch log.

Topics
- Create the Lambda Function (p. 97)
- View the Trigger for the Lambda Function in the AWS CodeCommit Repository (Console) (p. 99)

Create the Lambda Function

You can create an AWS CodeCommit trigger for a Lambda function as part of creating the function itself in the Lambda console. The following steps include a sample Lambda function. The sample is available in two languages: JavaScript and Python. The function returns the URLs used for cloning a repository to a CloudWatch log.

To create a Lambda function using a Lambda blueprint

1. Sign in to the AWS Management Console and open the AWS Lambda console at https://console.aws.amazon.com/lambda/.
2. On the Lambda Functions page, choose Create a Lambda function. (If you have not used Lambda before, choose Get Started Now.)
3. On the Select blueprint page, choose Blank function.
4. On the Configure triggers page, choose AWS CodeCommit from the drop-down list of services to integrate with Lambda.

   ![Configure triggers](image)

   In Repository name, choose the name of the repository where you want to configure a trigger that will use the Lambda function in response to repository events.
   - In Trigger name, type a name for the trigger (for example, MyLambdaFunctionTrigger).
   - In Events, choose the repository events that will trigger the Lambda function. If you choose All repository events, you cannot choose any other events. If you want to choose a subset of events, clear All repository events, and then choose the events you want from the list. For example, if
you want the trigger to run only when a user creates a tag or a branch in the AWS CodeCommit repository, remove All repository events, and then choose Create branch or tag.

- If you want the trigger to apply to all branches of the repository, in Branches, choose All branches. Otherwise, choose Specific branches. The default branch for the repository will be added by default. You can keep or delete this branch from the list. Choose up to ten branch names from the list of repository branches.

- In Custom data, optionally provide information you want included in the Lambda function (for example, the name of the IRC channel used by developers to discuss development in the repository). This field is a string. It cannot be used to pass any dynamic parameters.

Choose Next.

5. On the Configure function page, in Name, type a name for the function (for example, MyCodeCommitFunction). Optionally, in Description, type a description for the function. If you want to create a sample JavaScript function, in Runtime, choose Node.js. If you want to create a sample Python function, choose Python 2.7.

6. In Code entry type, choose Edit code inline, and then replace the hello world code with one of the two following samples.

For Node.js:

```javascript
var aws = require('aws-sdk');
exports.handler = function(event, context) {
    //Log the updated references from the event
    var references = event.Records[0].codecommit.references.map(function(reference) {
        return reference.ref;
    });
    console.log('References:', references);

    //Get the repository from the event and show its git clone URL
    var repository = event.Records[0].eventSourceARN.split(':')[5];
    var params = {
        repositoryName: repository
    };
    codecommit.getRepository(params, function(err, data) {
        if (err) {
            console.log(err);
            var message = "Error getting repository metadata for repository " + repository;
            console.log(message);
            context.fail(message);
        } else {
            console.log('Clone URL:', data.repositoryMetadata.cloneUrlHttp);
            context.succeed(data.repositoryMetadata.cloneUrlHttp);
        }
    });
};
```

For Python:

```python
import json
import boto3

codecommit = boto3.client('codecommit')

def lambda_handler(event, context):
    #Log the updated references from the event
```
references = { reference['ref'] for reference in event['Records'][0]['codecommit']['references'] }
print("References: " + str(references))

# Get the repository from the event and show its git clone URL
repository = event['Records'][0]['eventSourceARN'].split(':')[-5]
try:
    response = codecommit.get_repository(repositoryName=repository)
    print("Clone URL: " + response['repositoryMetadata']['cloneUrlHttp'])
    return response['repositoryMetadata']['cloneUrlHttp']
except Exception as e:
    print(e)
    print('Error getting repository {}. Make sure it exists and that your repository is in the same region as this function.'.format(repository))
    raise e

7. **In Lambda function handler and role**, do the following:
   
   - In **Handler**, leave the default value as derived from the function (index.handler for the Node.js sample or lambda_function.lambda_handler for the Python sample).
   - In **Role**, choose **Create a custom role** from the list. In the IAM console, do the following:
     - In **IAM Role**, choose lambda_basic_execution.
     - In **Policy Name**, choose **Create a new role policy**.
     - Choose Allow to create the role and return to the Lambda console. A value of lambda_basic_execution should now be displayed for Role.

   **Note**
   
   If you choose a different role or a different name for the role, be sure to use it in the steps in this topic.

   Choose Next.

8. On the **Review** page, review the settings for the function, and then choose **Create function**.

### View the Trigger for the Lambda Function in the AWS CodeCommit Repository (Console)

After you have created the Lambda function, you can view and test the trigger in AWS CodeCommit. Testing the trigger will run the function in response to the repository events you specify.

**To view and test the trigger for the Lambda function**

2. From the list of repositories, choose the repository where you want to view triggers.
3. In the navigation pane for the repository, choose **Settings**. In **Settings**, choose **Triggers**.
4. Review the list of triggers for the repository. You should see the trigger you created in the Lambda console. Choose it from the list.
5. In the **Edit trigger** pane, choose **Test trigger**. This option will attempt to invoke the function with sample data about your repository, including the most recent commit ID for the repository. (If no commit history exists, sample values consisting of zeroes will be generated instead.) This will help you confirm you have correctly configured access between AWS CodeCommit and the Lambda function.
6. Choose **Cancel** after you see a success message from the test.
7. To further verify the functionality of the trigger, make and push a commit to the repository where you configured the trigger. You should see a response from the Lambda function on the **Monitoring**
tab for that function in the Lambda console. From the Monitoring tab, choose View logs in CloudWatch. The CloudWatch console will open in a new tab and display events for your function. Select the log stream from the list that corresponds to the time you pushed your commit. You should see event data similar to the following:

```
START RequestId: 70afdc9a-EXAMPLE Version: $LATEST
2015-11-10T18:18:28.689Z 70afdc9a-EXAMPLE References: ['refs/heads/master']
2015-11-10T18:29.142Z 70afdc9a-EXAMPLE Clone URL: https://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo
END RequestId: 70afdc9a-EXAMPLE
REPORT RequestId: 70afdc9a-EXAMPLE Duration: 1126.87 ms Billed Duration: 1200 ms Memory Size: 128 MB Max Memory Used: 14 MB
```

Example: Create a Trigger in AWS CodeCommit for an Existing AWS Lambda Function

The easiest way to create a trigger that will invoke a Lambda function is to create that trigger in the Lambda console. This built-in integration ensures that AWS CodeCommit will have the permissions required to run the function. You can add a trigger for an existing Lambda function by going to the Lambda console, choosing the function, and on the Triggers tab for the function, and then following the steps in Add trigger. These are similar steps to the ones shown in Create the Lambda Function (p. 97).

However, you can also create a trigger for a Lambda function in an AWS CodeCommit repository. Doing so requires that you choose an existing Lambda function to invoke, and also requires that you manually configure the permissions required for AWS CodeCommit to run the function.

**Topics**
- Manually Configure Permissions to Allow AWS CodeCommit to Run a Lambda Function (p. 100)
- Create a Trigger for the Lambda Function in an AWS CodeCommit Repository (Console) (p. 102)
- Create a Trigger to a Lambda Function for an AWS CodeCommit Repository (AWS CLI) (p. 102)

Manually Configure Permissions to Allow AWS CodeCommit to Run a Lambda Function

If you create a trigger in AWS CodeCommit that invokes a Lambda function, you must manually configure the permissions to allow AWS CodeCommit to run the Lambda function. To avoid this manual configuration, consider creating the trigger in the Lambda console for the function instead.

**To allow AWS CodeCommit to run a Lambda function**

1. Open a plain-text editor and create a JSON file that specifies the Lambda function name, the details of the AWS CodeCommit repository, and the actions you want to allow in Lambda, similar to the following:

   ```json
   {
     "FunctionName": "MyCodeCommitFunction",
     "StatementId": "1",
     "Action": "lambda:InvokeFunction",
     "Principal": "codecommit.amazonaws.com",
     "SourceArn": "arn:aws:codecommit:us-east-1:80398EXAMPLE:MyDemoRepo",
     "SourceAccount": "80398EXAMPLE"
   }
   ```
2. Save the file as a JSON file with a name that is easy for you to remember (for example, AllowAccessfromMyDemoRepo.json).

3. At the terminal (Linux, macOS, or Unix) or command line (Windows), run the `aws lambda add-permissions` command to add a permission to the resource policy associated with your Lambda function, using the JSON file you just created:

   ```bash
   aws lambda add-permission --cli-input-json file://AllowAccessfromMyDemoRepo.json
   ```

   This command returns the JSON of the policy statement you just added, similar to the following:

   ```json
   {
     "Statement": {
       "Condition": {
         "StringEquals": {
           "AWS:SourceAccount": "80398EXAMPLE",
           "ArnLike": {
           }
         },
         "Action": ["lambda:InvokeFunction"]
       },
       "Effect": "Allow",
       "Principal": {
         "Service": "codecommit.amazonaws.com"
       }
     }
   }
   ```

   For more information about resource policies for Lambda functions, see AddPermission and The Pull/Push Event Models in the Lambda User Guide.

4. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.

5. In the Dashboard navigation pane, choose Roles, and in the list of roles, select lambda_basic_execution.

6. On the summary page for the role, choose the Permissions tab, and in the Inline Policies section, choose Create Role Policy.

7. On the Set Permissions page, choose Policy Generator, and then choose Select.

8. On the Edit Permissions page, do the following:
   - In Effect, choose Allow.
   - In AWS Service, choose AWS CodeCommit.
   - In Actions, select GetRepository.
   - In Amazon Resource Name (ARN), type the ARN for the repository (for example, arn:aws:codecommit:us-east-1:80398EXAMPLE:MyDemoRepo).

   Choose Add Statement, and then choose Next Step.


   Your policy statement should look similar to the following example:

   ```json
   {
     "Version": "2012-10-17",
     "Statement": [
       {
         "Sid": "Stmt11111111",
         "Effect": "Allow",
         "Action": ["codecommit:GetRepository"],
         "Resource": ["arn:aws:codecommit:us-east-1:80398EXAMPLE:MyDemoRepo"]
       }
     ]
   }
   ```
Create a Trigger for the Lambda Function in an AWS CodeCommit Repository (Console)

After you have created the Lambda function, you can create a trigger in AWS CodeCommit that will run the function in response to the repository events you specify.

Note
Before you can successfully test or run the trigger for the example, you must configure the policies that allow AWS CodeCommit to invoke the function and the Lambda function to get information about the repository. For more information, see To allow AWS CodeCommit to run a Lambda function (p. 100).

To create a trigger for a Lambda function in the AWS CodeCommit console

2. From the list of repositories, choose the repository where you want to create triggers for repository events.
3. In the navigation pane for the repository, choose Settings. In Settings, choose Triggers.
4. Choose Create trigger.
5. In the Create trigger pane, do the following:
   • In Trigger name, type a name for the trigger (for example, MyLambdaFunctionTrigger).
   • In Events, choose the repository events that will trigger the Lambda function.
     
     If you choose All repository events, you cannot choose any other events. If you want to choose a subset of events, clear All repository events, and then choose the events you want from the list. For example, if you want the trigger to run only when a user creates a tag or a branch in the AWS CodeCommit repository, remove All repository events, and then choose Create branch or tag.
     
     • If you want the trigger to apply to all branches of the repository, in Branches, choose All branches. Otherwise, choose Specific branches. The default branch for the repository will be added by default. You can keep or delete this branch from the list. Choose up to ten branch names from the list of repository branches.
   • In Send to, choose AWS Lambda.
   • In Lambda function ARN, choose the function name from the list, or choose Add an AWS Lambda function ARN and then type the ARN for the function.
   • In Custom data, optionally provide information you want included in the Lambda function (for example, the name of the IRC channel used by developers to discuss development in the repository). This field is a string. It cannot be used to pass any dynamic parameters.
6. Optionally, choose Test trigger. This option will attempt to invoke the function with sample data about your repository, including the most recent commit ID for the repository. (If no commit history exists, sample values consisting of zeroes will be generated instead.) This will help you confirm you have correctly configured access between AWS CodeCommit and the Lambda function.
7. Choose Create to finish creating the trigger.
8. To verify the functionality of the trigger, make and push a commit to the repository where you configured the trigger. You should see a response from the Lambda function on the Monitoring tab for that function in the Lambda console.

Create a Trigger to a Lambda Function for an AWS CodeCommit Repository (AWS CLI)

You can also use the command line to create a trigger for a Lambda function in response to AWS CodeCommit repository events, such as when someone pushes a commit to your repository.
To create a trigger for an Lambda function

1. Open a plain-text editor and create a JSON file that specifies:
   
   - The Lambda function name.
   - The repository and branches you want to monitor with this trigger. (If you do not specify any branches, the trigger will apply to all branches in the repository.)
   - The events that will activate this trigger.

   Save the file.

   For example, if you want to create a trigger for a repository named `MyDemoRepo` that will publish all repository events to a Lambda function named `MyCodeCommitFunction` for two branches, `master` and `preprod`:

   ```json
   {
     "repositoryName": "MyDemoRepo",
     "triggers": [
       {
         "name": "MyLambdaFunctionTrigger",
         "customData": "",
         "branches": [
           "master",
           "preprod"
         ],
         "events": [
           "all"
         ]
       }
     ]
   }
   ```

   There must be a trigger block in the JSON for each trigger for a repository. To create more than one trigger for a repository, include additional blocks in the JSON. Remember that all triggers created in this file are for the specified repository. You cannot create triggers for multiple repositories in a single JSON file. For example, if you wanted to create two triggers for a repository, you could create a JSON file with two trigger blocks. In the following example, no branches are specified in the second trigger block, so that trigger will apply to all branches:

   ```json
   {
     "repositoryName": "MyDemoRepo",
     "triggers": [
       {
         "name": "MyLambdaFunctionTrigger",
         "customData": "",
         "branches": [
           "master",
           "preprod"
         ],
         "events": [
           "all"
         ]
       },
       {
         "name": "MyOtherLambdaFunctionTrigger",
         "destinationArn": "arn:aws:lambda:us-east-1:80398EXAMPLE:function:MyOtherCodeCommitFunction",
         "customData": "",
         "branches": [
           "master",
           "preprod"
         ],
         "events": [
           "all"
         ]
       }
     ]
   }
   ```
You can create triggers for events you specify, such as when a commit is pushed to a repository. Event types include:

- **all** for all events in the specified repository and branches.
- **updateReference** for when commits are pushed to the specified repository and branches.
- **createReference** for when a new branch or tag is created in the specified repository.
- **deleteReference** for when a branch or tag is deleted in the specified repository.

**Note**
You can use more than one event type in a trigger. However, if you specify `all`, you cannot specify other events.

To see the full list of valid event types, at the terminal or command prompt, type `aws codecommit put-repository-triggers help`.

In addition, you can include a string in `customData` (for example, an IRC channel name developers use when discussing development in this repository). This field is a string. It cannot be used to pass any dynamic parameters. This string will be appended as an attribute to the AWS CodeCommit JSON returned in response to the trigger.

2. At a terminal or command prompt, optionally run the `test-repository-triggers` command. For example, the following is used to test that the JSON file named `trigger.json` is valid and that AWS CodeCommit can trigger the Lambda function. This test uses sample data to trigger the function if no real data is available.

```
aws codecommit test-repository-triggers --cli-input-json file://trigger.json
```

If successful, this command returns information similar to the following:

```
{
    "successfulExecutions": [
        "MyLambdaFunctionTrigger"
    ],
    "failedExecutions": []
}
```

3. At a terminal or command prompt, run the `put-repository-triggers` command to create the trigger in AWS CodeCommit. For example, to use a JSON file named `trigger.json` to create the trigger:

```
aws codecommit put-repository-triggers --cli-input-json file://trigger.json
```

This command returns a configuration ID, similar to the following:

```
{
    "configurationId": "0123456-I-AM-AN-EXAMPLE"
}
```

4. To view the configuration of the trigger, run the `get-repository-triggers` command, specifying the name of the repository:
aws codecommit get-repository-triggers --repository-name MyDemoRepo

This command returns the structure of all triggers configured for the repository, similar to the following:

```json
{
    "configurationId": "0123456-I-AM-AN-EXAMPLE",
    "triggers": [
        {
            "events": [
                "all"
            ],
            "destinationArn": "arn:aws:lambda:us-east-1:80398EXAMPLE:MyCodeCommitFunction",
            "branches": [
                "master",
                "preprod"
            ],
            "name": "MyLambdaFunctionTrigger",
            "customData": "Project ID 12345"
        }
    ]
}
```

5. To test the functionality of the trigger, make and push a commit to the repository where you configured the trigger. You should see a response from the Lambda function on the Monitoring tab for that function in the Lambda console.

## Edit Triggers for an AWS CodeCommit Repository

You can edit the triggers that have been created for an AWS CodeCommit repository. You can change the events and branches for the trigger, the action taken in response to the event, and other settings.

### Topics
- [Edit a Trigger for a Repository (Console) (p. 105)](#)
- [Edit a Trigger for a Repository (AWS CLI) (p. 105)](#)

### Edit a Trigger for a Repository (Console)

2. From the list of repositories, choose the repository where you want to edit a trigger for repository events.
3. In the navigation pane for the repository, choose Settings. In Settings, choose Triggers.
4. From the list of triggers for the repository, select the trigger you want to edit, and then choose Edit.
5. Make the changes you want to the trigger, and then choose Update to save your changes.

### Edit a Trigger for a Repository (AWS CLI)

1. At a terminal (Linux, macOS, or Unix) or command prompt (Windows), run the `get-repository-triggers` command to create a JSON file with the structure of all of the triggers configured for your repository. For example, to create a JSON file named `MyTriggers.json` with the structure of all of the triggers configured for a repository named `MyDemoRepo`:
aws codecommit get-repository-triggers --repository-name MyDemoRepo >MyTriggers.json

This command returns nothing, but a file named MyTriggers.json is created in the directory where you ran the command.

2. Edit the JSON file in a plain-text editor and make changes to the trigger block of the trigger you want to edit. Replace the configurationId pair with a repositoryName pair. Save the file.

For example, if you want to edit a trigger named MyFirstTrigger in the repository named MyDemoRepo so that it applies to all branches, you would replace configurationId with repositoryName, and remove the specified master and preprod branches in red italic text. By default, if no branches are specified, the trigger will apply to all branches in the repository:

```
{
  "repositoryName": "MyDemoRepo",
  "triggers": [
    {
      "branches": [
        "master",
        "preprod"
      ],
      "name": "MyFirstTrigger",
      "customData": "",
      "events": ["all"],
    }
  ]
}
```

3. At the terminal or command line, run the put-repository-triggers command. This will update all triggers for the repository, including the changes you made to the MyFirstTrigger trigger:

```
aws codecommit put-repository-triggers --repository-name MyDemoRepo
file://MyTriggers.json
```

This command returns a configuration ID, similar to the following:

```
{
  "configurationId": "0123456-I-AM-AN-EXAMPLE"
}
```

Test Triggers for an AWS CodeCommit Repository

You can test the triggers that have been created for an AWS CodeCommit repository. Testing involves running the trigger with sample data from your repository, including the most recent commit ID. If no commit history exists for the repository, sample values consisting of zeroes will be generated instead. Testing triggers helps you confirm you have correctly configured access between AWS CodeCommit and the target of the trigger, whether that is an AWS Lambda function or an Amazon Simple Notification Service notification.

**Topics**
- Test a Trigger for a Repository (Console) (p. 107)
- Test a Trigger for a Repository (AWS CLI) (p. 107)
Test a Trigger for a Repository (Console)

2. From the list of repositories, choose the repository where you want to test a trigger for repository events.
3. In the navigation pane for the repository, choose **Settings**. In **Settings**, choose **Triggers**.
4. Choose the trigger you want to edit from the list of triggers, and then choose **Edit**.
5. In the **Edit trigger** dialog box, choose **Test trigger**. You will see a success or failure message. If successful, you will also see a corresponding action response from the Lambda function or the Amazon SNS topic.

Test a Trigger for a Repository (AWS CLI)

1. At a terminal (Linux, macOS, or Unix) or command prompt (Windows), run the `get-repository-triggers` command to create a JSON file with the structure of all of the triggers configured for your repository. For example, to create a JSON file named `TestTrigger.json` with the structure of all of the triggers configured for a repository named `MyDemoRepo`:

   ```bash
   aws codecommit get-repository-triggers --repository-name MyDemoRepo > TestTrigger.json
   ```

   This command creates a file named `TestTriggers.json` in the directory where you ran the command.

2. Edit the JSON file in a plain-text editor and make the changes to the trigger statement. Replace the `configurationId` pair with a `repositoryName` pair. Save the file.

   For example, if you want to test a trigger named `MyFirstTrigger` in the repository named `MyDemoRepo` so that it applies to all branches, you would replace the `configurationId` pair with `repositoryName` and then save a file that looks similar to the following as `TestTrigger.json`:

   ```json
   {
   "repositoryName": "MyDemoRepo",
   "triggers": [
   {
   "branches": [
   "master",
   "preprod"
   ],
   "name": "MyFirstTrigger",
   "customData": "",
   "events": [
   "all"
   ]
   }
   ]
   }
   ```

3. At the terminal or command line, run the `test-repository-triggers` command. This will update all triggers for the repository, including the changes you made to the `MyFirstTrigger` trigger:

   ```bash
   aws codecommit test-repository-triggers --cli-input-json file://TestTrigger.json
   ```

   This command returns a response similar to the following:
Delete Triggers from a Repository

You might want to delete triggers if they are no longer being used. You cannot undo the deletion of a trigger, but you can re-create one.

Note
If you configured one or more triggers for your repository, deleting the repository does not delete the Amazon SNS topics or Lambda functions you configured as the targets of those triggers. Be sure to delete those resources, too, if they are no longer needed.

Topics
• Delete a Trigger from a Repository (Console) (p. 108)
• Delete a Trigger from a Repository (AWS CLI) (p. 108)

Delete a Trigger from a Repository (Console)

2. From the list of repositories, choose the repository where you want to delete triggers for repository events.
3. In the navigation pane for the repository, choose Settings. In Settings, choose Triggers.
4. Select the triggers you want to delete from the list of triggers, and then choose Delete.
5. In the dialog box, choose Delete to confirm.

Delete a Trigger from a Repository (AWS CLI)

1. At a terminal (Linux, macOS, or Unix) or command prompt (Windows), run the `get-repository-triggers` command to create a JSON file with the structure of all of the triggers configured for your repository. For example, to create a JSON file named `MyTriggers.json` with the structure of all of the triggers configured for a repository named `MyDemoRepo`:

   ```sh
   aws codecommit get-repository-triggers --repository-name MyDemoRepo >MyTriggers.json
   ```

   This command creates a file named `MyTriggers.json` in the directory where you ran the command.
2. Edit the JSON file in a plain-text editor and remove the trigger block for the trigger you want to delete. Replace the `configurationId` pair with a `repositoryName` pair. Save the file.

   For example, if you want to remove a trigger named `MyFirstTrigger` from the repository named `MyDemoRepo`, you would replace `configurationId` with `repositoryName`, and remove the statement in red italic text:

   ```json
   { "repositoryName": "MyDemoRepo", "triggers": ["MyFirstTrigger"]
   ```
3. At the terminal or command line, run the `put-repository-triggers` command. This will update the triggers for the repository and delete the `MyFirstTrigger` trigger:

```
aws codecommit put-repository-triggers --repository-name MyDemoRepo file://MyTriggers.json
```

This command returns a configuration ID, similar to the following:

```
{
    "configurationId": "0123456-I-AM-AN-EXAMPLE"
}
```

**Note**

To delete all triggers for a repository named `MyDemoRepo`, your JSON file would look similar to this:

```
{
    "repositoryName": "MyDemoRepo",
    "triggers": []
}
```

---

**View AWS CodeCommit Repository Details**

To view information about available repositories, you can use:

- Git from a local repo connected to the AWS CodeCommit repository.
- The AWS CLI.
- The AWS CodeCommit console.

Before you follow these instructions, complete the steps in *Setting Up* (p. 4).

**Topics**
Use the AWS CodeCommit Console to View Repository Details

Use the AWS CodeCommit console to quickly view all repositories created with your AWS account.

2. Choose the name of the repository from the list.
3. Do one of the following:
   - To view the URL for cloning the repository, in the navigation pane, choose Code, choose Clone URL, and then choose the protocol you want to use when cloning the repository.
   - To view configurable details for the repository, in the navigation pane, choose Settings.

Note
If you are signed in as an IAM user, you can configure and save your preferences for viewing code and other console settings. For more information, see Working with User Preferences (p. 208).

Use Git to View AWS CodeCommit Repository Details

To use Git from a local repo to view details about AWS CodeCommit repositories, run the git remote show command.

Before you perform these steps, connect the local repo to the AWS CodeCommit repository. For instructions, see Connect to a Repository (p. 78).

1. Run the git remote show remote-name command, where remote-name is the alias of the AWS CodeCommit repository (by default, origin).

   Tip
   To get a list of AWS CodeCommit repository names along with their URLs, run the git remote -v command.

   For example, to view details about the AWS CodeCommit repository with the alias origin:

   ```
   git remote show origin
   ```

2. For HTTPS:

   ```
   * remote origin
     Fetch URL: https://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo
     Push URL: https://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo
     HEAD branch: (unknown)
     Remote branches:
       MyNewBranch tracked
       master tracked
     Local ref configured for 'git pull':
       MyNewBranch merges with remote MyNewBranch (up to date)
     Local refs configured for 'git push':
       MyNewBranch pushes to MyNewBranch (up to date)
       master pushes to master (up to date)
   ```
For SSH:

* remote origin
  Fetch URL: ssh://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo
  Push URL: ssh://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo

HEAD branch: (unknown)
Remote branches:
  MyNewBranch tracked
  master tracked
Local ref configured for 'git pull':
  MyNewBranch merges with remote MyNewBranch (up to date)
Local refs configured for 'git push':
  MyNewBranch pushes to MyNewBranch (up to date)
  master pushes to master (up to date)

Tip
To look up the SSH key ID for your IAM user, open the IAM console and expand Security Credentials on the IAM user details page. The SSH key ID can be found in SSH Keys for AWS CodeCommit.

For more options, see your Git documentation.

**Use the AWS CLI to View AWS CodeCommit Repository Details**

To use AWS CLI commands with AWS CodeCommit, install the AWS CLI. For more information, see Command Line Reference (p. 295).

To use the AWS CLI to view repository details, run the following commands:

- list-repositories (p. 111) to view a list of AWS CodeCommit repository names and their corresponding IDs.
- get-repository (p. 112) to view information about a single AWS CodeCommit repository.
- batch-get-repositories (p. 112) to view information about multiple repositories in AWS CodeCommit.

**To view a list of AWS CodeCommit repositories**

1. Run the list-repositories command:

   ```
   aws codecommit list-repositories
   ```

   You can use the optional --sort-by or --order options to change the order of returned information.

2. If successful, this command outputs a repositories object that contains the names and IDs of all repositories in AWS CodeCommit associated with the AWS account.

   Here is some example output based on the preceding command:

   ```json
   {
     "repositories": [
       {
         "repositoryName": "MyDemoRepo",
         "repositoryId": "f7579e13-b83e-4027-aaef-650c0EXAMPLE",
       },
     ],
   }
   ```
To view details about a single AWS CodeCommit repository

1. Run the `get-repository` command, specifying the name of the AWS CodeCommit repository with the `--repository-name` option.

   **Tip**
   To get the AWS CodeCommit repository's name, run the `list-repositories` (p. 111) command.

   For example, to view details about an AWS CodeCommit repository named `MyDemoRepo`:

   ```bash
   aws codecommit get-repository --repository-name MyDemoRepo
   ```

2. If successful, this command outputs a `repositoryMetadata` object with the following information:
   - The repository's name (`repositoryName`).
   - The repository's description (`repositoryDescription`).
   - The repository's unique, system-generated ID (`repositoryId`).
   - The ID of the AWS account associated with the repository (`accountId`).

Here is some example output, based on the preceding example command:

```json
{
   "repositoryMetadata": {
      "creationDate": 1429203623.625,
      "defaultBranch": "master",
      "repositoryName": "MyDemoRepo",
      "cloneUrlSsh": "ssh://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo",
      "lastModifiedDate": 1430783812.0869999,
      "repositoryDescription": "My demonstration repository",
      "cloneUrlHttp": "https://codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo",
      "repositoryId": "f7579e13-b83e-4027-aaef-650c0EXAMPLE",
      "accountId": "111111111111"
   }
}
```

To view details about multiple AWS CodeCommit repositories

1. Run the `batch-get-repositories` command with the `--repository-names` option. Add a space between each AWS CodeCommit repository name.

   **Tip**
   To get the names of the repositories in AWS CodeCommit, run the `list-repositories` (p. 111) command.
For example, to view details about two AWS CodeCommit repositories named `MyDemoRepo` and `MyOtherDemoRepo`:

```bash
aws codecommit batch-get-repositories --repository-names MyDemoRepo MyOtherDemoRepo
```

2. If successful, this command outputs an object with the following information:

- A list of any AWS CodeCommit repositories that could not be found (`repositoriesNotFound`).
- A list of AWS CodeCommit repositories (`repositories`). Each AWS CodeCommit repository's name is followed by:
  - The repository's description (`repositoryDescription`).
  - The repository's unique, system-generated ID (`repositoryId`).
  - The ID of the AWS account associated with the repository (`accountId`).

Here is some example output, based on the preceding example command:

```json
{
    "repositoriesNotFound": [],
    "repositories": [
        {
            "creationDate": 1429203623.625,
            "defaultBranch": "master",
            "repositoryName": "MyDemoRepo",
            "cloneUrlSsh": "ssh://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo",
            "lastModifiedDate": 1430783812.0869999,
            "repositoryDescription": "My demonstration repository",
            "cloneUrlHttp": "https://codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo",
            "repositoryId": "f7579e13-b83e-4027-aaef-650cEXAMPLE",
            "accountId": "111111111111"
        },
        {
            "creationDate": 1429203623.627,
            "defaultBranch": "master",
            "repositoryName": "MyOtherDemoRepo",
            "cloneUrlSsh": "ssh://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyOtherDemoRepo",
            "lastModifiedDate": 1430783812.0889999,
            "repositoryDescription": "My other demonstration repository",
            "cloneUrlHttp": "https://codecommit.us-east-2.amazonaws.com/v1/repos/MyOtherDemoRepo",
            "repositoryId": "cfc29ac4-b0cb-44dc-9990-f6f51EXAMPLE",
            "Arn": "arn:aws:codecommit:us-east-2:80398EXAMPLE:MyOtherDemoRepo",
            "accountId": "111111111111"
        }
    ],
    "repositoriesNotFound": []
}
```

Change AWS CodeCommit Repository Settings

To change the settings of an AWS CodeCommit repository, such as its description or name, you can use the AWS CLI and the AWS CodeCommit console.
Important
Changing a repository's name may break any local repos that use the old name in their remote URL. Run the `git remote set-url` command to update the remote URL to use the new repository's name.

Topics
- Use the AWS CodeCommit Console to Change Repository Settings (p. 114)
- Use the AWS CLI to Change AWS CodeCommit Repository Settings (p. 115)

Use the AWS CodeCommit Console to Change Repository Settings

To use the AWS CodeCommit console to change an AWS CodeCommit repository's settings in AWS CodeCommit, follow these steps.

2. In the list of repositories, choose the name of the repository where you want to change settings.
3. In the navigation pane, choose **Settings**.
4. To change the name of the repository, in the **General** tab, in **Repository name**, type a new name in the **Name** text box, choose **Change name**, and then choose **Change name** again.

   Important
   Changing the name of the AWS CodeCommit repository will change the SSH and HTTPS URLs that users need to connect to the repository. Users will not be able to connect to this repository until they update their connection settings. Also, because the repository's ARN will change, changing the repository name will invalidate any IAM user policies that rely on this repository's ARN.

   To connect to the repository after the name is changed, each user must use the `git remote set-url` command and specify the new URL to use. For example, if you changed the name of the repository from MyDemoRepo to MyRenamedDemoRepo, users who use HTTPS to connect to the repository would run the following Git command:

   ```
git remote set-url origin https://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyRenamedDemoRepo
   ```

   Users who use SSH to connect to the repository would run the following Git command:

   ```
git remote set-url origin ssh://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyRenamedDemoRepo
   ```

   For more options, see your Git documentation.

5. To change the repository's description, modify the text in the **Description** text box, and then choose **Save changes**.

   Note
   The description field accepts all HTML characters and all valid Unicode characters. If you are an application developer using the GetRepository or BatchGetRepositories APIs and plan to display the repository description field in a web browser, see the AWS CodeCommit API Reference.

6. To change the default branch, choose a different branch from the **Default branch** drop-down list, choose **Save changes**, and then choose **Change default**.

7. To delete the repository, choose **Delete repository**. In the box next to **Type the name of the repository to confirm deletion**, type the repository's name, and then choose **Delete**.
Use the AWS CLI to Change AWS CodeCommit Repository Settings

To use AWS CLI commands with AWS CodeCommit, install the AWS CLI. For more information, see Command Line Reference (p. 295).

To use AWS CLI to change an AWS CodeCommit repository's settings in AWS CodeCommit, run one or more of the following commands:

- `update-repository-description` (p. 115) to change an AWS CodeCommit repository's description.
- `update-repository-name` (p. 115) to change an AWS CodeCommit repository's name.

To change an AWS CodeCommit repository's description

1. Run the `update-repository-description` command, specifying:
   - The name of the AWS CodeCommit repository (with the `--repository-name` option).
     
   **Tip**
   To get the name of the AWS CodeCommit repository, run the `list-repositories` (p. 111) command.
   - The new repository description (with the `--repository-description` option).
     
   **Note**
   The description field accepts all HTML characters and all valid Unicode characters. If you are an application developer using the GetRepository or BatchGetRepositories APIs and plan to display the repository description field in a web browser, see the AWS CodeCommit API Reference.

   For example, to change the description for the AWS CodeCommit repository named `MyDemoRepo` to "This description was changed":

   ```bash
   aws codecommit update-repository-description --repository-name MyDemoRepo --repository-description "This description was changed"
   ```

   This command produces output only if there are errors.

2. To verify the changed description, run the `get-repository` command, specifying the name of the AWS CodeCommit repository whose description you changed with the `--repository-name` option.

   The output of the command will show the changed text in `repositoryDescription`.

To change an AWS CodeCommit repository's name

1. Run the `update-repository-name` command, specifying:
   - The current name of the AWS CodeCommit repository (with the `--old-name` option).
Sync Changes Between a Local Repo and an AWS CodeCommit Repository

You use Git to synchronize changes between a local repo and the AWS CodeCommit repository connected to the local repo.

To push changes from the local repo to the AWS CodeCommit repository, run `git push remote-name branch-name`.

To pull changes to the local repo from the AWS CodeCommit repository, run `git pull remote-name branch-name`.

For both pushing and pulling, `remote-name` is the nickname the local repo uses for the AWS CodeCommit repository; `branch-name` is the name of the branch on the AWS CodeCommit repository to push to or pull from.

**Tip**
To get the nickname the local repo uses for the AWS CodeCommit repository, run `git remote`. To get a list of branch names, run `git branch`. An asterisk (*) appears next to the name of the current branch. (Alternatively, run `git status` to show the current branch name.)

**Note**
If you cloned the repository, from the local repo's perspective, `remote-name` is not the name of the AWS CodeCommit repository. When you clone a repository, `remote-name` is set automatically to `origin`.

For example, to push changes from the local repo to the `master` branch in the AWS CodeCommit repository with the nickname `origin`:

```sh
git push origin master
```

Similarly, to pull changes to the local repo from the `master` branch in the AWS CodeCommit repository with the nickname `origin`:

```sh
git pull origin master
```
Push Commits to Two Repositories

```
> git pull origin master
```

**Tip**

If you add the `-u` option to `git push`, you will set upstream tracking information. For example, if you run `git push -u origin master`, in the future you can run `git push` and `git pull` without `remote-name branch-name`. To get upstream tracking information, run `git remote show remote-name` (for example, `git remote show origin`).

For more options, see your Git documentation.

---

**Push Commits to an Additional Git Repository**

You can configure your local repo to push changes to two remote repositories. For example, you might want to continue using your existing Git repository solution while you try out AWS CodeCommit. Follow these basic steps to push changes in your local repo to both AWS CodeCommit and a separate Git repository.

**Tip**

If you do not have a current Git repository, you can create an empty one on a service other than AWS CodeCommit and then migrate your AWS CodeCommit repository to it. You should follow steps similar to the ones in [Migrate to AWS CodeCommit](#) (p. 211).

1. From the command prompt or terminal, switch to your local repo directory and run the `git remote -v` command. You should see output similar to the following:

   For HTTPS:
   
   ```
   origin  https://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo (fetch)
   origin  https://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo (push)
   ```

   For SSH:
   
   ```
   origin  ssh://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo (fetch)
   origin  ssh://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo (push)
   ```

2. Run the `git remote set-url --add --push origin git-repository-name` command where `git-repository-name` is the URL and name of the Git repository where you want to host your code. This changes the push destination of `origin` to that Git repository.

   **Note**
   
   `git remote set-url --add --push` overrides the default URL for pushes, so you will have to run this command twice, as demonstrated in later steps.

   For example, the following command changes the push of `origin` to `some-URL/MyDestinationRepo`:

   ```
   > git remote set-url --add --push origin some-URL/MyDestinationRepo
   ```

   This command returns nothing.

   **Tip**
   
   If you are pushing to a Git repository that requires credentials, make sure you configure those credentials in a credential helper or in the configuration of the `some-URL` string; otherwise, your pushes to that repository will fail.

3. Run the `git remote -v` command again, which should create output similar to the following:

   For HTTPS:
   
   ```
   origin  https://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo (fetch)
   origin  https://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo (push)
   origin  ssh://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo (fetch)
   origin  ssh://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo (push)
   ```

   For SSH:
   
   ```
   origin  ssh://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo (fetch)
   origin  ssh://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo (push)
   origin  ssh://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo (push)
   ```
Push Commits to Two Repositories

4. Now add the AWS CodeCommit repository. Run `git remote set-url --add --push origin` again, this time with the URL and repository name of your AWS CodeCommit repository.

For example, the following command adds the push of `origin` to `https://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo`:

For HTTPS:

```
```

For SSH:

```
git remote set-url --add --push origin ssh://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo
```

This command returns nothing.

5. Run the `git remote -v` command again, which should create output similar to the following:

For HTTPS:

```
origin https://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo (fetch)
origin some-URL/MyDestinationRepo (push)
origin https://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo (push)
```

For SSH:

```
origin ssh://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo (fetch)
origin some-URL/MyDestinationRepo (push)
origin ssh://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo (push)
```

You now have two Git repositories as the destination for your pushes, but your pushes will go to `some-URL/MyDestinationRepo` first. If the push to that repository fails, your commits will not be pushed to either repository.

Tip

If the other repository requires credentials you want to enter manually, consider changing the order of the pushes so that you push to AWS CodeCommit first. Run `git remote set-url --delete` to delete the repository that is pushed to first, and then run `git remote set-url --add` to add it again so that it becomes the second push destination in the list.

For more options, see your Git documentation.

6. To verify you are now pushing to both remote repositories, use a text editor to create the following text file in your local repo:

```
bees.txt
-------
API Version 2015-04-13
```

118
Bees are flying insects closely related to wasps and ants, and are known for their role in pollination and for producing honey and beeswax.

7. **Run** `git add` to stage the change in your local repo:

    ```
    git add bees.txt
    ```

8. **Run** `git commit` to commit the change in your local repo:

    ```
    git commit -m "Added bees.txt"
    ```

9. To push the commit from the local repo to your remote repositories, run `git push -u remote-name branch-name` where `remote-name` is the nickname the local repo uses for the remote repositories and `branch-name` is the name of the branch to push to the repository.

    **Tip**
    You only have to use the `-u` option the first time you push. The upstream tracking information will be set.

    For example, running `git push -u origin master` would show the push went to both remote repositories in the expected branches, with output similar to the following:

    **For HTTPS:**

    ```
    Counting objects: 5, done.
    Delta compression using up to 4 threads.
    Compressing objects: 100% (3/3), done.
    Writing objects: 100% (3/3), 5.61 KiB | 0 bytes/s, done.
    Total 3 (delta 1), reused 0 (delta 0)
    To some-URL/MyDestinationRepo
    a5ba4ed..250f6c3  master -> master
    Counting objects: 5, done.
    Delta compression using up to 4 threads.
    Compressing objects: 100% (3/3), done.
    Writing objects: 100% (3/3), 5.61 KiB | 0 bytes/s, done.
    Total 3 (delta 1), reused 0 (delta 0)
    remote:
    To https://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo
    a5ba4ed..250f6c3  master -> master
    ```

    **For SSH:**

    ```
    Counting objects: 5, done.
    Delta compression using up to 4 threads.
    Compressing objects: 100% (3/3), done.
    Writing objects: 100% (3/3), 5.61 KiB | 0 bytes/s, done.
    Total 3 (delta 1), reused 0 (delta 0)
    To some-URL/MyDestinationRepo
    a5ba4ed..250f6c3  master -> master
    Counting objects: 5, done.
    Delta compression using up to 4 threads.
    Compressing objects: 100% (3/3), done.
    Writing objects: 100% (3/3), 5.61 KiB | 0 bytes/s, done.
    Total 3 (delta 1), reused 0 (delta 0)
    remote:
    To ssh://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo
    a5ba4ed..250f6c3  master -> master
    ```

    **For more options, see your Git documentation.**
Configure Cross-Account Access to an AWS CodeCommit Repository

You can configure access to AWS CodeCommit repositories for IAM users and groups in another AWS account. This is often referred to as cross-account access. This section provides examples and step-by-step instructions for configuring cross-account access for a repository named MySharedDemoRepo in the US East (Ohio) Region in an AWS account (referred to as AccountA) to IAM users who belong to an IAM group named DevelopersWithCrossAccountRepositoryAccess in another AWS account (referred to as AccountB).

This section is divided into three parts:

• Actions for the Administrator in AccountA.
• Actions for the Administrator in AccountB.
• Actions for the repository user in AccountB.

To configure cross-account access:

• The administrator in AccountA signs in as an IAM user with the permissions required to create and manage repositories in AWS CodeCommit and create roles in IAM. If you are using managed policies, apply IAMFullAccess and AWSCodeCommitFullAccess to this IAM user.

The example account ID for AccountA is 111122223333.

• The administrator in AccountB, signs in as an IAM user with the permissions required to create and manage IAM users and groups, and to configure policies for users and groups. If you are using managed policies, apply IAMFullAccess to this IAM user.

The example account ID for AccountB is 888888888888.

• The repository user in AccountB, to emulate the activities of a developer, signs in as an IAM user who is a member of the IAM group created to allow access to the AWS CodeCommit repository in AccountA. This account must be configured with:
  • AWS console access.
  • An access key and secret key to use when installing and configuring the AWS CLI credential helper on a local computer or Amazon EC2 instance, as described in For HTTPS Connections on Linux, macOS, or Unix with the AWS CLI Credential Helper (p. 35) or For HTTPS Connections on Windows with the AWS CLI Credential Helper (p. 39).

For more information, see IAM users.

Topics

• Cross-Account Repository Access: Actions for the Administrator in AccountA (p. 120)
• Cross Account Repository Access: Actions for the Administrator in AccountB (p. 123)
• Cross-Account Repository Access: Actions for the Repository User in AccountB (p. 124)

Cross-Account Repository Access: Actions for the Administrator in AccountA

To allow users or groups in AccountB to access a repository in AccountA, an AccountA administrator must:
• Create a policy in AccountA that grants access to the repository.
• Create a role in AccountA that can be assumed by IAM users and groups in AccountB.
• Attach the policy to the role.

The following sections provide steps and examples.

Topics
• Step 1: Create a Policy for Repository Access in AccountA (p. 121)
• Step 2: Create a Role for Repository Access in AccountA (p. 122)

Step 1: Create a Policy for Repository Access in AccountA

You can create a policy in AccountA that grants access to the repository in AccountB. Depending on the level of access you want to allow, do one of the following:

• Configure the policy to allow AccountB users access to a specific repository, but do not allow them to view a list of all repositories in AccountA.
• Configure additional access to allow AccountB users to choose the repository from a list of all repositories in AccountA.

To create a role for repository access

1. Sign in to the AWS Management Console as an IAM user with permissions to create roles in AccountA.
2. Open the IAM console at https://console.aws.amazon.com/iam/.
3. In the navigation pane, choose Policies.
4. Choose Create policy.
5. Choose the JSON tab, and paste the following JSON policy document into the JSON text box. Replace `us-east-2` with the region for the repository, `111122223333` with the account ID for AccountA, and `MySharedDemoRepo` with the name for your AWS CodeCommit repository in AccountA:

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codecommit:BatchGet*",
        "codecommit:Create*",
        "codecommit:DeleteBranch",
        "codecommit:Get*",
        "codecommit:List*",
        "codecommit:Describe*",
        "codecommit:Put*",
        "codecommit:Post*",
        "codecommit:Merge*",
        "codecommit:Test*",
        "codecommit:Update*",
        "codecommit:GitPull",
        "codecommit:GitPush"
      ],
      "Resource": [
        "arn:aws:codecommit:us-east-2:111122223333:MySharedDemoRepo"
      ]
    }
  ]
}
```

API Version 2015-04-13
121
If you want users who assume this role to be able to view a list of repositories on the AWS CodeCommit console home page, add an additional statement to the policy, as follows:

```json

  ]}

```

This access makes it easier for users who assume this role to find the repository to which they have access. They can choose the name of the repository from the list and be directed to the home page of the shared repository (Code). Users cannot access any of the other repositories they see in the list, but they can view the repositories in AccountA on the Dashboard page.

If you do not want to allow users who assume the role to be able to view a list of all repositories in AccountA, use the first policy example, but make sure that you send those users a direct link to the home page of the shared repository in the AWS CodeCommit console.

6. Choose Review policy. The policy validator reports syntax errors (for example, if you forget to replace the example AWS account ID and repository name with your AWS account ID and repository name).

7. On the Review policy page, type a name for the policy (for example, CrossAccountAccessForMySharedDemoRepo). You can also provide an optional description for this policy. Choose Create policy.

### Step 2: Create a Role for Repository Access in AccountA

After you have configured a policy, create a role that IAM users and groups in AccountB can assume, and attach the policy to that role.
To create a policy for repository access

1. In the IAM console, choose Roles.
2. Choose Create role.
3. Choose Another AWS account.
4. In Account ID, type the AWS account ID for AccountB (for example, 888888888888). Choose Next: Permissions.
5. In Attach permissions policies, select the policy you created in the previous procedure (CrossAccountAccessForMySharedDemoRepo). Choose Next: Review.
6. In Role name, type a name for the role. For example, you could name this role MyCrossAccountRepositoryContributorRole. You can also type an optional description for the role to help others understand the purpose of the role.
7. Choose Create role.
8. Open the role you just created, and copy the role ARN (for example, arn:aws:iam::111122223333:role/MyCrossAccountRepositoryContributorRole). You need to provide this ARN to the AccountB administrator.

Cross Account Repository Access: Actions for the Administrator in AccountB

To allow users or groups in AccountB to access a repository in AccountA, the AccountB administrator must create a group in AccountB. This group must be configured with a policy that allows group members to assume the role created by the AccountA administrator.

The following sections provide steps and examples.

Topics
- Step 1: Create an IAM Group for Repository Access for AccountB Users (p. 123)
- Step 2: Create a Policy and Add Users to the IAM Group (p. 124)

Step 1: Create an IAM Group for Repository Access for AccountB Users

The simplest way to manage which IAM users in AccountB can access the AccountA repository is to create an IAM group in AccountB that has permission to assume the role in AccountA, and then add the IAM users to that group.

To create a group for cross-account repository access

1. Sign in to the AWS Management Console as an IAM user with the permissions required to create IAM groups and policies and manage IAM users in AccountB.
2. Open the IAM console at https://console.aws.amazon.com/iam/.
3. In the IAM console, choose Groups.
5. In Group Name, type a name for the group (for example, DevelopersWithCrossAccountRepositoryAccess). Choose Next Step.
6. In Attach Policy, choose Next Step. You create the cross-account policy in the next procedure. Finish creating the group.
Step 2: Create a Policy and Add Users to the IAM Group

Now that you have a group, create the policy that allows members of this group to assume the role that gives them access to the repository in AccountA. Then add to the group the IAM users in AccountB that you want to allow access in AccountA.

To create a policy for the group and add users to it

1. In the IAM console, choose Groups, and then choose the name of the group you just created (for example, DevelopersWithCrossAccountRepositoryAccess).
2. Choose the Permissions tab. Expand Inline Policies, and then choose the link to create an inline policy. (If you are configuring a group that already has an inline policy, choose Create Group Policy.)
3. Choose Custom Policy, and then choose Select.
4. In Policy Name, type a name for the policy (for example, AccessPolicyForSharedRepository).
5. In Policy Document, paste the following policy. In Resource, replace the ARN with the ARN of the policy created by the administrator in AccountA (for example, arn:aws:iam::111122223333:role/MyCrossAccountRepositoryContributorRole), and then choose Apply Policy. For more information about the policy created by the administrator in AccountA, see Step 1: Create a Policy for Repository Access in AccountA (p. 121).

```json
{
    "Version": "2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": "sts:AssumeRole",
        "Resource":
            "arn:aws:iam::111122223333:role/MyCrossAccountRepositoryContributorRole"
    }
}
```
6. Choose the Users tab. Choose Add Users to Group, and then add the AccountB IAM users. For example, you might add an IAM user with the user name Saanvi_Sarkar to the group.

Note
Users in AccountB must have programmatic access, including an access key and secret key, to configure their local computers for access to the shared AWS CodeCommit repository. If you are creating IAM users, be sure to save the access key and secret key. To ensure the security of your AWS account, the secret access key is accessible only at the time you create it.

Cross-Account Repository Access: Actions for the Repository User in AccountB

To access the repository in AccountA, users in the AccountB group must configure their local computers for repository access. The following sections provide steps and examples.

Topics
- Step 1: Configure the AWS CLI and Git for an AccountB User to Access the Repository in AccountA (p. 125)
- Step 2: Clone and Access the AWS CodeCommit Repository in AccountA (p. 126)
Step 1: Configure the AWS CLI and Git for an AccountB User to Access the Repository in AccountA

You must use the credential helper, not SSH keys or Git credentials, to access repositories in another AWS account. AccountB users must configure their computers to use the credential helper to access the shared AWS CodeCommit repository in AccountA. You cannot use SSH keys or Git credentials to access repositories in another AWS account. However, you can continue to use SSH keys or Git credentials when accessing repositories in AccountB.

To configure the AWS CLI and Git for cross-account access

1. Install the AWS CLI on the local computer. For more information, see instructions for your operating system in Installing the AWS CLI.

2. Install Git on the local computer. To install Git, we recommend websites such as Git Downloads or Git for Windows.

   When you install Git, do not install the Git Credential Manager. The Git Credential Manager is not compatible with the credential helper included with the AWS CLI.

   **Note**
   AWS CodeCommit supports Git versions 1.7.9 and later. Git is an evolving, regularly updated platform. Occasionally, a feature change might affect the way it works with AWS CodeCommit. If you encounter issues with a specific version of Git and AWS CodeCommit, review the information in Troubleshooting (p. 235).

3. From the terminal or command line, at the directory location where you want to clone the repository, run the `git config --local user.name` and `git config --local user.email` commands to set the user name and email for the commits you will make to the repository. For example:

   ```bash
   git config --local user.name "Saanvi Sarkar"
   git config --local user.email saanvi_sarkar@example.com
   ```

   These commands return nothing, but the email and user name you specify will be associated with the commits you make to the repository.

4. Run the `aws configure --profile` command to configure a profile to use when connecting to the repository in AccountA. When prompted, provide the access key and secret key for your IAM user. For example, run the following command to create an AWS CLI profile named `MyCrossAccountAccessProfile` that you use to access a repository in AccountA in US East (Ohio) (us-east-2):

   ```bash
   aws configure --profile MyCrossAccountAccessProfile
   ```

   When prompted, provide the following information:

   - AWS Access Key ID [None]: *Your-IAM-User-Access-Key*
   - AWS Secret Access Key ID [None]: *Your-IAM-User-Secret-Access-Key*
   - Default region name ID [None]: *us-east-2*
   - Default output format [None]: *json*

5. In a plain-text editor, open the `config` file, also known as the AWS CLI configuration file. Depending on your operating system, this file might be located at `~/.aws/config` on Linux, macOS, or Unix, or at `drive:\Users\USERNAME\aws\config` on Windows.

6. In the file, find the entry that corresponds to the `MyCrossAccountAccessProfile` profile you just created. It should look similar to the following:

   ```yaml
   [profile MyCrossAccountAccessProfile]
   ```
region = US East (Ohio)
output = json

Add two lines to the profile configuration, role_arn and source_profile. Provide the ARN of the role in AccountA you will assume to access the repository in the other account, and the name of your IAM user in AccountB. For example:

```
[profile MyCrossAccountAccessProfile]
region = US East (Ohio)
role_arn = arn:aws:iam::111122223333:role/MyCrossAccountRepositoryContributorRole
source_profile = Saanvi_Sarkar
output = json
```

Save your changes, and close the plain-text editor.

7. Run the `git config` command twice: once to configure Git to use the AWS CLI credential helper with the AWS CLI profile you just created, and again to use HTTP. For example:

```
$ git config --global credential.helper '!aws codecommit credential-helper --profile MyCrossAccountAccessProfile $@'
$ git config --global credential.UseHttpPath true
```

8. Run the `git config -l` command to verify your configuration. A successful configuration contains lines similar to the following:

```
user.name=Saanni_Sarkar
user.email=saanvi_sarkar@example.com
credential.helper=!aws --profile MyCrossAccountAccessProfile codecommit credential-helper $@
credential.usehttppath=true
```

**Note**

If you see the following line in your configuration file, another credential manager is configured for your system, which can prevent the AWS CLI credential helper from working:

```
credential.helper=manager
```

To fix this problem, run the following command:

```
$ git config --system --unset credential.helper
```

---

**Step 2: Clone and Access the AWS CodeCommit Repository in AccountA**

Run `git clone`, `git push`, and `git pull` to clone, push to, and pull from, the cross-account AWS CodeCommit repository. You can also sign in to the AWS Management Console, switch roles, and use the AWS CodeCommit console to interact with the repository in the other account.

**Note**

Depending on how the IAM role was configured, you might be able to view repositories on the default page for AWS CodeCommit. If you cannot view the repositories, ask the repository administrator to email you a URL link to the Code page for that repository in the AWS CodeCommit console. The URL is similar to the following:
To clone the cross-account repository to your local computer

1. At the command line or terminal, in the directory where you want to clone the repository, run the `git clone` command with the HTTPS clone URL. For example:

```bash
git clone https://git-codecommit.us-east-2.amazonaws.com/v1/repos/MySharedDemoRepo
```

Unless you specify otherwise, the repository is cloned into a subdirectory with the same name as the repository.

2. Change directories to the cloned repository, and either make a change to a file or add a file. For example, you could add a file named `NewFile.txt`.

3. Add the file to the tracked changes for the local repo, commit the change, and push the file to the AWS CodeCommit repository. For example:

```bash
git add NewFile.txt
git commit -m "Added a file to test cross-account access to this repository"
git push
```

For more information, see [Git with AWS CodeCommit Tutorial](p. 58).

Now that you've added a file, go to the AWS CodeCommit console to view your commit, review other users' changes to the repo, participate in pull requests, and more.

To access the cross-account repository in the AWS CodeCommit console

1. Sign in to the AWS Management Console in AccountB (`888888888888`) as the IAM user who has been granted cross-account access to the repository in AccountA.

2. Choose your user name on the navigation bar, and in the drop-down list, choose **Switch Role**.

   **Note**
   
   If this is the first time you have selected this option, review the information on the page, and then choose **Switch Role** again.

3. On the **Switch Role** page, do the following:

   - In **Account**, type the account ID for AccountA (for example, `111122223333`).
   - In **Role**, type the name of the role you want to assume for access to the repository in AccountA (for example, `MyCrossAccountRepositoryContributorRole`).
   - In **Display Name**, type a friendly name for this role. This name appears in the console when you are assuming this role. It also appears in the list of assumed roles the next time you want to switch roles in the console.
   - (Optional) In **Color**, choose a color label for the display name.
   - Choose **Switch Role**.

   For more information, see [Switching to a Role (AWS Management Console)](p. 58).


   If the assumed role has permission to view the names of repositories in AccountA, you see a list of repositories and an error message that informs you that you do not have permissions to view their status. This is expected behavior. Choose the name of the shared repository from the list.
If the assumed role does not have permission to view the names of repositories in AccountA, you see an error message and a blank list with no repositories. Paste the URL link to the repository or modify the console link and change /list to the name of the shared repository (for example, /MySharedDemoRepo).

5. In Code, find the name of the file you added from your local computer. Choose it to browse the code in the file, and then browse the rest of the repository and start using its features.

For more information, see Getting Started with AWS CodeCommit Tutorial (p. 44).

Delete an AWS CodeCommit Repository

To delete an AWS CodeCommit repository, use the AWS CLI or the AWS CodeCommit console.

Note
Deleting a repository does not delete any local copies of that repository (local repos). To delete a local repo, use your local machine's directory and file management tools.

Topics
• Use the AWS CodeCommit Console to Delete a Repository (p. 128)
• Delete a Local Repo (p. 128)
• Use the AWS CLI to Delete an AWS CodeCommit Repository (p. 129)

Use the AWS CodeCommit Console to Delete a Repository

To use the AWS CodeCommit console to delete an AWS CodeCommit repository, follow these steps.

Important
After you delete an AWS CodeCommit repository, you will no longer be able to clone it to any local repo or shared repo. You will also no longer be able to pull data from it, or push data to it, from any local repo or shared repo. This action cannot be undone.

2. In the list of repositories, choose the name of the repository you want to delete.
3. In the navigation pane, choose Settings.
4. In General, choose Delete repository. In the box next to Type the name of the repository to confirm deletion, type the repository's name, and then choose Delete. The repository is permanently deleted.

Note
Deleting the repository in AWS CodeCommit does not delete any local repos.

Delete a Local Repo

Use your local machine's directory and file management tools to delete the directory that contains the local repo.

Deleting a local repo does not delete any AWS CodeCommit repository to which it might be connected.
Use the AWS CLI to Delete an AWS CodeCommit Repository

To use AWS CLI commands with AWS CodeCommit, install the AWS CLI. For more information, see Command Line Reference (p. 295).

To use the AWS CLI to delete an AWS CodeCommit repository, run the `delete-repository` command, specifying the name of the AWS CodeCommit repository to delete (with the `--repository-name` option).

**Important**
After you delete an AWS CodeCommit repository, you will no longer be able to clone it to any local repo or shared repo. You will also no longer be able to pull data from it, or push data to it, from any local repo or shared repo. This action cannot be undone.

**Tip**
To get the AWS CodeCommit repository's name, run the `list.repositories` (p. 111) command.

For example, to delete a repository named `MyDemoRepo`:

```
aws codecommit delete-repository --repository-name MyDemoRepo
```

If successful, the ID of the AWS CodeCommit repository that was permanently deleted will appear in the output:

```
{
  "repositoryId": "f7579e13-b03e-4027-aaef-650c0EXAMPLE"
}
```

Deleting an AWS CodeCommit repository does not delete any local repos that may be connected to it.
Working with Files in AWS CodeCommit Repositories

In AWS CodeCommit, a file is a version-controlled, self-contained piece of information available to you and other users of the repository and branch where the file is stored. You can organize your repository files with a directory structure, just as you would on a computer. Unlike your computer, AWS CodeCommit automatically tracks every change to a file. You can compare versions of a file, and store different versions of a file in different repository branches.

To add a file to a repository, or edit an existing file in a repository, you can use a Git client. You can also use the AWS CodeCommit console, the AWS CLI, or the AWS CodeCommit API.

For information about working with other aspects of your repository in AWS CodeCommit, see Working with Repositories (p. 75), Working with Pull Requests (p. 138), Working with Branches (p. 194), Working with Commits (p. 163), and Working with User Preferences (p. 208).

Topics

- Browse Files in an AWS CodeCommit Repository (p. 131)
- Create or Add a File to an AWS CodeCommit Repository (p. 132)
- Edit the Contents of a File in an AWS CodeCommit Repository (p. 135)
Browse Files in an AWS CodeCommit Repository

After you connect to an AWS CodeCommit repository, you can clone it to a local repo or use the AWS CodeCommit console to browse its contents. This topic provides instructions for browsing the content of an AWS CodeCommit repository by using the console.

**Note**
For active AWS CodeCommit users, there is no charge for browsing code from the AWS CodeCommit console. For information about when charges may apply, see [Pricing](https://aws.amazon.com/codecommit/pricing/).

Browse an AWS CodeCommit Repository

You can use the AWS CodeCommit console to review the files contained in a repository or to quickly read the contents of a file.

**To browse the content of a repository**

2. On the **Dashboard** page, from the list of repositories, choose the repository you want to browse.
3. In the **Code** view, browse the contents of the default branch for your repository.

   To change the view to a different branch or tag, choose the view selector button. Either choose a branch or tag name from the drop-down list, or in the filter box, type the name of the branch or tag, and then choose it from the list.

4. Do one of the following:
   - To view the contents of a directory, choose it from the list. You can choose any of the directories in the navigation list to return to that directory view. You can also use the up arrow at the top of the directory list.
Create or Add a File

To view the contents of a file, choose it from the list. If the file is larger than the commit object limit, it cannot be displayed in the console and must be viewed in a local repo instead. For more information, see Limits (p. 282). To exit the file view, from the code navigation bar, choose the directory you want to view.

**Note**

If you choose a binary file, a warning message appears, asking you to confirm that you want to display the contents. To view the file, choose *Show file contents*. If you do not want to view the file, from the code navigation bar, choose the directory you want to view.

If you choose a markdown file (.md), use the *Rendered Markdown* and *Markdown Source* buttons to toggle between the rendered and syntax views.

Create or Add a File to an AWS CodeCommit Repository

You can use the AWS CodeCommit console, AWS CLI, or a Git client to add a file to a repository. You can upload a file from your local computer to the repository, or you can use the code editor in the console to create the file. The editor is a quick and easy way to add a simple file, such as a readme.md file, to a branch in a repository.
Create or Upload a File in the AWS CodeCommit Console

You can use the AWS CodeCommit console to create a file and add it to a branch in an AWS CodeCommit repository. As part of creating the file, you can provide your user name and an email address. You can also add a commit message so other users understand who added the file and why. You can also upload a file directly from your local computer to a branch in a repository.

To add a file to a repository (console)

2. On the Dashboard page, from the list of repositories, choose the repository where you want to add a file.
3. In the Code view, choose the branch where you want to add the file. By default, the contents of the default branch are shown when you open the Code view.
   
   To change the view to a different branch, choose the view selector button. Either choose a branch name from the drop-down list, or in the filter box, type the name of the branch, and then choose it from the list.
4. Choose Add file, and then choose one of the following options:
   - Create file, to use the code editor to create the contents of a file and add it to the repository.
   - Upload file, to upload a file from your local computer to the repository.
5. Provide information to other users about who added this file to the repository and why.
   - In Author name, type your name. This name is used as both the author name and the committer name in the commit information. AWS CodeCommit defaults to using your IAM user name or a derivation of your console login as the author name.
   - In Email address, type an email address so that other repository users can contact you about this change.
   - In Commit message, type a brief description.
6. Do one of the following:
   - If you are uploading a file, choose the file from your local computer.
   - If you are creating a file, type the content you want to add in the code editor, and provide a name for the file.

Add a File Using the AWS CLI

You can use the AWS CLI and the put-file command to add a file in an AWS CodeCommit repository. You can also use the put-file command to add a directory or path structure for the file.

Note

To use AWS CLI commands with AWS CodeCommit, install the AWS CLI. For more information, see Command Line Reference (p. 295).
To add a file to a repository (AWS CLI)

1. On your local computer, create the file you want to add to the AWS CodeCommit repository.
2. At the terminal or command line, run the `put-file` command, specifying:
   - The repository where you want to add the file.
   - The branch where you want to add the file.
   - The full commit ID of the most recent commit made to that branch, also known as the tip or head commit.
   - The local location of the file. The syntax used for this location varies depending on your local operating system.
   - The name of the file you want to add, including the path where the updated file is stored in the repository, if any.
   - The user name and email you want associated with this file.
   - A commit message that explains why you added this file.

The user name, email address, and commit message are optional, but help other users know who made the change and why. If you do not supply a user name, AWS CodeCommit defaults to using your IAM user name or a derivation of your console login as the author name.

For example, to add a file named `ExampleSolution.py` to a repository named `MyDemoRepo` to a branch named `feature-randomizationfeature` whose most recent commit has an ID of `4c925148EXAMPLE`:

```shell
aws codecommit put-file --repository-name MyDemoRepo --branch-name feature-randomizationfeature --file-content file://MyDirectory/ExampleSolution.py --file-path /solutions/ExampleSolution.py --parent-commit-id 4c925148EXAMPLE --name "María García" --email "maría_garcía@example.com" --commit-message "I added a third randomization routine."
```

**Note**
When you add binary files, make sure that you use `fileb://` to specify the local location of the file.

If successful, this command returns output similar to the following:

```json
{
  "blobId": "2eb4af3bEXAMPLE",
  "commitId": "317f8570EXAMPLE",
  "treeId": "347a3408EXAMPLE"
}
```

**Add a File Using Git**

You can add files in a local repo and push your changes to an AWS CodeCommit repository. For more information, see [Git with AWS CodeCommit Tutorial](p. 58).
Edit the Contents of a File in an AWS CodeCommit Repository

You can use the AWS CodeCommit console, AWS CLI, or a Git client to edit the contents of a file in an AWS CodeCommit repository.

Topics

• Edit a File in the AWS CodeCommit Console (p. 135)
• Edit a File Using the AWS CLI (p. 136)
• Edit a File Using Git (p. 134)

Edit a File in the AWS CodeCommit Console

You can use the AWS CodeCommit console to edit a file that has been added to a branch in an AWS CodeCommit repository. As part of editing the file, you can provide your user name and an email address. You can also add a commit message so other users understand who made the change and why.
To edit a file in a repository (console)

2. On the Dashboard page, from the list of repositories, choose the repository you want to browse.
3. In the Code view, choose the branch where you want to edit the file. By default, the contents of the default branch are shown when you open the Code view.

To change the view to a different branch, choose the view selector button. Either choose a branch name from the drop-down list, or in the filter box, type the name of the branch, and then choose it from the list.
4. Navigate the contents of the branch and choose the file you want to edit. In the file view, choose Edit.

   Note
   If you choose a binary file, a warning message appears asking you to confirm that you want to display the contents. You should not use the AWS CodeCommit console to edit binary files.

5. Edit the file, and provide information to other users about who made this change and why.
   • In Author name, type your name. This name is used as both the author name and the committer name in the commit information. AWS CodeCommit defaults to using your IAM user name or a derivation of your console login as the author name.
   • In Email address, type an email address so that other repository users can contact you about this change.
   • In Commit message, type a brief description of your changes.
6. Choose Commit changes to save your changes to the file and commit the changes to the repository.

Edit a File Using the AWS CLI

You can use the AWS CLI and the put-file command to make changes to a file in an AWS CodeCommit repository. You can also use the put-file command to add a directory or path structure for the changed file, if you want to store the changed file in a location different from the original.

   Note
   To use AWS CLI commands with AWS CodeCommit, install the AWS CLI. For more information, see Command Line Reference (p. 295).

To edit a file in a repository (AWS CLI)

1. Using a local copy of the file, make the changes you want to add to the AWS CodeCommit repository.
2. At the terminal or command line, run the put-file command, specifying:
   • The repository where you want to add the edited file.
   • The branch where you want to add the edited file.
   • The full commit ID of the most recent commit made to that branch, also known as the tip or head commit.
   • The local location of the file.
   • The name of the updated file you want to add, including the path where the updated file is stored in the repository, if any.
   • The user name and email you want associated with this file change.
   • A commit message that explains the change you made.
The user name, email address, and commit message are optional, but help other users know who made the change and why. If you do not supply a user name, AWS CodeCommit defaults to using your IAM user name or a derivation of your console login as the author name.

For example, to add edits made to a file named ExampleSolution.py to a repository named MyDemoRepo to a branch named feature-randomizationfeature whose most recent commit has an ID of 4c925148EXAMPLE:

```
aws codecommit put-file --repository-name MyDemoRepo --branch-name feature-randomizationfeature --file-content file://MyDirectory/ExampleSolution.py --file-path /solutions/ExampleSolution.py --parent-commit-id 4c925148EXAMPLE --name "María García" --email "maria_garcia@example.com" --commit-message "I fixed the bug Mary found."
```

**Note**
If you want to add a changed binary file, make sure to use --file-content with the notation `fileb://MyDirectory/MyFile.raw`.

If successful, this command returns output similar to the following:

```
{
  "blobId": "2eb4af3bEXAMPLE",
  "commitId": "317f8570EXAMPLE",
  "treeId": "347a3408EXAMPLE"
}
```

---

## Edit a File Using Git

You can edit files in a local repo and push your changes to an AWS CodeCommit repository. For more information, see [Git with AWS CodeCommit Tutorial (p. 58)](#).
Working with Pull Requests in AWS CodeCommit Repositories

A pull request is the primary way you and other repository users can review, comment on, and merge code changes from one branch to another. You can use pull requests to collaboratively review code changes for minor changes or fixes, major feature additions, or new versions of your released software. Here is one possible workflow for a pull request:

Li Juan, a developer working in a repo named MyDemoRepo, wants to work on a new feature for an upcoming version of a product. To keep her work separate from production-ready code, she creates a branch off of the default branch and names it `feature-randomizationfeature`. She writes code, makes commits, and pushes the new feature code into this branch. She wants other repository users to review the code for quality before she merges her changes into the default branch. To do this, she creates a pull request. The pull request contains the comparison between her working branch and the branch of the code where she intends to merge her changes (in this case, the default branch). Other users review her code and changes, adding comments and suggestions. She might update her working branch multiple times with code changes in response to comments. Her changes are incorporated into the pull request every time she pushes them to that branch in AWS CodeCommit. She might also incorporate changes that have been made in the intended destination branch while the pull request is open, so users can be sure they're reviewing all the proposed changes in context. When she and her reviewers are satisfied, she merges her code and closes the pull request.
Pull requests require two branches: a source branch that contains the code you want reviewed, and a destination branch, where you merge the reviewed code. The source branch contains the AFTER commit, which is the commit that contains the changes you want to merge into the destination branch. The destination branch contains the BEFORE commit, which represents the “before” state of the code (before the pull request branch is merged into the destination branch).

The pull request displays the differences between these two branches, so users can view and comment on the changes. You can update the pull request in response to comments by committing and pushing changes to the source branch.
When your code has been reviewed, you can close the pull request in one of several ways:

- Merge the branches locally and push your changes. This closes the request automatically.
- Use the AWS CodeCommit console to either close the pull request without merging or, if there are no conflicts, to close and merge the branches.
- Use the AWS CLI.

Before you create a pull request:

- Create a branch that contains the code you want reviewed (the source branch).
- Commit and push the code you want reviewed to the source branch.
- Set up notifications for your repository, so other users can be notified about the pull request and changes to it. (This step is optional, but recommended.)

Pull requests are more effective when you’ve set up IAM users for your repository users in your AWS account. It’s easier to distinguish which user made which comment. IAM users also have the advantage
of being able to use Git credentials for repository access. For more information, see Step 1: Initial Configuration for AWS CodeCommit (p. 6). However, you can use pull requests with other kinds of users, including federated access users.

For information about working with other aspects of your repository in AWS CodeCommit, see Working with Repositories (p. 75), Working with Files (p. 130), Working with Commits (p. 163), Working with Branches (p. 194), and Working with User Preferences (p. 208).

Topics
- Create a Pull Request (p. 141)
- View Pull Requests in an AWS CodeCommit Repository (p. 144)
- Review a Pull Request (p. 148)
- Update a Pull Request (p. 155)
- Close a Pull Request in an AWS CodeCommit Repository (p. 159)

Create a Pull Request

Creating pull requests helps other users see and review your code changes before you merge them into another branch. First, you create a branch for your code changes. This is referred to as the source branch for a pull request. After you commit and push changes to the repository, you can create a pull request that compares the contents of that branch (the source branch) to the branch where you want to merge your changes after the pull request is closed (the destination branch).

You can use the AWS CodeCommit console or the AWS CLI to create pull requests for your repository.

Topics
- Use the AWS CodeCommit Console to Create a Pull Request (p. 141)
- Use the AWS CLI to Create a Pull Request (p. 143)

Use the AWS CodeCommit Console to Create a Pull Request

You can use the AWS CodeCommit console to create a pull request in an AWS CodeCommit repository. If your repository is configured with notifications (p. 84), subscribed users receive an email when you create a pull request.

2. In the list of repositories, choose the name of the repository.
3. In the navigation pane, choose Pull Requests.

   Tip
   You can also create pull requests from Branches, Code, or Compare.

4. Choose Create pull request.
5. In **Create pull request**, in **Source**, choose the branch that contains the changes you want reviewed.

6. In **Destination**, review the branch where you intend to merge your code changes when the pull request is closed. By default, the destination branch is preconfigured with the default branch of the repository, but you can choose a different branch.

7. Choose **Compare**. A comparison runs on the two branches, and the differences between them are displayed. An analysis is also performed to determine whether the two branches can be merged automatically when the pull request is closed.

8. Review the comparison details and the changes to be sure that the pull request contains the changes and commits you want reviewed. If not, adjust your choices for source and destination branches, and choose **Compare** again.

9. When you are satisfied with the comparison results for the pull request, in **Title**, provide a short but descriptive title for this review. This is the title that appears in the list of pull requests for the repository.

10. (Optional) In **Description**, provide details about this review and any other useful information for reviewers.

11. Choose **Create**.
Your pull request appears in the list of pull requests for the repository. If you configured notifications (p. 84), subscribers to the Amazon SNS topic receive an email to inform them of the newly created pull request.

Use the AWS CLI to Create a Pull Request

To use AWS CLI commands with AWS CodeCommit, install the AWS CLI. For more information, see Command Line Reference (p. 295).

To use the AWS CLI to create a pull request in an AWS CodeCommit repository:

1. Run the `create-pull-request` command, specifying:
   - The name of the pull request (with the `--title` option).
   - The description of the pull request (with the `--description` option).
   - A list of targets for the `create-pull-request` command, including:
     - The name of the AWS CodeCommit repository where the pull request is created (with the `repositoryName` attribute).
     - The name of the branch that contains the code changes you want reviewed, also known as the source branch (with the `sourceReference` attribute).
• (Optional) The name of the branch where you intend to merge your code changes, also known as the destination branch, if you do not want to merge to the default branch (with the destinationReference attribute).

• A unique, client-generated idempotency token (with the --client-request-token option).

For example, to create a pull request named My Pull Request with a description of Please review these changes by Tuesday that targets the MyNewBranch source branch and is to be merged to the default branch master in an AWS CodeCommit repository named MyDemoRepo:

```
aws codecommit create-pull-request --title "My Pull Request" --description "Please review these changes by Tuesday" --client-request-token 123Example --targets repositoryName=MyDemoRepo,sourceReference=MyNewBranch
```

2. If successful, this command produces output similar to the following:

```
{
    "pullRequest": {
        "authorArn": "arn:aws:iam::111111111111:user/Jane_Doe",
        "clientRequestToken": "123Example",
        "creationDate": 1508962823.285,
        "description": "Please review these changes by Tuesday",
        "lastActivityDate": 1508962823.285,
        "pullRequestId": "42",
        "pullRequestStatus": "OPEN",
        "pullRequestTargets": [
            {
                "destinationCommit": "5d036259EXAMPLE",
                "destinationReference": "refs/heads/master",
                "mergeMetadata": {
                    "isMerged": false,
                },
                "repositoryName": "MyDemoRepo",
                "sourceCommit": "317f8570EXAMPLE",
                "sourceReference": "refs/heads/MyNewBranch"
            }
        ],
        "title": "My Pull Request"
    }
}
```

View Pull Requests in an AWS CodeCommit Repository

You can use the AWS CodeCommit console or the AWS CLI to view pull requests for your repository. By default, you see only open pull requests, but you can change the filter to view all pull requests, only closed requests, only pull requests that you created, and more.

Topics
• Use the AWS CodeCommit Console to View Pull Requests (p. 145)
• Use the AWS CLI to View Pull Requests (p. 145)
Use the AWS CodeCommit Console to View Pull Requests

You can use the AWS CodeCommit console to view a list of pull requests in an AWS CodeCommit repository. By changing the filter, you can change the list display to only show you a certain set of pull requests. For example, you can view a list of pull requests you created with a status of Open, or you can choose a different filter and view pull requests you created with a status of Closed.

2. In the list of repositories, choose the name of the repository.
3. In the navigation pane, choose Pull Requests.
4. By default, a list of all open pull requests is displayed. Pull requests are displayed in the order of most recent activity.

5. To display other pull requests, in View, choose the display filter that meets your needs:
   - **All open requests** (default): Displays all pull requests with a status of Open.
   - **All requests**: Displays all pull requests.
   - **All closed requests**: Displays all pull requests with a status of Closed.
   - **All my requests**: Displays all pull requests that you created, regardless of the status. It does not display reviews that you have commented on or otherwise participated in.
   - **All my open requests**: Displays all pull requests that you created with a status of Open.
   - **All my closed requests**: Displays all pull requests that you created with a status of Closed.
6. When you find a pull request in the displayed list that you would like to view, choose it.

Use the AWS CLI to View Pull Requests

To use AWS CLI commands with AWS CodeCommit, install the AWS CLI. For more information, see Command Line Reference (p. 295).
To use the AWS CLI to view pull requests in an AWS CodeCommit repository, follow these steps.

1. To view a list of pull requests in a repository, run the `list-pull-requests` command, specifying:
   - The name of the AWS CodeCommit repository where you want to view pull requests (with the `--repository-name` option).
   - (Optional) The status of the pull request (with the `--pull-request-status` option).
   - (Optional) The Amazon Resource Name (ARN) of the IAM user who created the pull request (with the `--author-arn` option).
   - (Optional) An enumeration token that can be used to return batches of results (with the `--next-token` option)
   - (Optional) A limit on the number of returned results per request (with the `--max-results` option).

   For example, to list pull requests created by an IAM user with the ARN `arn:aws:iam::111111111111:user/Li_Juan` and the status of `CLOSED` in an AWS CodeCommit repository named `MyDemoRepo`:

   ```bash
   aws codecommit list-pull-requests --author-arn arn:aws:iam::111111111111:user/Li_Juan
   --pull-request-status CLOSED --repository-name MyDemoRepo
   
   If successful, this command produces output similar to the following:
   
   ```json
   {
     "nextToken": "",
     "pullRequestIds": ["2","12","16","22","23","35","30","39","47"]
   }
   ```

   Pull request IDs are displayed in the order of most recent activity.

2. To view details of a pull request, run the `get-pull-request` command with the `--pull-request-id` option, specifying the ID of the pull request. For example, to view information about a pull request with the ID of `42`:

   ```bash
   aws codecommit get-pull-request --pull-request-id 42
   
   If successful, this command produces output similar to the following:
   
   ```json
   {
     "pullRequest": {
       "authorArn": "arn:aws:iam::111111111111:user/Jane_Doe",
       "title": "Pronunciation difficulty analyzer",
       "pullRequestTargets": [
         {
           "destinationReference": "refs/heads/master",
           "destinationCommit": "5d036259EXAMPLE",
           "sourceReference": "refs/heads/jane-branch",
           "sourceCommit": "317f8570EXAMPLE",
           "repositoryName": "MyDemoRepo",
           "mergeMetadata": {
             "isMerged": false,
           },
         },
       ],
       "lastActivityDate": 1508442444,
       "pullRequestId": "42",
       "clientRequestToken": "123Example",
       "pullRequestStatus": "OPEN",
     },
   }
   ```
To view events in a pull request, run the `describe-pull-request-events` command with the `--pull-request-id` option, specifying the ID of the pull request. For example, to view the events for a pull request with the ID of 8:

```
aws codecommit describe-pull-request-events --pull-request-id 8
```

If successful, this command produces output similar to the following:

```
{
  "pullRequestEvents": [
    {
      "pullRequestId": "8",
      "pullRequestEventType": "PULL_REQUEST_CREATED",
      "eventDate": 1510341779.53,
      "actor": "arn:aws:iam::111111111111:user/Zhang_Wei"
    },
    {
      "pullRequestId": "8",
      "pullRequestStatusChangedEventMetadata": {
        "pullRequestStatus": "CLOSED"
      },
      "pullRequestId": "8",
      "pullRequestEventType": "PULL_REQUEST_STATUS_CHANGED",
      "eventDate": 1510341930.72,
      "actor": "arn:aws:iam::111111111111:user/Jane_Doe"
    }
  ]
}
```

To view whether there are any merge conflicts for a pull request, run the `get-merge-conflicts` command, specifying:

- The name of the AWS CodeCommit repository (with the `--repository-name` option).
- The branch, tag, HEAD, or other fully qualified reference for the source of the changes to use in the merge evaluation (with the `--source-commit-specifier` option).
- The branch, tag, HEAD, or other fully qualified reference for the destination of the changes to use in the merge evaluation (with the `--destination-commit-specifier` option).
- The merge option to use (with the `--merge-option` option)

For example, to view whether there are any merge conflicts between the tip of a source branch named `my-feature-branch` and a destination branch named `master` in a repository named `MyDemoRepo`:

```
aws codecommit get-merge-conflicts --repository-name MyDemoRepo --source-commit-specifier my-feature-branch --destination-commit-specifier master --merge-option FAST_FORWARD_MERGE
```

If successful, this command returns output similar to the following:

```
{
  "destinationCommitId": "fac04518EXAMPLE",
  "mergeable": false,
  "sourceCommitId": "16d097f03EXAMPLE"
}
```
Review a Pull Request

You can use the AWS CodeCommit console to review the changes included in a pull request. You can add comments to the request, to the files, and to specific lines of code. You can also reply to comments made by other users. If your repository is configured with notifications (p. 84), you receive emails when users reply to your comments or when users comment on a pull request.

You can use the AWS CLI to comment on a pull request and reply to comments. To review the changes, you must use the `git diff` command or a diff tool.

Topics
- Use the AWS CodeCommit Console to Review a Pull Request (p. 148)
- Use the AWS CLI to Review Pull Requests (p. 152)

Use the AWS CodeCommit Console to Review a Pull Request

You can use the AWS CodeCommit console to review a pull request in an AWS CodeCommit repository.

2. In the list of repositories, choose the name of the repository.
3. In the navigation pane, choose Pull Requests.
4. By default, a list of all open pull requests is displayed. Choose the open pull request you want to review. You can also comment on a closed pull request.
5. In the pull request, choose Changes.
6. Do one of the following:
To add a general comment, in **Comments on changes**, type a comment and then choose **Save**. You can use **Markdown**, or you can type your comment in plaintext.

![Comments on changes](image)

To add a comment to a file in the commit, in **Changes**, find the name of the file. Choose the comment bubble that appears next to the file name, type a comment, and then choose **Save**.
To add a comment to a changed line in the pull request, in **Changes**, go to the line you want to comment on. Choose the comment bubble , type a comment, and then choose **Save**.
7. To reply to comments on a commit, in Changes or Activity, choose Reply.
If notifications (p. 84) are configured, the user who created the pull request receives email about your comments. You receive email if a user replies to your comments or if the pull request is updated.

**Use the AWS CLI to Review Pull Requests**

To use AWS CLI commands with AWS CodeCommit, install the AWS CLI. For more information, see Command Line Reference (p. 295).

To use the AWS CLI to review pull requests in an AWS CodeCommit repository:

1. To add a comment to a pull request in a repository, run the `post-comment-for-pull-request` command, specifying:
   - The ID of the pull request (with the `--pull-request-id` option).
   - The name of the repository that contains the pull request (with the `--repository-name` option).
Use the AWS CLI to Review Pull Requests

- The full commit ID of the commit in the destination branch where the pull request will be merged (with the \---before-commit-id option).
- The full commit ID of the commit in the source branch that is the current tip of the branch for the pull request when you post the comment (with the \---after-commit-id option).
- A unique, client-generated idempotency token (with the \---client-request-token option).
- The content of your comment (with the \---content option).
- A list of location information about where to place the comment, including:
  - The name of the file being compared, including its extension and subdirectory, if any (with the \texttt{filePath} attribute).
  - The line number of the change within a compared file (with the \texttt{filePosition} attribute).
  - Whether the comment on the change is "before" or "after" in the comparison between the source and destination branches (with the \texttt{relativeFileVersion} attribute).

For example, to add the comment "These don’t appear to be used anywhere. Can we remove them?" on the change to the \texttt{ahs_count.py} file in a pull request with the ID of 47 in a repository named \texttt{MyDemoRepo}:

```bash
aws codecommit post-comment-for-pull-request --pull-request-id "47" --repository-name MyDemoRepo --before-commit-id 317f8570EXAMPLE --after-commit-id 5d036259EXAMPLE --client-request-token 123Example --content "These don’t appear to be used anywhere. Can we remove them?" --location filePath=ahs_count.py,filePosition=367,relativeFileVersion=AFTER
```

If successful, this command produces output similar to the following:

```json
{
    "afterBlobId": "1f330709EXAMPLE",
    "afterCommitId": "5d036259EXAMPLE",
    "beforeBlobId": "8090646cEXAMPLE",
    "beforeCommitId": "317f8570EXAMPLE",
    "comment": {
        "authorArn": "arn:aws:iam::111111111111:user/Saanvi_Sarkar",
        "clientRequestToken": "123Example",
        "commentId": "abcd1234EXAMPLEb5678efgh",
        "content": "These don’t appear to be used anywhere. Can we remove them?",
        "creationDate": 1508369622.123,
        "deleted": false,
        "lastModifiedDate": 1508369622.123
    },
    "location": {
        "filePath": "ahs_count.py",
        "filePosition": 367,
        "relativeFileVersion": "AFTER"
    },
    "repositoryName": "MyDemoRepo",
    "pullRequestId": "47"
}
```

2. To view comments for a pull request, run the \texttt{get-comments-for-pull-request} command, specifying:

- The AWS CodeCommit repository's name (with the \---repository-name option).
- The full commit ID of the commit in the source branch that was the tip of the branch at the time the comment was made (with the \---after-commit-id option).
- The full commit ID of the commit in the destination branch that was the tip of the branch at the time the pull request was created (with the \---before-commit-id option).
• (Optional) An enumeration token to return the next batch of the results (with the --next-token option).
• (Optional) A non-negative integer to limit the number of returned results (with the --max-results option).

For example, to view comments for a pull request in a repository named *MyDemoRepo*:

```
aws codecommit get-comments-for-pull-request --repository-name MyDemoRepo --before-commit-ID 317f8570EXAMPLE --after-commit-id 5d036259EXAMPLE
```

If successful, this command produces output similar to the following:

```
{
  "commentsForPullRequestData": [
    {
      "afterBlobId": "1f330709EXAMPLE",
      "afterCommitId": "5d036259EXAMPLE",
      "beforeBlobId": "80906a4cEXAMPLE",
      "beforeCommitId": "317f8570EXAMPLE",
      "comments": [
        {
          "authorArn": "arn:aws:iam::111111111111:user/Saanvi_Sarkar",
          "clientRequestToken": "",
          "commentId": "abcd1234EXAMPLEb5678efgh",
          "content": "These don't appear to be used anywhere. Can we remove them?",
          "creationDate": 1508369622.123,
          "deleted": false,
          "lastModifiedDate": 1508369622.123
        },
        {
          "authorArn": "arn:aws:iam::111111111111:user/Li_Juan",
          "clientRequestToken": "",
          "commentId": "442b498bEXAMPLE5756813",
          "content": "Good catch. I'll remove them."
          "creationDate": 1508369829.104,
          "deleted": false,
          "lastModifiedDate": 150836912.273
        }
      ],
      "location": {
        "filePath": "ahs_count.py",
        "filePosition": 367,
        "relativeFileVersion": "AFTER"
      },
      "repositoryName": "MyDemoRepo",
      "pullRequestId": "42"
    }
  ],
  "nextToken": "exampleToken"
}
```

3. To post a reply to a comment in a pull request, run the `post-comment-reply` command, specifying:

- The system-generated ID of the comment to which you want to reply (with the --in-reply-to option).
- A unique, client-generated idempotency token (with the --client-request-token option).
- The content of your reply (with the --content option).
For example, to add the reply "Good catch. I’ll remove them." to the comment with the system-generated ID of `abcd1234EXAMPLEb5678efgh`:

/aws codecommit post-comment-reply --in-reply-to abcd1234EXAMPLEb5678efgh --content "Good catch. I’ll remove them." --client-request-token 123Example

If successful, this command produces output similar to the following:

```
{
   "comment": {
      "authorArn": "arn:aws:iam::111111111111:user/Li_Juan",
      "clientRequestToken": "123Example",
      "commentId": "442b498bEXAMPLE5756813",
      "content": "Good catch. I’ll remove them.",
      "creationDate": 1508369829.136,
      "deleted": false,
      "CommentId": "abcd1234EXAMPLEb5678efgh",
      "lastModifiedDate": 150836912.221
   }
}
```

Update a Pull Request

You can use your local Git client to push commits to the source branch, which updates the pull request with code changes. You might update the pull request with more commits because:

- You want users to review code changes you made to the source branch code in response to comments in the pull request.
- One or more commits have been made to the destination branch since the pull request was created. You want to incorporate those changes into the source branch as part of the review (forward integration). This changes the state of the pull request to **Mergeable** and enables the merging and closing of the pull request from the console.

You can use the AWS CodeCommit console or the AWS CLI to update the title or description of a pull request. You might want to update the pull request because:

- Other users don't understand the description, or the original title is misleading.
- You want the title or description to reflect changes made to the source branch of an open pull request.

**Topics**

- Use Git to Update a Pull Request (p. 155)
- Use the AWS CodeCommit Console to Update a Pull Request (p. 156)
- Use the AWS CLI to Update Pull Requests (p. 157)

**Use Git to Update a Pull Request**

You can use Git to update the source branch of a pull request with changes to the code to:

- Add more code to the review.
- Incorporate changes suggested in review comments.
Use the AWS CodeCommit Console to Update a Pull Request

You can use the AWS CodeCommit console to update the title and description of a pull request in an AWS CodeCommit repository.

2. In the list of repositories, choose the name of the repository.
3. In the navigation pane, choose Pull Requests.
4. By default, a list of all open pull requests is displayed. Choose the open pull request you want to update.
5. In the pull request, choose the option to edit the title or description.

### Use the AWS CLI to Update Pull Requests

To use AWS CLI commands with AWS CodeCommit, install the AWS CLI. For more information, see [Command Line Reference](p. 295).

To use the AWS CLI to update pull requests in an AWS CodeCommit repository:

1. To update the title of a pull request in a repository, run the `update-pull-request-title` command, specifying:
   - The ID of the pull request (with the `--pull-request-id` option).
   - The title of the pull request (with the `--title` option).

   For example, to update the title of a pull request with the ID of **47**:

   ```bash
   aws codecommit update-pull-request-title --pull-request-id 47 --title "Consolidation of global variables - updated review"
   ```

   If successful, this command produces output similar to the following:

   ```json
   { 
     "pullRequest": { 
       "authorArn": "arn:aws:iam::111111111111:user/Li_Juan",
       "clientRequestToken": "",
       "creationDate": 1508530823.12,
       "description": "Review the latest changes and updates to the global variables. I have updated this request with some changes, including removing some unused variables.",
       "lastActivityDate": 1508372657.188,
       "pullRequestId": "47",
   }
   ```
2. To update the description of a pull request, run the `update-pull-request-description` command, specifying:

- The ID of the pull request (with the `--pull-request-id` option).
- The description (with the `--description` option).

For example, to update the description of a pull request with the ID of 47:

```bash
codecommit update-pull-request-description --pull-request-id 47 --description "Updated the pull request to remove unused global variable."
```

If successful, this command produces output similar to the following:

```json
{
  "pullRequest": {
    "authorArn": "arn:aws:iam::111111111111:user/Li_Juan",
    "clientRequestToken": "",
    "creationDate": 1508530823.155,
    "description": "Updated the pull request to remove unused global variable.",
    "lastActivityDate": 1508372423.204,
    "pullRequestId": "47",
    "pullRequestTargets": {
      "destinationCommit": "9f31c968EXAMPLE",
      "destinationReference": "refs/heads/master",
      "mergeMetadata": {
        "isMerged": false,
      },
      "repositoryName": "MyDemoRepo",
      "sourceCommit": "99132ab0EXAMPLE",
      "sourceReference": "refs/heads/variables-branch"
    }
  },
  "title": "Consolidation of global variables"
}
```
Close a Pull Request in an AWS CodeCommit Repository

When you're satisfied that your code has been reviewed, you can close a pull request in one of several ways:

• On your local computer, you can use the `git merge` command to merge the source branch into the destination branch, and then push your merged code to the destination branch. This closes the pull request automatically. The `git merge` command also allows you to choose the merge option or strategy you use for the merge. This is the only option available to merge the branches if the pull request status shows Resolve conflicts. To learn more about `git merge` and merge options, see `git-merge` or your Git documentation.

• In the console, you can close a pull request without merging the code. You might want to do this if you want to use the `git merge` command to merge the branches manually, or if the code in the pull request source branch isn't code you want merged into the destination branch. This is the only option available if the code cannot be merged automatically. You see an advisory message to resolve conflicts, which you must do on your local computer with the `git merge` command or a diff or merge tool.

• In the console, you might be able to merge your source branch to the destination branch automatically, which closes the pull request automatically. You see an advisory message that the pull request is mergeable, and the Merge button in the pull request is active. When you choose Merge, the merge is performed using the fast-forward merge option.

  **Note**
  The fast-forward option does not create a commit or commit message for the merge. If you want a merge commit to appear in the history of the destination branch, you can choose not to automatically merge the code as part of closing the pull request. Instead, you can manually merge the branches using the `git merge` command with a different merge option.

• AWS CodeCommit closes a pull request automatically if either the source or destination branch of the pull request is deleted.

• In the AWS CLI, you can update the status of a pull request from OPEN to CLOSED. This closes the pull request. You can also use the AWS CLI to attempt to merge and close the pull request.

**Topics**
- Use the AWS CodeCommit Console to Close a Pull Request (p. 159)
- Use the AWS CLI to Close Pull Requests (p. 160)

**Use the AWS CodeCommit Console to Close a Pull Request**

You can use the AWS CodeCommit console to close a pull request in an AWS CodeCommit repository. After the status of a pull request is changed to Closed, it cannot be changed back to Open, but users can still comment on the changes and reply to comments.

2. In the list of repositories, choose the name of the repository.
3. In the navigation pane, choose Pull Requests.
4. By default, a list of all open pull requests is displayed. Choose the open pull request you would like to close.
5. In the pull request, choose one of the following:

- **Merge**: This option, if available, closes the pull request and attempts to merge the code into the destination branch using the fast-forward merge option. You can also optionally select the option to automatically delete the pull request source branch after the merge is successful. If the merge attempt is not successful, the source branch is not deleted.

  **Note**
  The Merge option is available only if there are no merge conflicts detected between the source and destination branches.

- **Close pull request**: This option closes the pull request without attempting to merge the source branch into the destination branch. This option does not provide a way to delete the source branch as part of closing the pull request, but you can do it yourself after the request is closed.

---

**Use the AWS CLI to Close Pull Requests**

To use AWS CLI commands with AWS CodeCommit, install the AWS CLI. For more information, see **Command Line Reference (p. 295)**.

To use the AWS CLI to close pull requests in an AWS CodeCommit repository:

1. To update the status of a pull request in a repository from **OPEN** to **CLOSED**, run the `update-pull-request-status` command, specifying:

   - The ID of the pull request (with the `--pull-request-id` option).
   - The status of the pull request (with the `--pull-request-status` option).

   For example, to update the status of a pull request with the ID of **42** to a status of **CLOSED** in an AWS CodeCommit repository named **MyDemoRepo**:
aws codecommit update-pull-request-status --pull-request-id 42 --pull-request-status CLOSED

If successful, this command produces output similar to the following:

```
{
  "pullRequest": {
    "authorArn": "arn:aws:iam::111111111111:user/Jane_Doe",
    "clientRequestToken": "123Example",
    "creationDate": 1508962823.165,
    "description": "A code review of the new feature I just added to the service.",
    "lastActivityDate": 1508442444.12,
    "pullRequestId": "42",
    "pullRequestStatus": "CLOSED",
    "pullRequestTargets": [
      {
        "destinationCommit": "5d036259EXAMPLE",
        "destinationReference": "refs/heads/master",
        "mergeMetadata": { "isMerged": false },
        "repositoryName": "MyDemoRepo",
        "sourceCommit": "317f8570EXAMPLE",
        "sourceReference": "refs/heads/jane-branch"
      }
    ],
    "title": "Pronunciation difficulty analyzer"
  }
}
```

2. To merge and close a pull request, run the `merge-pull-request-by-fast-forward` command, specifying:

- The ID of the pull request (with the `--pull-request-id` option).
- The full commit ID of the tip of the source branch (with the `--source-commit-id` option).
- The name of the repository (with the `--repository-name` option).

For example, to merge and close a pull request with the ID of 47 and a source commit ID of 99132ab0EXAMPLE in a repository named MyDemoRepo:

```
aws codecommit merge-pull-request-by-fast-forward --pull-request-id 47 --source-commit-id 99132ab0EXAMPLE --repository-name MyDemoRepo
```

If successful, this command produces output similar to the following:

```
{
  "pullRequest": {
    "authorArn": "arn:aws:iam::111111111111:user/Li_Juan",
    "clientRequestToken": "",
    "creationDate": 1508500823.142,
    "description": "Review the latest changes and updates to the global variables",
    "lastActivityDate": 1508887223.155,
    "pullRequestId": "47",
    "pullRequestStatus": "CLOSED",
    "pullRequestTargets": [
      {
        "destinationCommit": "9f31c968EXAMPLE",
        "destinationReference": "refs/heads/master",
        "isMerged": "true",
        "repositoryName": "MyDemoRepo"
      }
    ],
    "title": "Global variable update"
  }
}
```
"mergeMetadata": {
    "isMerged": true,
    "mergedBy": "arn:aws:iam::111111111111:user/Mary_Major"
},
"repositoryName": "MyDemoRepo",
"sourceCommit": "99132ab0EXAMPLE",
"sourceReference": "refs/heads/variables-branch"
},
"title": "Consolidation of global variables"}
Working with Commits in AWS CodeCommit Repositories

Commits to a repository are snapshots of the contents and changes to the contents of your repository. Every time a user commits and pushes a change, that information is saved and stored. So, too, is information that includes who committed the change, the date and time of the commit, and the changes made as part of the commit. You can also add tags to commits, to easily identify specific commits. In AWS CodeCommit, you can:

- Review commits.
- View the history of commits in a graph.
- Compare a commit to its parent or to another specifier.
- Add comments to your commits and reply to comments made by others.

Before you can push commits to an AWS CodeCommit repository, you must set up your local computer to connect to the repository. For the simplest method, see For HTTPS Users Using Git Credentials (p. 5).

For information about working with other aspects of your repository in AWS CodeCommit, see Working with Repositories (p. 75), Working with Files (p. 130), Working with Pull Requests (p. 138), Working with Branches (p. 194), and Working with User Preferences (p. 208).

Topics
- Create a Commit in AWS CodeCommit (p. 164)
- View Commit Details in AWS CodeCommit (p. 166)
- Compare Commits in AWS CodeCommit (p. 173)
Create a Commit in AWS CodeCommit

Follow these steps to use Git to create a commit in a local repo. If the local repo is connected to an AWS CodeCommit repository, you use Git to push the commit from the local repo to the AWS CodeCommit repository.

1. Complete the prerequisites, including Setting Up (p. 4).

   **Important**
   If you have not completed setup, you will not be able to connect or commit to the repository.

2. Make sure you are creating a commit in the desired branch. To see a list of available branches and find out which branch you are currently set to use, run `git branch`. All branches will be displayed. An asterisk (*) will appear next to your current branch. To switch to a different branch, run `git checkout branch-name`.

3. Make a change to the branch (such as adding, modifying, or deleting a file).

   For example, in the local repo, create a file named `bird.txt` with the following text:

   ```
   bird.txt
   --------
   Birds (class Aves or clade Avialae) are feathered, winged, two-legged, warm-blooded, egg-laying vertebrates.
   ```

4. Run `git status`, which should indicate that `bird.txt` has not yet been included in any pending commit:

   ```
   ... Untracked files:
   (use "git add <file>..." to include in what will be committed)
   bird.txt
   ```

5. Run `git add bird.txt` to include the new file in the pending commit.

6. If you run `git status` again, you should see output similar to the following. It indicates that `bird.txt` is now part of the pending commit or staged for commit:

   ```
   ... Changes to be committed:
   (use "git reset HEAD <file>..." to unstage)
   new file:   bird.txt
   ```

7. To finalize the commit, run `git commit` with the `-m` option (for example, `git commit -m "Adding bird.txt to the repository."`) The `-m` option creates the commit message.

8. If you run `git status` again, you should see output similar to the following. It indicates that the commit is ready to be pushed from the local repo to the AWS CodeCommit repository:

   ```
   ... nothing to commit, working directory clean
   ```
9. Before you push the finalized commit from the local repo to the AWS CodeCommit repository, you can see what will be pushed by running `git diff --stat remote-name/branch-name`, where `remote-name` is the nickname the local repo uses for the AWS CodeCommit repository and `branch-name` is the name of the branch to compare.

**Tip**
To get the nickname, run `git remote`. To get a list of branch names, run `git branch`. An asterisk (*) will appear next to the current branch. You can also run `git status` to get the current branch name.

**Note**
If you cloned the repository, from the local repo's perspective, `remote-name` is not the name of the AWS CodeCommit repository. When you clone a repository, `remote-name` is set automatically to `origin`.

For example, `git diff --stat origin/master` would show output similar to the following:

```
bird.txt | 1 +
1 file changed, 1 insertion(+)
```

Of course, the output assumes you have already connected the local repo to the AWS CodeCommit repository. (For instructions, see Connect to a Repository (p. 78).)

10. When you're ready to push the commit from the local repo to the AWS CodeCommit repository, run `git push remote-name branch-name`, where `remote-name` is the nickname the local repo uses for the AWS CodeCommit repository and `branch-name` is the name of the branch to push to the AWS CodeCommit repository.

For example, running `git push origin master` would show output similar to the following:

For HTTPS:

```
Counting objects: 7, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (5/5), 516 bytes | 0 bytes/s, done.
Total 5 (delta 2), reused 0 (delta 0)
remote: To https://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo
  b9e7aa6..3dbf4dd master -> master
```

For SSH:

```
Counting objects: 7, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (5/5), 516 bytes | 0 bytes/s, done.
Total 5 (delta 2), reused 0 (delta 0)
remote: To ssh://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo
  b9e7aa6..3dbf4dd master -> master
```

**Tip**
If you add the `-u` option to `git push` (for example, `git push -u origin master`), then you only need to run `git push` in the future because upstream tracking information has been set. To get upstream tracking information, run `git remote show remote-name` (for example, `git remote show origin`).

For more options, see your Git documentation.
View Commit Details in AWS CodeCommit

You can use the AWS CodeCommit console to browse the history of commits in a repository. This can help you identify changes made in a repository, including:

- When and by whom the changes were made.
- When specific commits were merged into a particular branch.

Viewing the history of commits for a branch might also help you understand the difference between branches. If you use tagging, you can also quickly view the commit that was labeled with a specific tag and the parents of that tagged commit. At the command line, you can use Git to view details about the commits in a local repo or an AWS CodeCommit repository.

Browse Commits in a Repository

You can use the AWS CodeCommit console to browse the history of commits to a repository. You can also view a graph of the commits in the repository and its branches over time. This can help you understand the history of the repository, including when changes were made.

Note
Using the `git rebase` command to rebase a repository changes the history of a repository, which might cause commits to appear out of order. For more information, see Git Branching-Rebasing or your Git documentation.

Topics
- Browse the Commit History of a Repository (p. 166)
- View a Graph of the Commit History of a Repository (p. 167)

Browse the Commit History of a Repository

You can browse the commit history for a specific branch or tag of the repository, including information about the committer and the commit message. You can also view the code for a specific commit.

To browse the history of commits (console)

2. On the Dashboard page, from the list of repositories, choose the repository for which you want to review the commit history.
3. In the navigation pane, choose Commits. In the commit history view, a history of commits for the repository in the default branch is displayed, in reverse chronological order of the commit date. Date and time are in coordinated universal time (UTC). You can view the commit history of a different branch by choosing the view selector button and then choosing a branch from the list. If you are using tags in your repository, you can view a commit with a specific tag and its parents by choosing that tag in the view selector button.
4. To view the difference between a commit and its parent, and to see any comments on the changes, choose the abbreviated commit ID. For more information, see Compare a Commit to Its Parent (p. 174) and Comment on a Commit (p. 179). To view the difference between a commit and any other commit specifier, including a branch, tag, or commit ID, see Compare Any Two Commit Specifiers (p. 177).

5. Do one or more of the following:
   - To view the date and time a change was made, hover over the days or months ago description.
   - To view the email associated with the author, hover over the user name.
   - To view the full commit ID, copy and then paste it into a text editor or other location. To copy it, choose the copy icon.
   - To view the code as it was at the time of a commit, choose the code icon (< >) for the commit. The contents of the repository as they were at the time of that commit is displayed in the Code view. The view selector button displays the abbreviated commit ID instead of a branch or tag.
   - If the full commit subject is too long to fit in the initial view, choose the arrow next to the message. The commit message box expands to display up to 5,000 characters of the subject and message.
   - To collapse the list of commits for a particular date, choose the arrow next to that date.

**View a Graph of the Commit History of a Repository**

You can view a graph of the commits made to a repository. The Commit Visualizer view is a directed acyclic graph (DAG) representation of all the commits made to a branch of the repository. This graphical representation can help you understand when commits and associated features were added or merged. It can also help you pinpoint when a change was made in relation to other changes.

**Note**
Commits that are merged using the fast-forward method do not appear as separate lines in the graph of commits.

**To view a graph of commits (console)**

2. On the Dashboard page, from the list of repositories, choose the repository for which you want to view a commit graph.
3. In the navigation pane, choose Visualizer.
In the commit graph, the subject for each commit message appears next to that point in the graph. You can use the direction buttons to change which side of the graph shows branches.

**Note**
The graph can display up to 35 branches on a page. If there are more than 35 branches, the graph is too complex to display. You can simplify the view in two ways:

- By using the view selector button to show the graph for a specific branch.
- By pasting a full commit ID into the search box to render the graph from that commit.

4. To see more details about a commit point, choose the point in the graph.

The detail view shows:
Use the AWS CLI to View Commit Details

Git lets you view details about commits. You can also use the AWS CLI to view details about the commits in a local repo or in an AWS CodeCommit repository, by running the following commands:

- `aws codecommit get-commit (p. 169)`, to view information about a commit.
- `aws codecommit get-differences (p. 170)`, to view information about changes for a commit specifier (branch, tag, HEAD, or other fully qualified references, such as commit IDs).
- `aws codecommit get-blob (p. 171)`, to view the base64-encoded content of an individual Git blob object in a repository.

To view information about a commit

1. Run the `aws codecommit get-commit` command, specifying:
   - The AWS CodeCommit repository's name (with the `--repository-name` option).
• The full commit ID.

For example, to view information about a commit with the ID `317f8570EXAMPLE` in an AWS CodeCommit repository named `MyDemoRepo`:

```
aws codecommit get-commit --repository-name MyDemoRepo --commit-id 317f8570EXAMPLE
```

2. If successful, the output of this command includes the following:

• Information about the author of the commit (as configured in Git), including the date in time stamp format and the coordinated universal time (UTC) offset.
• Information about the committer (as configured in Git) including the date in time stamp format and the UTC offset.
• The ID of the Git tree where the commit exists.
• The commit ID of the parent commit.
• The commit message.

Here is some example output, based on the preceding example command:

```
{
  "commit": {
    "additionalData": ",",
    "committer": {
      "date": "1484167798 -0800",
      "name": "Mary Major",
      "email": "mary_major@example.com"
    },
    "author": {
      "date": "1484167798 -0800",
      "name": "Mary Major",
      "email": "mary_major@example.com"
    },
    "treeId": "347a3408EXAMPLE",
    "parents": [
      "4c925148EXAMPLE"
    ],
    "message": "Fix incorrect variable name"
  }
}
```

To view information about the changes for a commit specifier

1. Run the `aws codecommit get-differences` command, specifying:

• The name of the AWS CodeCommit repository (with the `--repository-name` option).
• The commit specifiers you want to get information about. Only `--after-commit-specifier` is required. If you do not specify `--before-commit-specifier`, all files current as of the `--after-commit-specifier` will be shown.

For example, to view information about the differences between commits with the IDs `317f8570EXAMPLE` and `4c925148EXAMPLE` in an AWS CodeCommit repository named `MyDemoRepo`:

```
aws codecommit get-differences --repository-name MyDemoRepo --after-commit-specifier 317f8570EXAMPLE
```

API Version 2015-04-13
Use the AWS CLI to View Commit Details

```bash
aws codecommit get-differences --repository-name MyDemoRepo --before-commit-specifier 317f8570EXAMPLE --after-commit-specifier 4c925148EXAMPLE
```

2. If successful, the output of this command includes the following:

- A list of differences, including the change type (A for added, D for deleted, or M for modified).
- The mode of the file change type.
- The ID of the Git blob object that contains the change.

Here is some example output, based on the preceding example command:

```json
{
  "differences": [
    {
      "afterBlob": {
        "path": "blob.txt",
        "blobId": "2eb4af3bEXAMPLE",
        "mode": "100644"
      },
      "changeType": "M",
      "beforeBlob": {
        "path": "blob.txt",
        "blobId": "bf7fcf28fEXAMPLE",
        "mode": "100644"
      }
    }
  ]
}
```

To view information about a Git blob object

1. Run the `aws codecommit get-blob` command, specifying:
   - The name of the AWS CodeCommit repository (with the `--repository-name` option).
   - The ID of the Git blob (with the `--blob-id` option).

   For example, to view information about a Git blob with the ID of 2eb4af3bEXAMPLE in an AWS CodeCommit repository named MyDemoRepo:

   ```bash
   aws codecommit get-blob --repository-name MyDemoRepo --blob-id 2eb4af3bEXAMPLE
   ```

2. If successful, the output of this command includes the following:

   - The base64-encoded content of the blob, usually a file.

   For example, the output of the previous command might be similar to the following:

   ```json
   {
     "content": "QSBCaW5hcnkgTGFyToEXAMPLE="
   }
   ```
Use Git to View Commit Details

Before you follow these steps, you should have already connected the local repo to the AWS CodeCommit repository and committed changes. For instructions, see Connect to a Repository (p. 78).

To show the changes for the most recent commit to a repository, run the `git show` command.

```
git show
```

The command produces output similar to the following:

```
commit 4f8c6f9d
Author: Mary Major <mary.major@example.com>
Date:   Mon May 23 15:56:48 2016 -0700

    Added bumblebee.txt

diff --git a/bumblebee.txt b/bumblebee.txt
new file mode 100644
index 0000000..443b974
--- /dev/null
+++ b/bumblebee.txt
@@ -0,0 +1 @@
+A bumblebee, also written bumble bee, is a member of the bee genus Bombus, in the family Apidae.
\ No newline at end of file
```

**Note**

In this and the following examples, commit IDs have been abbreviated. The full commit IDs are not shown.

You can also use the `git show` command with the commit ID to view the changes that occurred:

```
git show 94ba1e60
```

```
commit 94ba1e60
Author: John Doe <johndoe@example.com>
Date:   Mon May 23 15:39:14 2016 -0700

    Added horse.txt

diff --git a/horse.txt b/horse.txt
new file mode 100644
index 0000000..080f68f
--- /dev/null
+++ b/horse.txt
@@ -0,0 +1 @@
+The horse (Equus ferus caballus) is one of two extant subspecies of Equus ferus.
```

To see the differences between two commits, run the `git diff` command and include the two commit IDs.

```
git diff ce22850d 4f8c6f9d
```

In this example, the difference between the two commits is that two files were added. The command produces output similar to the following:

```
diff --git a/bees.txt b/bees.txt
```
To use Git to view details about the commits in a local repo, run the `git log` command:

```
$ git log
```

If successful, this command produces output similar to the following:

```
commit 94ba1e60
Author: John Doe <johndoe@example.com>
Date: Mon May 23 15:39:14 2016 -0700
  Added horse.txt

commits 4c925148
Author: Jane Doe <janedoe@example.com>
Date: Mon May 22 14:54:55 2014 -0700
  Added cat.txt and dog.txt
```

To show only commit IDs and messages, run the `git log --pretty=oneline` command:

```
$ git log --pretty=oneline
```

If successful, this command produces output similar to the following:

```
94ba1e60 Added horse.txt
4c925148 Added cat.txt and dog.txt
```

For more options, see your Git documentation.

## Compare Commits in AWS CodeCommit

You can use the AWS CodeCommit console to view the differences between commit specifiers in an AWS CodeCommit repository. You can quickly view the difference between a commit and its parent. You can also compare any two references, including commit IDs.

### Topics
- [Compare a Commit to Its Parent](#)
- [Compare Any Two Commit Specifiers](#)
Compare a Commit to Its Parent

You can quickly view the difference between a commit and its parent to review the commit message, the committer, and exactly what changed.

2. On the Dashboard page, from the list of repositories, choose the repository where you want to view the difference between a commit and its parent.
3. In the navigation pane, choose Commits.
4. Choose the abbreviated commit ID of any commit in the list. The view changes to show details for this commit, including the differences between it and its parent commit.

You can show changes side by side (Split view) or inline (Unified view). You can also hide or show white space changes. You can also add comments. For more information, see Comment on a Commit (p. 179).

**Note**

If you are signed in as an IAM user, you can configure and save your preferences for viewing code and other console settings. For more information, see Working with User Preferences (p. 208).
Note
Depending on line ending style, your code editor, and other factors, you might see entire lines added or deleted instead of specific changes in a line. The level of detail matches what's returned in the `git show` or `git diff` commands.
5. To compare a commit to its parent from the **Commit Visualizer page**, choose a reference point on the graph, and then choose **View differences between this commit and its parent**.

This is an amended commit. Added results of second test

```python
- alv = "Number of alveolar hissing sibilants: {}"
- print(alv.format(total))
```
Compare Any Two Commit Specifiers

You can view the differences between any two commit specifiers in the AWS CodeCommit console. Commit specifiers are references, such as branches, tags, and commit IDs.

2. On the Dashboard page, from the list of repositories, choose the repository where you want to compare commits, branches, or tagged commits.
3. In the navigation pane, choose Compare.
4. Use the Choose buttons to compare two commit specifiers.
   - To compare the tip of a branch, choose the branch name. This selects the most recent commit from that branch for the comparison.
   - To compare a commit with a specific tag associated with it, choose the tag name. This selects the tagged commit for the comparison.
   - To compare a specific commit, paste the commit ID in the text box. To get the full commit ID, choose Commits in the navigation bar, and copy the commit ID from the list. On the Compare page, paste the full commit ID in the text box, and press Enter. You can repeat this to copy and paste a second commit ID, if you want to compare two commit IDs.
5. After you have selected the specifiers, choose **Compare**.

6. To reverse the comparison, choose the Flip button ( ), and then choose **Compare**.

7. To clear your comparison choices, choose **Clear**.
Comment on a Commit in AWS CodeCommit

You can use the AWS CodeCommit console to comment on commits in a repository, and view and reply to other users’ comments on commits. This can help you discuss changes made in a repository, including:

- Why specific changes were made.
- Whether more changes or fixes are required.
- Whether changes should be merged into another branch.

You can comment on an overall commit, on a file within a commit, or on a specific line or change within a file.

Note

For best results, use commenting when you are signed in as an IAM user. The commenting functionality is not optimized for users who sign in with root account credentials, federated access, or temporary credentials.

Topics

- View Comments on a Commit in a Repository (p. 179)
- Add and Reply to Comments on a Commit in a Repository (p. 181)
- Use the AWS CLI to View, Add, Update, and Reply to Comments (p. 185)

View Comments on a Commit in a Repository

You can use the AWS CodeCommit console to view comments on a commit.
To view comments on a commit (console)

2. On the Dashboard page, from the list of repositories, choose the repository for which you want to review comments on commits.
3. In the navigation pane, choose Commits. In the commit history view, a history of commits to the repository's default branch is displayed. Choose the title of the commit where you want to view any comments.

   The page for that specific commit is displayed, along with any comments already present on that commit, in Changes.

4. To only view the comments for a commit, choose Comments. All file and line comments are shown, along with any comments on the changes in the commit itself, in reverse chronological order.
Add and Reply to Comments on a Commit in a Repository

You can use the AWS CodeCommit console to add comments to the comparison of a commit and a parent, or to the comparison between two specified commits. You can also reply to existing comments.

Add and Reply to Comments on a Commit

You can add and reply to comments to a commit. Your comments are marked as those belonging to the IAM user or role you used to sign in to the console.

2. On the Dashboard page, from the list of repositories, choose the repository where you want to comment on commits.
3. In the navigation pane, choose Commits. In the commit history view, choose the title of the commit where you want to add or reply to comments.

The page for that commit is displayed, along with any comments.
4. To add a comment, do one of the following:
   - To add a general comment, in Comments on changes, type your comment, and then choose Save. You can use Markdown, or you can type your comment in plaintext.

To view the context of a comment, choose Show context. The view changes to where the comment was made in the Changes view.
To add a comment to a file in the commit, in **Changes**, find the name of the file. Choose the comment bubble, type your comment, and then choose **Save**.
• To add a comment to a changed line in the commit, in **Changes**, go to the line where the change appears. Choose the comment bubble 📝, type your comment, and then choose **Save**.

**Note**
You can edit your comment after you have saved it, but you cannot delete it from the AWS CodeCommit console. Consider using the **Preview** mode for your comment before you save your comment.

5. To reply to comments on a commit, in either **Changes** or **Comments**, choose **Reply**.
Add and Reply to Comments When Comparing Two Commit Specifiers

You can add comments to a comparison between branches, tags, or commits.

To add or reply to comments when comparing commit specifiers (console)

2. On the Dashboard page, from the list of repositories, choose the repository where you want to compare commits, branches, or tagged commits.
3. In the navigation pane, choose **Compare**.
4. Use the Choose buttons to compare two commit specifiers. Use the drop-down lists or paste in commit IDs.

5. Do one or more of the following:

   - To add comments to files or lines, choose the comment bubble.
   - To add general comments on the compared changes, go to Comments on changes.

Use the AWS CLI to View, Add, Update, and Reply to Comments

You can view, add, reply, update, and delete the contents of a comment by running the following commands:

- `get-comments-for-compared-commit (p. 185)`, to view the comments on the comparison between two commits.
- `get-comment (p. 186)`, to view details on a specific comment.
- `delete-comment-content (p. 187)`, to delete the contents of a comment that you created.
- `post-comment-for-compared-commit (p. 187)`, to create a comment on the comparison between two commits.
- `update-comment (p. 188)`, to update a comment.
- `post-comment-reply (p. 189)`, to reply to a comment.

To view comments on a commit

1. Run the `get-comments-for-compared-commit` command, specifying:

   - The AWS CodeCommit repository's name (with the `--repository-name` option).
   - The full commit ID of the 'after' commit, to establish the directionality of the comparison (with the `--after-commit-id` option).
   - The full commit ID of the 'before' commit, to establish the directionality of the comparison (with the `--before-commit-id` option).
• (Optional) An enumeration token to return the next batch of the results (with the `--next-token` option).

• (Optional) A non-negative integer to limit the number of returned results (with the `--max-results` option).

For example, to view comments made on the comparison between two commits in a repository named `MyDemoRepo`:

```
aws codecommit get-comments-for-compared-commit --repository-name MyDemoRepo --before-commit-ID 6e147360EXMAPLE --after-commit-id 317f8570EXAMPLE
```

2. If successful, this command produces output similar to the following:

```
{
    "commentsForComparedCommitData": [
        {
            "afterBlobId": "1f330709EXAMPLE",
            "afterCommitId": "317f8570EXAMPLE",
            "beforeBlobId": "80906a4cEXAMPLE",
            "beforeCommitId": "6e147360EXAMPLE",
            "comments": [
                {
                    "authorArn": "arn:aws:iam::111111111111:user/Li_Juan",
                    "clientRequestToken": "123Example",
                    "commentId": "ff30b348EXAMPLEb9aa670f",
                    "content": "Whoops - I meant to add this comment to the line, not the file, but I don't see how to delete it.",
                    "creationDate": 1508369768.142,
                    "deleted": false,
                    "commitId": "123abc-EXAMPLE",
                    "lastModifiedDate": 1508369842.278
                },
                {
                    "authorArn": "arn:aws:iam::111111111111:user/Li_Juan",
                    "clientRequestToken": "123Example",
                    "commentId": "553b509bEXAMPLE56198325",
                    "content": "Can you add a test case for this?",
                    "creationDate": 1508369612.240,
                    "deleted": false,
                    "commitId": "456def-EXAMPLE",
                    "lastModifiedDate": 1508369612.240
                }
            ],
            "location": {
                "filePath": "cl_sample.js",
                "filePosition": 1232,
                "relativeFileVersion": "after"
            },
            "repositoryName": "MyDemoRepo"
        }
    ],
    "nextToken": "exampleToken"
}
```

To view details of a comment on a commit

1. Run the `get-comment` command, specifying the system-generated comment ID. For example:
aws codecommit get-comment --comment-id ff30b348EXAMPLEb9aa670f

2. If successful, this command returns output similar to the following:

```json
{
  "comment": {
    "authorArn": "arn:aws:iam::111111111111:user/Li_Juan",
    "clientRequestToken": "123Example",
    "commentId": "ff30b348EXAMPLEb9aa670f",
    "content": "Whoops - I meant to add this comment to the line, but I don't see how to delete it.",
    "creationDate": 1508369768.142,
    "deleted": false,
    "commentId": "",
    "lastModifiedDate": 1508369842.278
  }
}
```

To delete the contents of a comment on a commit

1. Run the `delete-comment-content` command, specifying the system-generated comment ID. For example:

```bash
aws codecommit delete-comment-content --comment-id ff30b348EXAMPLEb9aa670f
```

**Note**

You can only delete the content of a comment if you created the comment.

2. If successful, this command produces output similar to the following:

```json
{
  "comment": {
    "creationDate": 1508369768.142,
    "deleted": true,
    "lastModifiedDate": 1508369842.278,
    "clientRequestToken": "123Example",
    "commentId": "ff30b348EXAMPLEb9aa670f",
    "authorArn": "arn:aws:iam::111111111111:user/Li_Juan"
  }
}
```

To create a comment on a commit

1. Run the `post-comment-for-compared-commit` command, specifying:

   - The AWS CodeCommit repository's name (with the `--repository-name` option).
   - The full commit ID of the 'after' commit, to establish the directionality of the comparison (with the `after-commit-id` option).
   - The full commit ID of the 'before' commit, to establish the directionality of the comparison (with the `before-commit-id` option).
   - A unique, client-generated idempotency token (with the `--client-request-token` option).
   - The content of your comment (with the `--content` option).
   - A list of location information about where to place the comment, including:

API Version 2015-04-13

187
• The name of the file being compared, including its extension and subdirectory, if any (with the `filePath` attribute).

• The line number of the change within a compared file (with the `filePosition` attribute).

• Whether the comment on the change is "before" or "after" in the comparison between the source and destination branches (with the `relativeFileVersion` attribute).

For example, to add the comment "Can you add a test case for this?" on the change to the `cl_sample.js` file in the comparison between two commits in a repository named `MyDemoRepo`:

```bash
aws codecommit post-comment-for-compared-commit --repository-name MyDemoRepo --before-commit-id 317f8570EXAMPLE --after-commit-id 5d036259EXAMPLE --client-request-token 123Example --content "Can you add a test case for this?" --location filePath=cl_sample.js,filePosition=1232,relativeFileVersion=AFTER
```

2. If successful, this command produces output similar to the following:

```json
{
   "afterBlobId": "1f330709EXAMPLE",
   "afterCommitId": "317f8570EXAMPLE",
   "beforeBlobId": "8090ea4cEXAMPLE",
   "beforeCommitId": "6e147360EXAMPLE",
   "comment": {
      "authorArn": "arn:aws:iam::111111111111:user/Li_Juan",
      "clientRequestToken": "",
      "commentId": "553b509bEXAMPLE56198325",
      "content": "Can you add a test case for this?",
      "creationDate": 1508369612.203,
      "deleted": false,
      "commentId": "abc123-EXAMPLE",
      "lastModifiedDate": 1508369612.203
   },
   "location": {
      "filePath": "cl_sample.js",
      "filePosition": 1232,
      "relativeFileVersion": "AFTER"
   },
   "repositoryName": "MyDemoRepo"
}
```

To update a comment on a commit

1. Run the `update-comment` command, specifying the system-generated comment ID and the content with which you want to replace any existing content.

   **Note**
   You can only update the content of a comment if you created the comment.

   For example, to add the content "Fixed as requested. I'll update the pull request." to a comment with an ID of `442b498bEXAMPLE5756813`:

   ```bash
   aws codecommit update-comment --comment-id 442b498bEXAMPLE5756813 --content "Fixed as requested. I'll update the pull request."
   ```

2. If successful, this command produces output similar to the following:

   ```json
   {
   }
To reply to a comment on a commit

1. To post a reply to a comment in a pull request, run the `post-comment-reply` command, specifying:
   - The system-generated ID of the comment to which you want to reply (with the `--in-reply-to` option).
   - A unique, client-generated idempotency token (with the `--client-request-token` option).
   - The content of your reply (with the `--content` option).

   For example, to add the reply "Good catch. I’ll remove them." to the comment with the system-generated ID of `abcd1234EXAMPLEb5678efgh`:

   ```bash
   aws codecommit post-comment-reply --in-reply-to abcd1234EXAMPLEb5678efgh --content "Good catch. I’ll remove them." --client-request-token 123Example
   ```

2. If successful, this command produces output similar to the following:

   ```json
   {
   "comment": {
       "authorArn": "arn:aws:iam::111111111111:user/Li_Juan",
       "clientRequestToken": "123Example",
       "commentId": "442b498bEXAMPLE5756813",
       "content": "Good catch. I’ll remove them."
   }
   }
   ```

Create a Tag in AWS CodeCommit

You can use a tag to mark a commit with a label that helps other repository users understand its importance. To create a tag in an AWS CodeCommit repository, you can use Git from a local repo connected to the AWS CodeCommit repository. After you have created a tag in the local repo, you can use `git push --tags` to push it to the AWS CodeCommit repository.

For more information about how to view tags in your repository, see View Tag Details (p. 190).

Use Git to Create a Tag

To use Git from a local repo to create a tag in an AWS CodeCommit repository, follow these steps.
In these steps, we assume that you have already connected the local repo to the AWS CodeCommit repository. For instructions, see Connect to a Repository (p. 78).

1. Run the `git tag new-tag-name commit-id` command, where `new-tag-name` is the new tag's name and `commit-id` is the ID of the commit to associate with the tag.

   For example, the following command creates a new tag named `beta` and associates it with the commit ID `dc082f9a...af873b88`:

   ```sh
   git tag beta dc082f9a...af873b88
   ```

2. To push the new tag from the local repo to the AWS CodeCommit repository, run the `git push remote-name new-tag-name` command, where `remote-name` is the name of the AWS CodeCommit repository and `new-tag-name` is the name of the new tag.

   For example, to push a new tag named `beta` to an AWS CodeCommit repository named `origin`:

   ```sh
   git push origin beta
   ```

   **Note**
   To push all new tags from your local repo to the AWS CodeCommit repository, run `git push --tags`. To ensure your local repo is updated with all of the tags in the AWS CodeCommit repository, run `git fetch` followed by `git fetch --tags`.

For more options, see your Git documentation.

---

## View Tag Details in AWS CodeCommit

In Git, a tag is a label you can apply to a reference like a commit to mark it with information that might be important to other repository users. For example, you might tag the commit that was the beta release point for a project with the tag `beta`. For more information, see Use Git to Create a Tag (p. 189).

You can use the AWS CodeCommit console to view information about tags in your repository, including the date and commit message of the commit referenced by each tag. From the console, you can compare the commit referenced by the tag with the head of the default branch of your repository. Like any other commit, you can also view the code at the point of that tag.

You can also use Git from your terminal or command line to view details about tags in a local repo.

**Topics**
- Use the AWS CodeCommit Console to View Tag Details (p. 190)
- Use Git to View Tag Details (p. 191)

---

### Use the AWS CodeCommit Console to View Tag Details

Use the AWS CodeCommit console to quickly view a list of tags for your repository and details about the commits referenced by the tags.

2. In the list of repositories, choose the name of the repository.
3. In the navigation pane, choose **Tags**.

![AWS CodeCommit User Guide](image)

**Note**
You can adjust the number of tags displayed on the **Tags** page by changing the number of tags per page.

4. Do one of the following:
   - To view the code as it was at that tagged commit, choose the tag name.
   - To view a graph of the repository from that tagged commit, choose the abbreviated commit ID.
   - To view details of the commit, including the full commit message, committer, and author, choose the commit message.
   - To compare the tagged commit with the head of the default branch in your repository, choose **Compare**.

### Use Git to View Tag Details

To use Git to view details about tags in a local repo, run one of the following commands:

- **`git tag`** (p. 191) to view a list of tag names.
- **`git show`** (p. 192) to view information about a specific tag.
- **`git ls-remote`** (p. 192) to view information about tags in an AWS CodeCommit repository.

**Note**
To ensure that your local repo is updated with all of the tags in the AWS CodeCommit repository, run **`git fetch`** followed by **`git fetch --tags`**.

In the following steps, we assume that you have already connected the local repo to AWS CodeCommit repository. For instructions, see **Connect to a Repository** (p. 78).

### To view a list of tags in a local repo

1. Run the **`git tag`** command:
Use Git to View Tag Details

2. If successful, this command produces output similar to the following:

```
beta
release
```

**Note**
If no tags have been defined, `git tag` returns nothing.

For more options, see your Git documentation.

**To view information about a tag in a local repo**

1. Run the `git show tag-name` command. For example, to view information about a tag named `beta`, run:

```
git show beta
```

2. If successful, this command produces output similar to the following:

```
commit 317f8570...ad9e3c09
Author: John Doe <johndoe@example.com>
Date:   Tue Sep 23 13:49:51 2014 -0700

  Added horse.txt

diff --git a/horse.txt b/horse.txt
new file mode 100644
index 0000000..df42ff1
  --- /dev/null
  +++ b/horse.txt
  @@ -0,0 +1 @@
  +The horse (Equus ferus caballus) is one of two extant subspecies of Equus ferus
  
  No newline at end of file
```

**Note**
To exit the output of the tag information, type `:q`.

For more options, see your Git documentation.

**To view information about tags in an AWS CodeCommit repository**

1. Run the `git ls-remote --tags` command.

```
git ls-remote --tags
```

2. If successful, this command produces as output a list of the tags in the AWS CodeCommit repository:

```
129ce87a...70fbbfba refs/tags/beta
785de9bd...59b402d8 refs/tags/release
```

If no tags have been defined, `git ls-remote --tags` returns a blank line.
Delete a Tag in AWS CodeCommit

To delete a tag in an AWS CodeCommit repository, use Git from a local repo connected to the AWS CodeCommit repository.

Use Git to Delete a Tag

To use Git from a local repo to delete a tag in an AWS CodeCommit repository, follow these steps.

These steps assume you have already connected the local repo to the AWS CodeCommit repository. For instructions, see Connect to a Repository (p. 78).

1. To delete the tag from the local repo, run the `git tag -d tag-name` command where `tag-name` is the name of the tag you want to delete.
   
   Tip
   To get a list of tag names, run `git tag`.

   For example, to delete a tag in the local repo named `beta`:

   ```
git tag -d beta
   ```

2. To delete the tag from the AWS CodeCommit repository, run the `git push remote-name --delete tag-name` command where `remote-name` is the nickname the local repo uses for the AWS CodeCommit repository and `tag-name` is the name of the tag you want to delete from the AWS CodeCommit repository.
   
   Tip
   To get a list of AWS CodeCommit repository names along with their URLs, run the `git remote -v` command.

   For example, to delete a tag named `beta` in the AWS CodeCommit repository named `origin`:

   ```
git push origin --delete beta
   ```
What is a branch? In Git, branches are simply pointers or references to a commit. In development, they're a convenient way to organize your work. You can use branches to separate work on a new or different version of files without impacting work in other branches. You can use branches to develop new features, store a specific version of your project from a particular commit, and more.

In AWS CodeCommit, you can change the default branch for your repository. This default branch is the one used as the base or default branch in local repos when users clone the repository. You can also create and delete branches and view details about a branch. You can quickly compare differences between a branch and the default branch (or any two branches). To view the history of branches and merges in your repository, you can use the Commit Visualizer (p. 167).

For information about working with other aspects of your repository in AWS CodeCommit, see Working with Repositories (p. 75), Working with Files (p. 130), Working with Pull Requests (p. 138), Working with Commits (p. 163), and Working with User Preferences (p. 208).

Topics
- Create a Branch in AWS CodeCommit (p. 195)
- View Branch Details in AWS CodeCommit (p. 197)
- Compare Branches in AWS CodeCommit (p. 200)
- Change Branch Settings in AWS CodeCommit (p. 203)
Create a Branch in AWS CodeCommit

You can use the AWS CodeCommit console to create branches for your repository. This is a quick way to separate work on a new or different version of files without impacting work in the default branch. After creating a branch in the AWS CodeCommit console, you'll need to pull that change to your local repo. Alternatively, you can create a branch locally and push that change to an AWS CodeCommit repository by using Git from a local repo connected to the AWS CodeCommit repository. You can also use the AWS CLI.

**Topics**
- Use the AWS CodeCommit Console to Create a Branch (p. 195)
- Use Git to Create a Branch (p. 196)
- Use the AWS CLI to Create a Branch (p. 196)

**Use the AWS CodeCommit Console to Create a Branch**

You can use the AWS CodeCommit console to create a branch in an AWS CodeCommit repository. When users next pull changes from the repository, they will see the new branch.

2. In the list of repositories, choose the name of the repository.
3. In the navigation pane, choose **Branches**.
4. Choose **Create branch**.

Create a branch to work on changes without affecting the original source for the branch. You can create a branch from another branch, a tag, or from a full commit ID. **Learn more**

**Branch name**

new-branch

**Branch from**

Branch: master

*Required
Use Git to Create a Branch

To use Git from a local repo to create a branch in an local repo and then push that branch to the AWS CodeCommit repository, follow these steps.

These steps assume you have already connected the local repo to the AWS CodeCommit repository. For instructions, see Connect to a Repository (p. 78).

1. Create a new branch in your local repo by running the `git checkout -b new-branch-name` command, where `new-branch-name` is the name of the new branch.

For example, the following command creates a new branch named `MyNewBranch` in the local repo:

```
git checkout -b MyNewBranch
```

2. To push the new branch from the local repo to the AWS CodeCommit repository, run the `git push` command, specifying both the `remote-name` and the `new-branch-name`.

For example, to push a new branch in the local repo named `MyNewBranch` to the AWS CodeCommit repository with the nickname `origin`:

```
git push origin MyNewBranch
```

**Note**

If you add the `-u` option to `git push` (for example, `git push -u origin master`), then in the future you can run `git push` without `remote-name branch-name`. Upstream tracking information will be set. To get upstream tracking information, run `git remote show remote-name` (for example, `git remote show origin`).

To set up a branch in the local repo that is connected to an existing branch in the AWS CodeCommit repository, run `git checkout remote-branch-name`.

For more options, see your Git documentation.

Use the AWS CLI to Create a Branch

To use AWS CLI commands with AWS CodeCommit, install the AWS CLI. For more information, see Command Line Reference (p. 295).

To use the AWS CLI to create a branch in an AWS CodeCommit repository and then push that branch to the AWS CodeCommit repository, follow these steps.

1. Run the `create-branch` command, specifying:

   - The name of the AWS CodeCommit repository where the branch will be created (with the `--repository-name` option).

   **Note**

   To get the name of the AWS CodeCommit repository, run the `list-repositories` (p. 111) command.
• The name of the new branch (with the `--branch-name` option).
• The ID of the commit to which the new branch will point (with the `--commit-id` option).

For example, to create a new branch named `MyNewBranch` that points to commit ID `317f8570EXAMPLE` in an AWS CodeCommit repository named `MyDemoRepo`:

```
aws codecommit create-branch --repository-name MyDemoRepo --branch-name MyNewBranch --commit-id 317f8570EXAMPLE
```

This command produces output only if there are errors.

2. To update your local repo's list of available AWS CodeCommit repository branches with the new remote branch name, run `git remote update remote-name`.

For example, to update your local repo's list of available branches for the AWS CodeCommit repository with the nickname `origin`:

```
git remote update origin
```

**Note**

Alternatively, you can run the `git fetch` command. You can also view all remote branches by running `git branch --all`, but until you update your local repo's list, the remote branch you created will not appear in the list.

For more options, see your Git documentation.

3. To set up a branch in the local repo that is connected to the new branch in the AWS CodeCommit repository, run `git checkout remote-branch-name`.

**Note**

To get a list of AWS CodeCommit repository names, along with their URLs, run the `git remote -v` command.

**View Branch Details in AWS CodeCommit**

To view details about the branches in an AWS CodeCommit repository, you can use the AWS CodeCommit console. You can view the date of the last commit to a branch, the commit message, and more. You can also use Git from a local repo connected to the AWS CodeCommit repository or the AWS CLI to view branch details.

**Topics**

- Use the AWS CodeCommit Console to View Branch Details (p. 197)
- Use Git to View Branch Details (p. 198)
- Use the AWS CLI to View Branch Details (p. 199)

**Use the AWS CodeCommit Console to View Branch Details**

Use the AWS CodeCommit console to quickly view a list of branches for your repository and details about the branches.


API Version 2015-04-13
2. In the list of repositories, choose the name of the repository.
3. In the navigation pane, choose **Branches**.

4. You can adjust the number of branches displayed on the **Branches** page by changing the number of branches shown on a page. The name of the branch used as the default for the repository is displayed next to the **Default** label. To view details about the most recent commit to a branch, choose the commit message. To view the files and code in a branch, choose the branch name.

### Use Git to View Branch Details

To use Git from a local repo to view details about both the local and remote tracking branches for an AWS CodeCommit repository, run the **git branch** command.

The following steps assume you have already connected the local repo to the AWS CodeCommit repository. For instructions, see [Connect to a Repository](#).

1. Run the **git branch** command, specifying the **--all** option:

   ```shell
   git branch --all
   ```

2. If successful, this command returns output similar to the following:

   ```text
   MyNewBranch
   * master
   remotes/origin/MyNewBranch
   remotes/origin/master
   ```

   The asterisk (*) appears next to the currently open branch. The entries after that are remote tracking references.

   **Tip**
   - **`git branch`** shows local branches.
   - **`git branch -r`** shows remote branches.
git checkout *existing-branch-name* switches to the specified branch name and, if *git branch* is run immediately afterward, displays it with an asterisk (*).

*git remote update remote-name* updates your local repo with the list of available AWS CodeCommit repository branches. (To get a list of AWS CodeCommit repository names, along with their URLs, run the *git remote -v* command.)

For more options, see your Git documentation.

## Use the AWS CLI to View Branch Details

To use AWS CLI commands with AWS CodeCommit, install the AWS CLI. For more information, see Command Line Reference (p. 295).

To use the AWS CLI to view details about the branches in an AWS CodeCommit repository, run one or more of the following commands:

- list-branches (p. 199) to view a list of branch names.
- get-branch (p. 199) to view information about a specific branch.

### To view a list of branch names

1. Run the **list-branches** command, specifying the name of the AWS CodeCommit repository (with the **--repository-name** option).

   **Tip**

   To get the name of the AWS CodeCommit repository, run the **list-repositories** (p. 111) command.

   For example, to view details about the branches in an AWS CodeCommit repository named MyDemoRepo:

   ```bash
   aws codecommit list-branches --repository-name MyDemoRepo
   ```

2. If successful, this command outputs a `branchNameList` object, with an entry for each branch.

   Here is some example output based on the preceding example command:

   ```json
   {
     "branches": [
       "MyNewBranch",
       "master"
     ]
   }
   ```

### To view information about a branch

1. Run the **get-branch** command, specifying:

   - The repository name (with the **--repository-name** option).
   - The branch name (with the **--branch-name** option).

   For example, to view information about a branch named MyNewBranch in an AWS CodeCommit repository named MyDemoRepo:
aws codecommit get-branch --repository-name MyDemoRepo --branch-name MyNewBranch

2. If successful, this command outputs the name of the branch and the ID of the last commit made to the branch.

Here is some example output based on the preceding example command:

```json
{
    "branch": {
        "branchName": "MyNewBranch",
        "commitID": "317f8570EXAMPLE"
    }
}
```

## Compare Branches in AWS CodeCommit

You can compare branches in an AWS CodeCommit repository by using the AWS CodeCommit console. Comparing branches helps you quickly view the differences between a branch and the default branch, or view the differences between any two branches.

**Topics**
- Compare a Branch to the Default Branch (p. 200)
- Compare Two Specific Branches (p. 202)

### Compare a Branch to the Default Branch

Use the AWS CodeCommit console to quickly view the differences between a branch and the default branch for your repository.

2. In the list of repositories, choose the name of the repository.
3. In the navigation pane, choose Branches.
4. In the list of branches, find the branch you want to compare to the default branch, and then choose Compare.
The **Compare** view opens and displays the differences between the branch you chose and the default branch.
Compare Two Specific Branches

Use the AWS CodeCommit console to view the differences between two branches that you want to compare.

2. In the list of repositories, choose the name of the repository.
3. In the navigation pane, choose Compare.
4. Choose the two branches to compare, and then choose Compare. To view the list of changed files, expand the changed files list. You can view changes in files side by side (Split view) or inline (Unified view).

**Note**
If you are signed in as an IAM user, you can configure and save your preferences for viewing code and other console settings. For more information, see Working with User Preferences (p. 208).
Change Branch Settings in AWS CodeCommit

You can change the default branch to use in the AWS CodeCommit console. You can also use the AWS CLI to change the default branch for a repository. To change other branch settings, you can use Git from a local repo connected to the AWS CodeCommit repository.

Topics
- Use the AWS CodeCommit Console to Change the Default Branch (p. 203)
- Use the AWS CLI to Change Branch Settings (p. 204)

Use the AWS CodeCommit Console to Change the Default Branch

You can specify which branch is the default branch in an AWS CodeCommit repository in the AWS CodeCommit console.

2. In the list of repositories, choose the name of the repository where you want to change settings.
3. In the navigation pane, choose Branches.
4. Choose Change default branch. In the Default branch drop-down list, choose a different branch, and then choose Change.
Use the AWS CLI to Change Branch Settings

To use AWS CLI commands with AWS CodeCommit, install the AWS CLI. For more information, see Command Line Reference (p. 295).

To use the AWS CLI to change a repository’s branch settings in an AWS CodeCommit repository, run the following command:

- update-default-branch (p. 204) to change the default branch.

To change the default branch

1. Run the `update-default-branch` command, specifying:
   - The name of the AWS CodeCommit repository where the default branch will be updated (with the --repository-name option).
     
     **Tip**
     To get the name of the AWS CodeCommit repository, run the list-repositories (p. 111) command.

   - The name of the new default branch (with the --default-branch-name option).
     
     **Tip**
     To get the name of the branch, run the list-branches (p. 199) command.

2. For example, to change the default branch to `MyNewBranch` in an AWS CodeCommit repository named `MyDemoRepo`: 

   ```bash
   aws codecommit update-default-branch --repository-name MyDemoRepo --default-branch-name MyNewBranch
   ```
Delete a Branch in AWS CodeCommit

You can use the AWS CodeCommit console to delete a branch in a repository. Deleting a branch in AWS CodeCommit does not delete that branch in a local repo, so users might continue to have copies of that branch until the next time they pull changes. To delete a branch locally and push that change to the AWS CodeCommit repository, use Git from a local repo connected to the AWS CodeCommit repository.

Deleting a branch does not delete any commits, but it does delete all references to the commits in that branch. If you delete a branch that contains commits that have not been merged into another branch in the repository, you cannot retrieve those commits unless you have their full commit IDs.

Note
You cannot use the instructions in this topic to delete a repository's default branch. If you want to delete the default branch, you must create a new branch, make the new branch the default branch, and then delete the old branch. To learn how to create a new branch, see Create a Branch (p. 195). To learn how to make a branch the default branch, see Change Branch Settings (p. 203).

Topics
- Use the AWS CodeCommit Console to Delete a Branch (p. 205)
- Use the AWS CLI to Delete a Branch (p. 206)
- Use Git to Delete a Branch (p. 207)

Use the AWS CodeCommit Console to Delete a Branch

You can use the AWS CodeCommit console to delete a branch in an AWS CodeCommit repository.

2. In the list of repositories, choose the name of the repository.
3. In the navigation pane, choose Branches.
4. Find the name of the branch that you want to delete, and choose the delete icon. In the confirmation dialog, choose Delete.
Use the AWS CLI to Delete a Branch

You can use the AWS CLI to delete a branch in an AWS CodeCommit repository, if that branch is not the default branch for the repository. For more information about installing and using the AWS CLI, see Command Line Reference (p. 295).

1. At the terminal or command line, run the `delete-branch` command, specifying:
   - The name of the AWS CodeCommit repository where the branch will be deleted (with the `--repository-name` option).
     
     **Tip**
     To get the name of the AWS CodeCommit repository, run the list-repositories (p. 111) command.
   - The name of the branch to delete (with the `branch-name` option).
     
     **Tip**
     To get the name of the branch, run the list-branches (p. 199) command.

2. For example, to delete a branch named `MyNewBranch` in an AWS CodeCommit repository named `MyDemoRepo`:

   ```bash
   aws codecommit delete-branch --repository-name MyDemoRepo --branch-name MyNewBranch
   ```

   This command returns information about the deleted branch, including the name of the deleted branch and the full commit ID of the commit that was the head of the branch. For example:

   ```json
   "deletedBranch": {
   ```
Use Git to Delete a Branch

To use Git from a local repo to delete a branch in an AWS CodeCommit repository, follow these steps.

These steps assume you have already connected the local repo to the AWS CodeCommit repository. For instructions, see Connect to a Repository (p. 78).

1. To delete the branch from the local repo, run the `git branch -D branch-name` command where `branch-name` is the name of the branch you want to delete.

   Tip
   To get a list of branch names, run `git branch --all`.

   For example, to delete a branch in the local repo named `MyNewBranch`:

   ```
git branch -D MyNewBranch
   ```

2. To delete the branch from the AWS CodeCommit repository, run the `git push remote-name --delete branch-name` command where `remote-name` is the nickname the local repo uses for the AWS CodeCommit repository and `branch-name` is the name of the branch you want to delete from the AWS CodeCommit repository.

   Tip
   To get a list of AWS CodeCommit repository names along with their URLs, run the `git remote -v` command.

   For example, to delete a branch named `MyNewBranch` in the AWS CodeCommit repository named `origin`:

   ```
git push origin --delete MyNewBranch
   ```

   Tip
   This command will not delete a branch if it is the default branch.

   For more options, see your Git documentation.
Working with User Preferences

Some default settings in the AWS CodeCommit console can be configured. For example, you can change the number of repositories displayed on the dashboard. If you are signed in to the console as an IAM user, you can store information about how you prefer to use the AWS CodeCommit console, also known as your user preferences. This information is stored and applied every time you use the console. These preferences are applied to all repositories in all regions for your IAM user any time you access the AWS CodeCommit console. They are not repository-specific or region-specific. They do not have any effect on your interactions with the AWS CLI, AWS CodeCommit API, or other services that interact with AWS CodeCommit.

You can still change individual settings on console pages without having saved user preferences. Those choices persist until you close the console window. When you return to the console, any saved user preferences are applied.

**Note**

User preferences are only available for IAM users. You cannot set them if you use federated access, temporary access, or a root account to access the console.

User preferences include:

- When viewing a list of repositories in your AWS account, the number of repositories displayed on the dashboard.
- When viewing changes in code, whether to use **Unified** or **Split** view, and whether to show or hide whitespace changes.
- When viewing a graph of commits, whether to display commits on branches to the left or to the right of the default branch.

View and Save User Preferences

You can view and change your user preferences for the AWS CodeCommit console. These settings apply only to your IAM user in the AWS CodeCommit console. They do not affect other IAM users in your AWS account.

**To view and save preferences in the AWS CodeCommit console for your IAM user**


   You must sign in as an IAM user. You cannot configure user preferences for other user types.

2. In the title bar navigation, choose **User preferences**.

3. In **User preferences**, make your changes to configure your preferences.
Do one of the following:

- To save and apply your changes, choose **Save**.
• To view the AWS CodeCommit console defaults, choose **Restore**. These defaults are applied if you choose **Save**.
• To return to the console where you left off, choose **Back**. Alternatively, choose **Dashboard** to go to the AWS CodeCommit console dashboard.
Migrate to AWS CodeCommit

You can migrate a Git repository to an AWS CodeCommit repository in a number of ways: by cloning it, mirroring it, migrating all or just some of the branches, and so on. You can also migrate local, unversioned content on your computer to AWS CodeCommit.

The following topics demonstrate some of the ways you can choose to migrate a repository. Your steps may vary, depending on the type, style, or complexity of your repository and the decisions you make about what and how you want to migrate. For very large repositories, you might want to consider migrating incrementally (p. 227).

Note
You can migrate to AWS CodeCommit from other version control systems, such as Perforce, Subversion, or TFS, but you will have to migrate to Git first.
For more options, see your Git documentation.
Alternatively, you can review the information about migrating to Git in the Pro Git book by Scott Chacon and Ben Straub.

Topics
- Migrate a Git Repository to AWS CodeCommit (p. 211)
- Migrate Local or Unversioned Content to AWS CodeCommit (p. 219)
- Migrate a Repository Incrementally (p. 227)

Migrate a Git Repository to AWS CodeCommit

You can migrate an existing Git repository to an AWS CodeCommit repository. The procedures in this topic walk you through the process of migrating a project hosted on another Git repository to AWS CodeCommit. As part of this process, you will:

- Complete the initial setup required for AWS CodeCommit.
- Create an AWS CodeCommit repository.
- Clone the repository and push it to AWS CodeCommit.
- View files in the AWS CodeCommit repository.
- Share the AWS CodeCommit repository with your team.
Step 0: Setup Required for Access to AWS CodeCommit

Before you can migrate a repository to AWS CodeCommit, you must create and configure an IAM user for AWS CodeCommit and configure your local computer for access. You should also install the AWS CLI to manage AWS CodeCommit. Although you can perform most AWS CodeCommit tasks without it, the AWS CLI offers flexibility when working with Git at the command line or terminal.

If you are already set up for AWS CodeCommit, you can skip ahead to Step 1: Create an AWS CodeCommit Repository (p. 214).

To create and configure an IAM user for accessing AWS CodeCommit

2. Create an IAM user, or use an existing one, in your AWS account. Make sure you have an access key ID and a secret access key associated with that IAM user. For more information, see Creating an IAM User in Your AWS Account.

   Note
   AWS CodeCommit requires AWS Key Management Service. If you are using an existing IAM user, make sure there are no policies attached to the user that expressly deny the AWS KMS actions required by AWS CodeCommit. For more information, see AWS KMS and Encryption (p. 288).
3. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
4. In the IAM console, in the navigation pane, choose Users, and then choose the IAM user you want to configure for AWS CodeCommit access.
5. On the Permissions tab, choose Add Permissions.
6. In Grant permissions, choose Attach existing policies directly.
7. Select AWSCodeCommitFullAccess from the list of policies, or another managed policy for AWS CodeCommit access. For more information about managed policies for AWS CodeCommit, see AWS Managed (Predefined) Policies for AWS CodeCommit (p. 258).

   After you have selected the policy you want to attach, choose Next: Review to review the list of policies that will be attached to the IAM user. If the list is correct, choose Add permissions.

   For more information about AWS CodeCommit managed policies and sharing access to repositories with other groups and users, see Share a Repository (p. 80) and Authentication and Access Control for AWS CodeCommit (p. 250).

To install and configure the AWS CLI

1. On your local machine, download and install the AWS CLI. This is a prerequisite for interacting with AWS CodeCommit from the command line. For more information, see Getting Set Up with the AWS Command Line Interface.
AWS CodeCommit User Guide
Step 0: Setup Required for Access to AWS CodeCommit

Note
AWS CodeCommit works only with AWS CLI versions 1.7.38 and later. To determine which version of the AWS CLI you have installed, run the `aws --version` command.

To upgrade an older version of the AWS CLI to the latest version, see Installing the AWS Command Line Interface.

2. Run this command to verify the AWS CodeCommit commands for the AWS CLI are installed:

   ```bash
   aws codecommit help
   ```

   This command should return a list of AWS CodeCommit commands.

3. Configure the AWS CLI with the `configure` command, as follows:

   ```bash
   aws configure
   ```

   When prompted, specify the AWS access key and AWS secret access key of the IAM user you will use with AWS CodeCommit. Also, be sure to specify the region where the repository exists, such as `us-east-2`. When prompted for the default output format, specify `json`. For example:

   ```
   AWS Access Key ID [None]: Type your target AWS access key ID here, and then press Enter
   AWS Secret Access Key [None]: Type your target AWS secret access key here, and then press Enter
   Default region name [None]: Type a supported region for AWS CodeCommit here, and then press Enter
   Default output format [None]: Type `json` here, and then press Enter
   ```

   To connect to a repository or a resource in another region, you must re-configure the AWS CLI with the default region name for that region. Supported default region names for AWS CodeCommit include:
   - `us-east-2`
   - `us-east-1`
   - `eu-west-1`
   - `us-west-2`
   - `ap-northeast-1`
   - `ap-southeast-1`
   - `ap-southeast-2`
   - `eu-central-1`
   - `ap-northeast-2`
   - `sa-east-1`
   - `us-west-1`
   - `eu-west-2`
   - `ap-south-1`
   - `ca-central-1`

   For more information about AWS CodeCommit and regions, see Regions and Git Connection Endpoints (p. 278). For more information about IAM, access keys, and secret keys, see How Do I Get Credentials? and Managing Access Keys for IAM Users.

Next, you must install Git.
Step 1: Create an AWS CodeCommit Repository

In this section, you will use the AWS CodeCommit console to create the AWS CodeCommit repository you will use for the rest of this tutorial. To use the AWS CLI to create the repository, see Use the AWS CLI to Create an AWS CodeCommit Repository (p. 77).

2. In the region selector, choose the region where you will create the repository. For more information, see Regions and Git Connection Endpoints (p. 278).

AWS CodeCommit supports both HTTPS and SSH authentication. To complete setup, you must configure either Git credentials for AWS CodeCommit (HTTPS, recommended for most users), an SSH key pair to use when accessing AWS CodeCommit (SSH), or the credential helper included in the AWS CLI (HTTPS).

- For Git credentials on all supported operating systems, see Step 3: Create Git Credentials for HTTPS Connections to AWS CodeCommit (p. 7).
- For SSH on Linux, macOS, or Unix, see SSH and Linux, macOS, or Unix: Set Up the Public and Private Keys for Git and AWS CodeCommit (p. 27).
- For SSH on Windows, see SSH and Windows: Set Up the Public and Private Keys for Git and AWS CodeCommit (p. 32).
- For the credential helper on Linux, macOS, or Unix, see Set Up the Credential Helper (Linux, macOS, or Unix) (p. 37).
- For the credential helper on Windows, see Set Up the Credential Helper (Windows) (p. 42).

AWS CodeCommit supports Git versions 1.7.9 and later. To install Git, we recommend websites such as Git Downloads.

**Note**

Git is an evolving, regularly updated platform. Occasionally, a feature change might affect the way it works with AWS CodeCommit. If you encounter issues with a specific version of Git and AWS CodeCommit, review the information in Troubleshooting (p. 235).

- For Linux, macOS, or Unix:

  To work with files, commits, and other information in AWS CodeCommit repositories, you must install Git on your local machine. AWS CodeCommit supports Git versions 1.7.9 and later.

  To install Git, we recommend websites such as Git Downloads.

  **Note**

  Git is an evolving, regularly updated platform. Occasionally, a feature change might affect the way it works with AWS CodeCommit. If you encounter issues with a specific version of Git and AWS CodeCommit, review the information in Troubleshooting (p. 235).

- For Windows:

  To work with files, commits, and other information in AWS CodeCommit repositories, you must install Git on your local machine. AWS CodeCommit supports Git versions 1.7.9 and later.

  To install Git, we recommend websites such as Git for Windows. If you use this link to install Git, you can accept all of the installation default settings except for the following:

  - When prompted during the Adjusting your PATH environment step, select the Use Git from the Windows Command Prompt option.
  - (Optional) If you intend to use HTTPS with the credential helper that is included in the AWS CLI instead of configuring Git credentials for AWS CodeCommit, on the Configuring extra options page, make sure the Enable Git Credential Manager option is cleared. The Git Credential Manager is only compatible with AWS CodeCommit if IAM users configure Git credentials. For more information, see For HTTPS Users Using Git Credentials (p. 5) and Git for Windows: I Installed Git for Windows, but I Am Denied Access to My Repository (403) (p. 242).

  **Note**

  Git is an evolving, regularly updated platform. Occasionally, a feature change might affect the way it works with AWS CodeCommit. If you encounter issues with a specific version of Git and AWS CodeCommit, review the information in Troubleshooting (p. 235).
3. On the Dashboard page, choose Create repository. (If a welcome page appears instead of the Dashboard page, choose Get Started Now.)

4. On the Create repository page, in Repository name, type a name for the repository.

   Note
   This name must be unique in the region for your AWS account.

5. (Optional) In the Description box, type a description for the repository. This can help you and other users identify the purpose of the repository.

   Note
   The description field accepts all HTML characters and all valid Unicode characters. If you are an application developer using the GetRepository or BatchGetRepositories APIs and plan to display the repository description field in a web browser, see the AWS CodeCommit API Reference.

6. Choose Create repository.

7. In Configure email notifications, configure notifications so that repository users receive emails about important repository events. This step is optional, but recommended. You can choose the event types (for example, comments on code) and whether to use an existing Amazon SNS topic or create one specifically for this purpose. You can choose to skip this step and configure notifications at a later time. For more information, see Configuring Notifications for Events in an AWS CodeCommit Repository (p. 84).

After it is created, the repository will appear in the list of repositories in your dashboard. In the URL column, choose the copy icon, and then choose the protocol (SSH or HTTPS) you will use to connect to AWS CodeCommit. Copy the URL.

For example, if you named your repository MyClonedRepository and you are using Git credentials with HTTPS in the US West (Oregon) region, the URL would look like the following:

https://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyClonedRepository
Step 2: Clone the Repository and Push to the AWS CodeCommit Repository

In this section, you will clone an existing Git repository to your local computer, creating what is called a local repo. You will then push the contents of the local repo to the AWS CodeCommit repository you created earlier.

1. From the terminal or command prompt on your local computer, run the `git clone` command with the `--mirror` option to clone a bare copy of the remote repository into a new folder named `aws-codecommit-demo`. Note that this is a bare repo meant only for migration, and is not the local repo for interacting with the migrated repository in AWS CodeCommit. You'll want to create that later, after the migration to AWS CodeCommit is complete.

   The following example clones a sample application created for AWS demonstration purposes and hosted on GitHub ([https://github.com/awslabs/aws-demo-php-simple-app.git](https://github.com/awslabs/aws-demo-php-simple-app.git)) to a local repo in a directory named `aws-codecommit-demo`.

   ```bash
   git clone --mirror https://github.com/awslabs/aws-demo-php-simple-app.git aws-codecommit-demo
   ```

2. Change directories to the directory where you made the clone.

   ```bash
   cd aws-codecommit-demo
   ```

3. Run the `git push` command, specifying the URL and name of the destination AWS CodeCommit repository and the `--all` option. (This is the URL you copied in Step 1: Create an AWS CodeCommit Repository (p. 214)).

   For example, if you named your repository `MyClonedRepository` and you are set up to use HTTPS, you would type the following command:

   ```bash
   git push https://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyClonedRepository --all
   ```

   **Note**

   The `--all` option only pushes all branches for the repository. It does not push other references, such as tags. If you want to push tags, wait until the initial push is complete, and then push again, this time using the `--tags` option, for example:

   ```bash
   git push ssh://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyClonedRepository --tags
   ```

   For more information about git push and its options, see Git push. For information about pushing large repositories, especially when pushing all references at once (for example, with the `--mirror` option), see Migrate a Repository in Increments (p. 227).

You can delete the `aws-codecommit-demo` folder and its contents after you have migrated the repository to AWS CodeCommit. To create a local repo with all the correct references for working with the repository in AWS CodeCommit, run the `git clone` command without the `--mirror` option. For example:
Step 3: View Files in AWS CodeCommit

After you have pushed the contents of your directory, you can use the AWS CodeCommit console to quickly view all of the files in that repository.

2. Choose the name of the repository from the list (for example, MyClonedRepository).
3. View the files in the repository for the branches, the clone URLs, the settings, and more.

Step 4: Share the AWS CodeCommit Repository

When you create a repository in AWS CodeCommit, two endpoints are generated: one for HTTPS connections and one for SSH connections. Both provide secure connections over a network. Your users can use either protocol. Both endpoints remain active no matter which protocol you recommend to your users. Before you can share your repository with others, you must create IAM policies that allow access to your repository to other users. Provide those access instructions to your users.

Create a customer managed policy for your repository

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the Dashboard navigation area, choose Policies, and then choose Create Policy.
3. On the Create Policy page, next to Copy an AWS Managed Policy, choose Select.
5. On the Review Policy page, in Policy Name, type a new name for the policy (for example, AWSCodeCommitPowerUser-MyDemoRepo).

In the Policy Document text box, replace the "*" portion of the Resource line with the Amazon Resource Name (ARN) of the AWS CodeCommit repository. For example:

```
"Resource": [
  "arn:aws:codecommit:us-east-2:80398EXAMPLE:MyDemoRepo"
]
```
Step 4: Share the AWS CodeCommit Repository

Tip
To find the ARN for the AWS CodeCommit repository, go to the AWS CodeCommit console and choose the repository name from the list. For more information, see View Repository Details (p. 109).

If you want this policy to apply to more than one repository, add each repository as a resource by specifying its ARN. Include a comma between each resource statement, as shown in the following example:

"Resource": [
  "arn:aws:codecommit:us-east-2:80398EXAMPLE:MyOtherDemoRepo"
]

6. Choose Validate Policy. After it is validated, choose Create Policy.

Tip
Creating a managed policy for a repository does not supply additional permissions required for individual users to set up Git credentials or SSH keys in IAM. You must apply these managed policies to individual IAM users.

• To allow users to use Git credentials to connect to AWS CodeCommit, select the IAMSelfManageServiceSpecificCredentials and IAMReadOnlyAccess managed policies and apply them to your users.

• To allow users to use SSH to connect to AWS CodeCommit, select the IAMUserSSHKeys and IAMReadOnlyAccess managed policies and apply them to your users.

To manage access to your repository, create an IAM group for its users, add IAM users to that group, and then attach the customer managed policy you created in the previous step, as well as any additional policies required for access, such as IAMUserSSHKeys or IAMSelfManageServiceSpecificCredentials.

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.

2. In the Dashboard navigation area, choose Groups, and then choose Create New Group.

3. On the Set Group Name page, in the Group Name box, type a name for the group (for example, MyDemoRepoGroup), and then choose Next Step. Consider including the repository name as part of the group name.

   Note
   This name must be unique across an AWS account.

4. Select the check box next to the customer managed policy you created in the previous section (for example, AWSCodeCommitPowerUser-MyDemoRepo).

   • If your users will use HTTPS and Git credentials to connect to AWS CodeCommit, select the check boxes next to IAMSelfManageServiceSpecificCredentials and IAMReadOnlyAccess, and then choose Next Step.

   • If your users will use SSH to connect to your repository, select the check boxes next to IAMUserSSHKeys and IAMReadOnlyAccess, and then choose Next Step.

5. On the Review page, choose Create Group. The group will be created in IAM with the specified policies already attached. It will appear in the list of groups associated with your AWS account.

6. Choose your group from the list.

7. On the group summary page, choose the Users tab, and then choose Add Users to Group. On the list that shows all users associated with your AWS account, select the check boxes next to the users to whom you want to allow access to the AWS CodeCommit repository, and then choose Add Users.
**Tip**
You can use the Search box to quickly find users by name.

8. When you have added your users, close the IAM console.

After you have created an IAM user that will access AWS CodeCommit using the policy group and policies you configured, send that user the connection information they will use to connect to the repository.

2. In the region selector, choose the region where the repository was created. Repositories are specific to an AWS region. For more information, see Regions and Git Connection Endpoints (p. 278).
3. On the Dashboard page, choose the name of the repository you want to share.
4. On the Code page, choose Clone URL, and then choose the protocol you want your users to use.
5. Copy the displayed URL for the connection protocol your users will use when connecting to your AWS CodeCommit repository.
6. Send your users the connection information along with any other instructions, such as installing the AWS CLI, configuring a profile, or installing Git. Make sure to include the configuration information for the connection protocol (for example, for HTTPS, configuring the credential helper for Git).

---

**Migrate Local or Unversioned Content to AWS CodeCommit**

The procedures in this topic walk you through the process of migrating an existing project or local content on your computer to an AWS CodeCommit repository. As part of this process, you will:

- Complete the initial setup required for AWS CodeCommit.
- Create an AWS CodeCommit repository.
- Place a local folder under Git version control and push the contents of that folder to the AWS CodeCommit repository.
- View files in the AWS CodeCommit repository.
- Share the AWS CodeCommit repository with your team.
Step 0: Setup Required for Access to AWS CodeCommit

Before you can migrate local content to AWS CodeCommit, you must create and configure an IAM user for AWS CodeCommit and configure your local computer for access. You should also install the AWS CLI to manage AWS CodeCommit. Although you can perform most AWS CodeCommit tasks without it, the AWS CLI offers flexibility when working with Git.

If you are already set up for AWS CodeCommit, you can skip ahead to Step 1: Create an AWS CodeCommit Repository (p. 222).

To create and configure an IAM user for accessing AWS CodeCommit

2. Create an IAM user, or use an existing one, in your AWS account. Make sure you have an access key ID and a secret access key associated with that IAM user. For more information, see Creating an IAM User in Your AWS Account.

   Note
   AWS CodeCommit requires AWS Key Management Service. If you are using an existing IAM user, make sure there are no policies attached to the user that expressly deny the AWS KMS actions required by AWS CodeCommit. For more information, see AWS KMS and Encryption (p. 288).
3. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
4. In the IAM console, in the navigation pane, choose Users, and then choose the IAM user you want to configure for AWS CodeCommit access.
5. On the Permissions tab, choose Add Permissions.
6. In Grant permissions, choose Attach existing policies directly.
7. Select AWSCodeCommitFullAccess from the list of policies, or another managed policy for AWS CodeCommit access. For more information about managed policies for AWS CodeCommit, see AWS Managed (Predefined) Policies for AWS CodeCommit (p. 258).

   After you have selected the policy you want to attach, choose Next: Review to review the list of policies that will be attached to the IAM user. If the list is correct, choose Add permissions.

   For more information about AWS CodeCommit managed policies and sharing access to repositories with other groups and users, see Share a Repository (p. 80) and Authentication and Access Control for AWS CodeCommit (p. 250).

To install and configure the AWS CLI

1. On your local machine, download and install the AWS CLI. This is a prerequisite for interacting with AWS CodeCommit from the command line. For more information, see Getting Set Up with the AWS Command Line Interface.
Note
AWS CodeCommit works only with AWS CLI versions 1.7.38 and later. To determine which version of the AWS CLI you have installed, run the `aws --version` command.
To upgrade an older version of the AWS CLI to the latest version, see Installing the AWS Command Line Interface.

2. Run this command to verify the AWS CodeCommit commands for the AWS CLI are installed:

```
aws codecommit help
```

This command should return a list of AWS CodeCommit commands.

3. Configure the AWS CLI with the `configure` command, as follows:

```
aws configure
```

When prompted, specify the AWS access key and AWS secret access key of the IAM user you will use with AWS CodeCommit. Also, be sure to specify the region where the repository exists, such as `us-east-2`. When prompted for the default output format, specify `json`. For example:

```
AWS Access Key ID [None]: Type your target AWS access key ID here, and then press Enter
AWS Secret Access Key [None]: Type your target AWS secret access key here, and then press Enter
Default region name [None]: Type a supported region for AWS CodeCommit here, and then press Enter
Default output format [None]: Type json here, and then press Enter
```

To connect to a repository or a resource in another region, you must re-configure the AWS CLI with the default region name for that region. Supported default region names for AWS CodeCommit include:

- `us-east-2`
- `us-east-1`
- `eu-west-1`
- `us-west-2`
- `ap-northeast-1`
- `ap-southeast-1`
- `ap-southeast-2`
- `eu-central-1`
- `ap-northeast-2`
- `sa-east-1`
- `us-west-1`
- `eu-west-2`
- `ap-south-1`
- `ca-central-1`

For more information about AWS CodeCommit and regions, see Regions and Git Connection Endpoints (p. 278). For more information about IAM, access keys, and secret keys, see How Do I Get Credentials? and Managing Access Keys for IAM Users.

Next, you must install Git.
Step 1: Create an AWS CodeCommit Repository

In this section, you will use the AWS CodeCommit console to create the AWS CodeCommit repository you will use for the rest of this tutorial. To use the AWS CLI to create the repository, see Use the AWS CLI to Create an AWS CodeCommit Repository (p. 77).

2. In the region selector, choose the region where you will create the repository. For more information, see Regions and Git Connection Endpoints (p. 278).

AWS CodeCommit supports both HTTPS and SSH authentication. To complete setup, you must configure either Git credentials for AWS CodeCommit (HTTPS, recommended for most users), an SSH key pair (SSH) to use when accessing AWS CodeCommit, or the credential helper included in the AWS CLI.

- For Git credentials on all supported operating systems, see Step 3: Create Git Credentials for HTTPS Connections to AWS CodeCommit (p. 7).
- For SSH on Linux, macOS, or Unix, see SSH and Linux, macOS, or Unix: Set Up the Public and Private Keys for Git and AWS CodeCommit (p. 27).
- For SSH on Windows, see SSH and Windows: Set Up the Public and Private Keys for Git and AWS CodeCommit (p. 32).
- For the credential helper on Linux, macOS, or Unix, see Set Up the Credential Helper (Linux, macOS, or Unix) (p. 37).
- For the credential helper on Windows, see Set Up the Credential Helper (Windows) (p. 42).
3. On the **Dashboard** page, choose **Create repository**. (If a welcome page appears instead of the **Dashboard** page, choose **Get Started Now**.)

4. On the **Create repository** page, in **Repository name**, type a name for the repository.

   **Note**
   This name must be unique in the region for your AWS account.

5. (Optional) In the **Description** box, type a description for the repository. This can help you and other users identify the purpose of the repository.

   **Note**
   The description field accepts all HTML characters and all valid Unicode characters. If you are an application developer using the `GetRepository` or `BatchGetRepositories` APIs and plan to display the repository description field in a web browser, see the [AWS CodeCommit API Reference](https://docs.aws.amazon.com/CodeCommit/latest/APIReference/).

6. Choose **Create repository**.

7. In **Configure email notifications**, configure notifications so that repository users receive emails about important repository events. This step is optional, but recommended. You can choose the event types (for example, comments on code) and whether to use an existing Amazon SNS topic or create one specifically for this purpose. You can choose to skip this step and configure notifications at a later time. For more information, see [Configuring Notifications for Events in an AWS CodeCommit Repository](https://docs.aws.amazon.com/CodeCommit/latest/UserGuide/configure-notifications.html) (p. 84).

After it is created, the repository will appear in the list of repositories in your dashboard. In the URL column, choose the copy icon, and then choose the protocol (HTTPS or SSH) you will use to connect to AWS CodeCommit. Copy the URL.

For example, if you named your repository *MyFirstRepo* and you are using HTTPS, the URL would look like the following:

```
https://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyFirstRepo
```
Step 2: Migrate Local Content to the AWS CodeCommit Repository

Now that you have an AWS CodeCommit repository, you can choose a directory on your local computer to convert into a local Git repository. The `git init` command can be used to either convert existing, unversioned content to a Git repository or, if you do not yet have files or content, to initialize a new, empty repository.

1. From the terminal or command line on your local computer, change directories to the directory you want to use as the source for your repository.
2. Run the `git init` command to initialize Git version control in the directory. This will create a `.git` subdirectory in the root of the directory that enables version control tracking. The `.git` folder also contains all of the required metadata for the repository.

```
git init
```

3. Add the files you want to add to version control. In this tutorial, you will run the `git add` command with the `.` specifier to add all of the files in this directory. For other options, consult your Git documentation.

```
git add .
```

4. Create a commit for the added files with a commit message.

```
git commit -m "Initial commit"
```

5. Run the `git push` command, specifying the URL and name of the destination AWS CodeCommit repository and the `--all` option. (This is the URL you copied in Step 1: Create an AWS CodeCommit Repository (p. 222).)

For example, if you named your repository `MyFirstRepo` and you are set up to use HTTPS, you would type the following command:

```
git push https://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyFirstRepo --all
```

Step 3: View Files in AWS CodeCommit

After you have pushed the contents of your directory, you can use the AWS CodeCommit console to quickly view all of the files in the repository.

2. Choose the name of the repository from the list (for example, `MyFirstRepository`).
3. View the files in the repository for the branches, the clone URLs, the settings, and more.
Step 4: Share the AWS CodeCommit Repository

When you create a repository in AWS CodeCommit, two endpoints are generated: one for HTTPS connections and one for SSH connections. Both provide secure connections over a network. Your users can use either protocol. Both endpoints remain active no matter which protocol you recommend to your users. Before you can share your repository with others, you must create IAM policies that allow access to your repository to other users. Provide those access instructions to your users.

Create a customer managed policy for your repository

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the Dashboard navigation area, choose Policies, and then choose Create Policy.
3. On the Create Policy page, next to Copy an AWS Managed Policy, choose Select.
5. On the Review Policy page, in Policy Name, type a new name for the policy (for example, AWSCodeCommitPowerUser-MyDemoRepo).

In the Policy Document text box, replace the "*" portion of the Resource line with the Amazon Resource Name (ARN) of the AWS CodeCommit repository. For example:

```
"Resource": [
  "arn:aws:codecommit:us-east-2:80398EXAMPLE:MyDemoRepo"
]
```

**Tip**

To find the ARN for the AWS CodeCommit repository, go to the AWS CodeCommit console and choose the repository name from the list. For more information, see View Repository Details (p. 109).

If you want this policy to apply to more than one repository, add each repository as a resource by specifying its ARN. Include a comma between each resource statement, as shown in the following example:

```
"Resource": [
  "arn:aws:codecommit:us-east-2:80398EXAMPLE:MyOtherDemoRepo"
]
```

6. Choose Validate Policy. After it is validated, choose Create Policy.
**Step 4: Share the AWS CodeCommit Repository**

**Tip**
Creating a managed policy for a repository does not supply additional permissions required for individual users to set up Git credentials or SSH keys in IAM. You must apply these managed policies to individual IAM users.

- To allow users to use Git credentials to connect to AWS CodeCommit, select the `IAMSelfManageServiceSpecificCredentials` and `IAMReadOnlyAccess` managed policies and apply them to your users.
- To allow users to use SSH to connect to AWS CodeCommit, select the `IAMUserSSHKeys` and `IAMReadOnlyAccess` managed policies and apply them to your users.

To manage access to your repository, create an IAM group for its users, add IAM users to that group, and then attach the customer managed policy you created in the previous step, as well as any additional policies required for access, such as `IAMSelfManageServiceSpecificCredentials` or `IAMUserSSHKeys`.

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the Dashboard navigation area, choose Groups, and then choose Create New Group.
3. On the Set Group Name page, in the Group Name box, type a name for the group (for example, `MyDemoRepoGroup`), and then choose Next Step. Consider including the repository name as part of the group name.

   **Note**
   This name must be unique across an AWS account.

4. Select the check box next to the customer managed policy you created in the previous section (for example, `AWSCodeCommitPowerUser-MyDemoRepo`).
   - If your users will use HTTPS and Git credentials to connect to AWS CodeCommit, select the check boxes next to `IAMSelfManageServiceSpecificCredentials` and `IAMReadOnlyAccess`, and then choose Next Step.
   - If your users will use SSH to connect to your repository, select the check boxes next to `IAMUserSSHKeys` and `IAMReadOnlyAccess`, and then choose Next Step.
5. On the Review page, choose Create Group. The group will be created in IAM with the specified policies already attached. It will appear in the list of groups associated with your AWS account.
6. Choose your group from the list.
7. On the group summary page, choose the Users tab, and then choose Add Users to Group. On the list that shows all users associated with your AWS account, select the check boxes next to the users to whom you want to allow access to the AWS CodeCommit repository, and then choose Add Users.

   **Tip**
   You can use the Search box to quickly find users by name.

8. When you have added your users, close the IAM console.

   After you have created an IAM user that will access AWS CodeCommit using the policy group and policies you configured, send that user the connection information they will use to connect to the repository.

2. In the region selector, choose the region where the repository was created. Repositories are specific to an AWS region. For more information, see Regions and Git Connection Endpoints (p. 278).
3. On the Dashboard page, choose the name of the repository you want to share.
4. On the Code page, choose Clone URL, and then choose the protocol you want your users to use.
5. Copy the displayed URL for the connection protocol your users will use when connecting to your AWS CodeCommit repository.
6. Send your users the connection information along with any other instructions, such as installing the AWS CLI, configuring a profile, or installing Git. Make sure to include the configuration information for the connection protocol (for example, for HTTPS, configuring the credential helper for Git).

Migrate a Repository Incrementally

When migrating to AWS CodeCommit, consider pushing your repository in increments or chunks to reduce the chances an intermittent network issue or degraded network performance will cause the entire push to fail. By using incremental pushes with a script like the following, you can restart the migration and push only those commits that did not succeed on the earlier attempt.

The procedures in this topic show you how to create and run a script that will migrate your repository in increments and repush only those increments that did not succeed until the migration is complete.

These instructions assume you have already completed the steps in Setting Up (p. 4) and Create a Repository (p. 76).

Topics

• Step 0: Determine Whether to Migrate Incrementally (p. 227)
• Step 1: Install Prerequisites and Add the AWS CodeCommit Repository as a Remote (p. 228)
• Step 2: Create the Script to Use for Migrating Incrementally (p. 229)
• Step 3: Run the Script and Migrate Incrementally to AWS CodeCommit (p. 229)
• Appendix: Sample Script incremental-repo-migration.py (p. 230)

Step 0: Determine Whether to Migrate Incrementally

There are several factors to consider to determine the overall size of your repository and whether to migrate incrementally. The most obvious is the overall size of the artifacts in the repository. Factors such as the accumulated history of the repository can also contribute to size. A repository with years of history and branches can be very large, even though the individual assets are not. There are a number of strategies you can pursue to make migrating these repositories simpler and more efficient, such as using a shallow clone strategy when cloning a repository with a long history of development, or turning off delta compression for large binary files. You can research options by consulting your Git documentation, or you can choose to set up and configure incremental pushes for migrating your repository using the sample script included in this topic, incremental-repo-migration.py.

You might want to configure incremental pushes if one or more of the following conditions is true:

• The repository you want to migrate has more than five years of history.
• Your internet connection is subject to intermittent outages, dropped packets, slow response, or other interruptions in service.
• The overall size of the repository is larger than 2 GB and you intend to migrate the entire repository.
• The repository contains large artifacts or binaries that do not compress well, such as large image files with more than five tracked versions.
• You have previously attempted a migration to AWS CodeCommit and received an "Internal Service Error” message.

Even if none of the above conditions are true, you can still choose to push incrementally.
Step 1: Install Prerequisites and Add the AWS CodeCommit Repository as a Remote

You can create your own custom script, which will have its own prerequisites. If you choose to use the sample included in this topic, you must first install its prerequisites, as well as clone the repository to your local computer and add the AWS CodeCommit repository as a remote for the repository you want to migrate.

Set up to run incremental-repo-migration.py

1. On your local computer, install Python 2.6 or later, if it is not already installed. For more information and the latest versions, see the Python website.
2. On the same computer, install GitPython, which is a Python library used to interact with Git repositories, if it is not already installed. For more information, see the GitPython documentation.
3. Use the `git clone --mirror` command to clone the repository you want to migrate to your local computer. From the terminal (Linux, macOS, or Unix) or the command prompt (Windows), use the `git clone --mirror` command to create a local repo for the repository, including the directory where you want to create the local repo. For example, to clone a Git repository named `MyMigrationRepo` with a URL of `https://example.com/my-repo/` to a directory named `my-repo`:

```
git clone --mirror https://example.com/my-repo/MyMigrationRepo.git my-repo
```

You should see output similar to the following, which indicates the repository has been cloned into a bare local repo named `my-repo`:

```
Cloning into bare repository 'my-repo'...
remote: Counting objects: 20, done.
remote: Compressing objects: 100% (17/17), done.
remote: Total 20 (delta 5), reused 15 (delta 3)
Unpacking objects: 100% (20/20), done.
Checking connectivity... done.
```

4. Change directories to the local repo for the repository you just cloned (for example, `my-repo`). From that directory, use the `git remote add` command to add the AWS CodeCommit repository as a remote repository for the local repo.

   **Note**
   
   When pushing large repositories, consider using SSH instead of HTTPS. When pushing a large change, a large number of changes, or a large repository, long-running HTTPS connections are often terminated prematurely due to networking issues or firewall settings. For more information about setting up AWS CodeCommit for SSH, see For SSH Connections on Linux, macOS, or Unix (p. 26) or For SSH Connections on Windows (p. 30).

   For example, to add the SSH endpoint for an AWS CodeCommit repository named `MyDestinationRepo` as a remote repository for the remote named `codecommit`, use the following command:

```
git remote add codecommit ssh://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDestinationRepo
```

   **Tip**
   
   Because this is a clone, the default remote name (`origin`) will already be in use. You must use another remote name. Although the example uses `codecommit`, you can use any name you want. Use the `git remote show` command to review the list of remotes set for your local repo.
Step 2: Create the Script to Use for Migrating Incrementally

These steps assume you will use the `incremental-repo-migration.py` sample script.

1. Open a text editor and paste the contents of the sample script (p. 230) into an empty document.
2. Save the document in a documents directory (not the working directory of your local repo) and name it `incremental-repo-migration.py`. Make sure the directory you choose is one configured in your local environment or path variables, so you can run the Python script from a command line or terminal.

Step 3: Run the Script and Migrate Incrementally to AWS CodeCommit

Now that you have created your `incremental-repo-migration.py` script, you can use it to incrementally migrate a local repo to an AWS CodeCommit repository. By default, the script pushes commits in batches of 1,000 commits and attempts to use the Git settings for the directory from which it is run as the settings for the local repo and remote repository. You can use the options included in `incremental-repo-migration.py` to configure other settings, if necessary.

1. From the terminal or command prompt, change directories to the local repo you want to migrate.
2. From that directory, type the following command:

   ```
   python incremental-repo-migration.py
   ```

3. The script runs and shows progress at the terminal or command prompt. Some large repositories will be slow to show progress. The script will stop if a single push fails three times. You can then rerun the script, and it will start from the batch that failed. You can rerun the script until all pushes succeed and the migration is complete.

Tip
You can run `incremental-repo-migration.py` from any directory as long as you use the `-l` and `-r` options to specify the local and remote settings to use. For example, to use the script from any directory to migrate a local repo located at `/tmp/my-repo` to a remote nicknamed `codecommit`:

```
python incremental-repo-migration.py -l "/tmp/my-repo" -r "codecommit"
```
You might also want to use the \( -b \) option to change the default batch size used when pushing incrementally. For example, if you are regularly pushing a repository with very large binary files that change often and are working from a location that has restricted network bandwidth, you might want to use the \( -b \) option to change the batch size to 500 instead of 1,000. For example:

```bash
python incremental-repo-migration.py -b 500
```

This will push the local repo incrementally in batches of 500 commits. If you decide to change the batch size again when migrating the repository (for example, if you decide to decrease the batch size after an unsuccessful attempt), remember to use the \( -c \) option to remove the batch tags before resetting the batch size with \( -b \):

```bash
python incremental-repo-migration.py -c
python incremental-repo-migration.py -b 250
```

**Important**
Do not use the \( -c \) option if you want to rerun the script after a failure. The \( -c \) option removes the tags used to batch the commits. Use the \( -c \) option only if you want to change the batch size and start again, or if you decide you no longer want to use the script.

**Appendix: Sample Script**

For your convenience, we have developed a sample Python script, `incremental-repo-migration.py`, for pushing a repository incrementally. This script is an open source code sample and provided as-is.

```python
#!/usr/bin/env python
import os
import sys
from optparse import OptionParser
from git import Repo, TagReference, RemoteProgress, GitCommandError

class PushProgressPrinter(RemoteProgress):
    def update(self, op_code, cur_count, max_count=None, message=' '):
        op_id = op_code & self.OP_MASK
        stage_id = op_code & self.STAGE_MASK
        if op_id == self.WRITING and stage_id == self.BEGIN:
            print("Objects: %d" % max_count)

class RepositoryMigration:
    MAX_COMMITS_TOLERANCE_PERCENT = 0.05
    PUSH_RETRY_LIMIT = 3
    MIGRATION_TAG_PREFIX = "codecommit_migration_"

    def migrate_repository_in_parts(self, repo_dir, remote_name, commit_batch_size, clean):
        self.next_tag_number = 0
        self.migration_tags = []
```

API Version 2015-04-13
230
self.walked_commits = set()
self.local_repo = Repo(repo_dir)
self.remote_name = remote_name
self.max_commits_per_push = commit_batch_size
self.max_commits_tolerance = self.max_commits_per_push * self.MAX_COMMITS_TOLERANCE_PERCENT

try:
    self.remote_repo = self.local_repo.remote(remote_name)
    self.get_remote_migration_tags()
except (ValueError, GitCommandError):
    print("Could not contact the remote repository. The most common reasons for this error are that the name of the remote repository is incorrect, or that you do not have permissions to interact with that remote repository.")
    sys.exit(1)

if clean:
    self.clean_up(clean_up_remote=True)
    return
self.clean_up()

print("Analyzing repository")
head_commit = self.local_repo.head.commit
sys.setrecursionlimit(max(sys.getrecursionlimit(), head_commit.count()))

# tag commits on default branch
leftover_commits = self.migrate_commit(head_commit)
self.tag_commits([commit for (commit, commit_count) in leftover_commits])

# tag commits on each branch
for branch in self.local_repo.heads:
    leftover_commits = self.migrate_commit(branch.commit)
    self.tag_commits([commit for (commit, commit_count) in leftover_commits])

# push the tags
self.push_migration_tags()

# push all branch references
for branch in self.local_repo.heads:
    print("Pushing branch %s" % branch.name)
    self.do_push_with_retries(ref=branch.name)

# push all tags
print("Pushing tags")
self.do_push_with_retries(push_tags=True)
self.get_remote_migration_tags()
self.clean_up(clean_up_remote=True)

print("Migration to CodeCommit was successful")

def migrate_commit(self, commit):
    if commit in self.walked_commits:
        return []

    pending_ancestor_pushes = []
    commit_count = 1

    if len(commit.parents) > 1:
        # This is a merge commit
        # Ensure that all parents are pushed first
        for parent_commit in commit.parents:
            pending_ancestor_pushes.extend(self.migrate_commit(parent_commit))
    elif len(commit.parents) == 1:
        # Split linear history into individual pushes
        # ...
incremental-repo-migration.py

```python
next_ancestor, commits_to_next_ancestor = 
    self.find_next_ancestor_for_push(commit.parents[0])
commit_count += commits_to_next_ancestor
pending_ancestor_pushes.extend(self.migrate_commit(next_ancestor))

self.walked_commits.add(commit)
return self.stage_push(commit, commit_count, pending_ancestor_pushes)

def find_next_ancestor_for_push(self, commit):
    commit_count = 0
    # Traverse linear history until we reach our commit limit, a merge commit, or an
    # initial commit
    while len(commit.parents) == 1 and commit_count < self.max_commits_per_push and
    commit not in self.walked_commits:
        commit_count += 1
        self.walked_commits.add(commit)
        commit = commit.parents[0]
    return commit, commit_count

def stage_push(self, commit, commit_count, pending_ancestor_pushes):
    # Determine whether we can roll up pending ancestor pushes into this push
    combined_commit_count = commit_count + sum(ancestor_commit_count for (ancestor,
    ancestor_commit_count) in pending_ancestor_pushes)
    if combined_commit_count < self.max_commits_per_push:
        # don't push anything, roll up all pending ancestor pushes into this pending
        push
        return [(commit, combined_commit_count)]
    if combined_commit_count <= (self.max_commits_per_push +
    self.max_commits_tolerance):
        # roll up everything into this commit and push
        self.tag_commits([commit])
        return []
    if commit_count >= self.max_commits_per_push:
        # need to push each pending ancestor and this commit
        self.tag_commits([ancestor for (ancestor, ancestor_commit_count) in
        pending_ancestor_pushes])
        self.tag_commits([commit])
        return []
    # push each pending ancestor, but roll up this commit
    self.tag_commits([ancestor for (ancestor, ancestor_commit_count) in
    pending_ancestor_pushes])
    return [(commit, commit_count)]

def tag_commits(self, commits):
    for commit in commits:
        self.next_tag_number += 1
        tag_name = self.MIGRATION_TAG_PREFIX + str(self.next_tag_number)
        if tag_name not in self.remote_migration_tags:
            tag = self.local_repo.create_tag(tag_name, ref=commit)
            self.migration_tags.append(tag)
        elif self.remote_migration_tags[tag_name] != str(commit):
            print("Migration tags on the remote do not match the local tags. Most
            likely your batch size has changed since the last time you ran this script. Please run
            this script with the --clean option, and try again.")
            sys.exit(1)

def push_migration_tags(self):
    print("Will attempt to push %d tags" % len(self.migration_tags))

```

self.migration_tags.sort(key=lambda tag:
    int(tag.name.replace(self.MIGRATION_TAG_PREFIX, "")))
for tag in self.migration_tags:
    print("Pushing tag %s (out of %d tags), commit %s" %
        (tag.name, self.next_tag_number, str(tag.commit)))
    self.do_push_with_retries(ref=tag.name)

def do_push_with_retries(self, ref=None, push_tags=False):
    for i in range(0, self.PUSH_RETRY_LIMIT):
        if i == 0:
            progress_printer = PushProgressPrinter()
        else:
            progress_printer = None
        try:
            if push_tags:
                infos = self.remote_repo.push(tags=True, progress=progress_printer)
            elif ref is not None:
                infos = self.remote_repo.push(refspec=ref, progress=progress_printer)
            else:
                infos = self.remote_repo.push(progress=progress_printer)
            success = True
        except GitCommandError as err:
            print(err)
            if push_tags:
                print("Pushing all tags failed after %d attempts" %
                    (self.PUSH_RETRY_LIMIT))
            elif ref is not None:
                print("Pushing %s failed after %d attempts" %
                    (ref, self.PUSH_RETRY_LIMIT))
            else:
                print("Pushing all branches failed after %d attempts" %
                    (self.PUSH_RETRY_LIMIT))
                sys.exit(1)
            break
        else:
            for info in infos:
                if info.flags & info.UP_TO_DATE or info.flags & info.NEW_TAG or
                    info.flags & info.NEW_HEAD:
                    continue
                success = False
                print(info.summary)
            if success:
                return
except GitCommandError as err:
    print(err)
    if push_tags:
        print("Pushing all tags failed after %d attempts" %
            (self.PUSH_RETRY_LIMIT))
    elif ref is not None:
        print("Pushing %s failed after %d attempts" %
            (ref, self.PUSH_RETRY_LIMIT))
        print("For more information about the cause of this error, run the following
            command from the local repo: 'git push %s %s'" %
            (self.remote_name, ref))
    else:
        print("Pushing all branches failed after %d attempts" %
            (self.PUSH_RETRY_LIMIT))
        sys.exit(1)

def get_remote_migration_tags(self):
    remote_tags_output = self.local_repo.git.ls_remote(self.remote_name,
        tags=True).split('\n')
    self.remote_migration_tags = dict((tag[1].replace("refs/tags/",""),
        tag[0]) for tag in remote_tags_output if self.MIGRATION_TAG_PREFIX in tag)

def clean_up(self, clean_up_remote=False):
    tags = [tag for tag in self.local_repo.tags if
        tag.name.startswith(self.MIGRATION_TAG_PREFIX)]
    # delete the local tags
    TagReference.delete(self.local_repo, *tags)
    # delete the remote tags
    if clean_up_remote:
        tags_to_delete = [":" + tag_name for tag_name in self.remote_migration_tags]
        self.remote_repo.push(refspec=tags_to_delete)
incremental-repo-migration.py

```python
parser = OptionParser()
parser.add_option("-l", "--local",
    action="store", dest="localrepo", default=os.getcwd(),
    help="The path to the local repo. If this option is not specified, the
    script will attempt to use current directory by default. If it is not a local git repo,
    the script will fail.")
parser.add_option("-r", "--remote",
    action="store", dest="remoterepo", default="codecommit",
    help="The name of the remote repository to be used as the push or
    migration destination. The remote must already be set in the local repo ('git remote
    add ...'). If this option is not specified, the script will use 'codecommit' by default.")
parser.add_option("-b", "--batch",
    action="store", dest="batchsize", default="1000",
    help="Specifies the commit batch size for pushes. If not explicitly set,
    the default is 1,000 commits.")
parser.add_option("-c", "--clean",
    action="store_true", dest="clean", default=False,
    help="Remove the temporary tags created by migration from both the local
    repo and the remote repository. This option will not do any migration work, just cleanup.
    Cleanup is done automatically at the end of a successful migration, but not after a
    failure so that when you re-run the script, the tags from the prior run can be used to
    identify commit batches that were not pushed successfully.")

(options, args) = parser.parse_args()
migration = RepositoryMigration()
migration.migrate_repository_in_parts(options.localrepo, options.remoterepo,
    int(options.batchsize), options.clean)
```

API Version 2015-04-13
234
Troubleshooting AWS CodeCommit

The following information might help you troubleshoot common issues in AWS CodeCommit.

Topics
- Troubleshooting Git Credentials and HTTPS Connections to AWS CodeCommit (p. 235)
- Troubleshooting SSH Connections to AWS CodeCommit (p. 236)
- Troubleshooting the Credential Helper and HTTPS Connections to AWS CodeCommit (p. 240)
- Troubleshooting Git Clients and AWS CodeCommit (p. 243)
- Troubleshooting Access Errors and AWS CodeCommit (p. 245)
- Troubleshooting Configuration Errors and AWS CodeCommit (p. 246)
- Troubleshooting Console Errors and AWS CodeCommit (p. 247)
- Troubleshooting Triggers and AWS CodeCommit (p. 248)
- Turn on Debugging (p. 248)

Troubleshooting Git Credentials and HTTPS Connections to AWS CodeCommit

The following information might help you troubleshoot common issues when using Git credentials and HTTPS to connect to AWS CodeCommit repositories.

Topics
- Git Credentials for AWS CodeCommit: I Keep Seeing a Prompt for Credentials When Connecting to My AWS CodeCommit Repository at the Terminal or Command Line (p. 235)
- Git Credentials for AWS CodeCommit: I Set Up Git Credentials, But My System Is Not Using Them As I Expected (p. 236)

Git Credentials for AWS CodeCommit: I Keep Seeing a Prompt for Credentials When Connecting to My AWS CodeCommit Repository at the Terminal or Command Line

Problem: When you try to push, pull, or otherwise interact with an AWS CodeCommit repository from the terminal or command line, you are prompted to provide a user name and password, and you must supply the Git credentials for your IAM user.

Possible fixes: The most common causes for this error are that your local computer is running an operating system that does not support credential management, or it does not have a credential management utility installed, or the Git credentials for your IAM user have not been saved to one of these credential management systems. Depending on your operating system and local environment, you might need to install a credential manager, configure the credential manager that is included in your operating system, or customize your local environment to use credential storage. For example, if your computer is running macOS, you can use the Keychain Access utility to store your credentials. If your computer is running Windows, you can use the Git Credential Manager that is installed with Git for Windows. For more information, see For HTTPS Users Using Git Credentials (p. 5) and Credential Storage in the Git documentation.
Git Credentials for AWS CodeCommit: I Set Up Git Credentials, But My System Is Not Using Them As I Expected

**Problem:** When you try to use AWS CodeCommit with a Git client, the client does not appear to use the Git credentials for your IAM user.

**Possible fixes:** The most common cause for this error is that you previously set up your computer to use the credential helper that is included with the AWS CLI. Check your .gitconfig file for configuration sections similar to the following, and remove them:

```
[credential "https://git-codecommit.*.amazonaws.com"]
  helper = !aws codecommit credential-helper $@
  UseHttpPath = true
```

Save the file, and then open a new command line or terminal session before you attempt to connect again.

You may also have multiple credential helpers or managers set up on your computer, and your system might be defaulting to another configuration. To reset which credential helper is used as the default, you can use the `--system` option instead of `--global` or `--local` when running the `git config` command.

For more information, see For HTTPS Users Using Git Credentials (p. 5) and Credential Storage in the Git documentation.

Troubleshooting SSH Connections to AWS CodeCommit

The following information might help you troubleshoot common issues when using SSH to connect to AWS CodeCommit repositories.

**Topics**
- Access Error: Public Key Is Uploaded Successfully to IAM but Connection Fails on Linux, macOS, or Unix Systems (p. 236)
- Access Error: Public Key Is Uploaded Successfully to IAM and SSH Tested Successfully but Connection Fails on Windows Systems (p. 237)
- Authentication Challenge: Authenticity of Host Can't Be Established When Connecting to an AWS CodeCommit Repository (p. 238)
- IAM Error: 'Invalid format' when attempting to add a public key to IAM (p. 240)
- Git on Windows: Bash Emulator or Command Line Freezes When Attempting to Connect Using SSH (p. 240)

Access Error: Public Key Is Uploaded Successfully to IAM but Connection Fails on Linux, macOS, or Unix Systems

**Problem:** When you try to connect to an SSH endpoint to communicate with an AWS CodeCommit repository, either when testing the connection or cloning a repository, the connection fails or is refused.
Possible fixes: The SSH Key ID assigned to your public key in IAM might not be associated with your connection attempt. You might not have configured a config file (p. 28), you might not have access to the configuration file, another setting might be preventing a successful read of the config file, or you might have provided the ID of the IAM user instead of the key ID.

The SSH Key ID can be found in the IAM console in the profile for your IAM user:

Try testing the connection with the following command:

```bash
ssh Your-SSH-Key-ID@git-codecommit.us-east-2.amazonaws.com
```

If you see a success message after confirming the connection, your SSH Key ID is valid. Edit your config file to associate your connection attempts with your public key in IAM. If you do not want to edit your config file for some reason, you can preface all connection attempts to your repository with your SSH Key ID. For example, if you wanted to clone a repository named `MyDemoRepo` without modifying your config file to associate your connection attempts, you would type the following command:

```bash
git clone ssh://Your-SSH-Key-ID@git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo my-demo-repo
```

For more information, see For SSH Connections on Linux, macOS, or Unix (p. 26).

Access Error: Public Key Is Uploaded Successfully to IAM and SSH Tested Successfully but Connection Fails on Windows Systems

Problem: When you try to use an SSH endpoint to clone or communicate with an AWS CodeCommit repository, an error message appears containing the phrase No supported authentication methods available.

Possible fixes: The most common reason for this error is that you have a Windows system environment variable set that directs Windows to use another program when you attempt to use SSH. For example, you might have set a GIT_SSH variable to point to one of the PuTTY set of tools (plink.exe). This might be a legacy configuration, or it might be necessary for one or more other programs installed on your computer. If you are sure that this environment variable is not needed, you can remove it by opening your system properties and deleting the environment variable.

To work around this issue, open a Bash emulator and then try your SSH connection again, but include `GIT_SSH_COMMAND="SSH"` as a prefix. For example, to clone a repository using SSH:

```bash
GIT_SSH_COMMAND="ssh" git clone ssh://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo my-demo-repo
```

A similar problem might occur if your version of Windows requires that you include the SSH Key ID as part of the connection string when connecting using SSH at the Windows command line. Try your connection again, this time including the SSH Key ID copied from IAM as part of the command. For example:
git clone ssh://Your-SSH-Key-ID@git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo
my-demo-repo

**Authentication Challenge: Authenticity of Host Can't Be Established When Connecting to an AWS CodeCommit Repository**

**Problem:** When you try to use an SSH endpoint to communicate with an AWS CodeCommit repository, a warning message appears containing the phrase *The authenticity of host 'host-name' can't be established.*

**Possible fixes:** Your credentials might not be set up correctly. Follow the instructions in For SSH Connections on Linux, macOS, or Unix (p. 26) or For SSH Connections on Windows (p. 30).

If you have followed those steps and the problem persists, someone might be attempting a man-in-the-middle attack. When you see the following message, type *no*, and press Enter.

Are you sure you want to continue connecting (yes/no)?

Make sure the fingerprint and public key for AWS CodeCommit connections match those documented in the SSH setup topics before you continue with the connection.

**Public fingerprints for AWS CodeCommit**

<table>
<thead>
<tr>
<th>Server</th>
<th>Cryptographic hash type</th>
<th>Fingerprint</th>
</tr>
</thead>
<tbody>
<tr>
<td>git-codecommit.us-east-2.amazonaws.com</td>
<td>SHA256</td>
<td>3lBl2w5xn/NA2Ck6dyeJIRQOWVn7n8UEs56fG6izQ</td>
</tr>
</tbody>
</table>
| git-codecommit.us-east-1.amazonaws.com | MD5                     | a6:9c:7d:bc:35:f5:d4:5f:8b:ba:6f:cb:8f:e7:0f |}
| git-codecommit.us-east-1.amazonaws.com | SHA256                   | eLMY1j0DKA4uvDZ8cI/KgtIayZANwX6t8+8isPtoBoY |
| git-codecommit.us-west-2.amazonaws.com | SHA256                   | 0Pjx9SQpkbPUAHwy58UVIq0IHcyo1fwCpO0u   |
| git-codecommit.eu-west-1.amazonaws.com | SHA256                   | tKjRkOL8dnJyTsBeSdN1sS8F/f0iqI3rrVggqTOP1UyQ  |
| git-codecommit.ap-northeast-1.amazonaws.com | SHA256                   | Xk/WeYD/K/bnBybzhluu4dWpBJtXPF7E30jhU7se4Ow |
### Authentication Challenge: Authenticity of Host Can't Be Established When Connecting to an AWS CodeCommit Repository

<table>
<thead>
<tr>
<th>Server</th>
<th>Cryptographic hash type</th>
<th>Fingerprint</th>
</tr>
</thead>
<tbody>
<tr>
<td>git-codecommit.ap-southeast-1.amazonaws.com</td>
<td>SHA256</td>
<td>Z1sVa7OVzxrTif +RkUhhPv6Es22mSB3uTBojPXiNo</td>
</tr>
<tr>
<td>git-codecommit.ap-southeast-2.amazonaws.com</td>
<td>SHA256</td>
<td>nYp +gHas80HY3DqbP4yanCDFhDVjseefVbHEXq</td>
</tr>
<tr>
<td>git-codecommit.eu-central-1.amazonaws.com</td>
<td>SHA256</td>
<td>MwGrkiEki8QkktBt1AgXbYtohoZyBnZ6F6VY5f</td>
</tr>
<tr>
<td>git-codecommit.ap-northeast-2.amazonaws.com</td>
<td>SHA256</td>
<td>eegAPqrWY9YSYoZ9IHKOmxetfXBHzAzd8Eys</td>
</tr>
<tr>
<td>git-codecommit.sa-east-1.amazonaws.com</td>
<td>SHA256</td>
<td>kW+VKBojpRaq/ ZbXkgbtMqKgEDEK7JnISIV3SvoyCmzu</td>
</tr>
<tr>
<td>git-codecommit.us-west-1.amazonaws.com</td>
<td>SHA256</td>
<td>gzauWTXDK2u5KuMMi5vbRTmfyerdIwgSbZk</td>
</tr>
<tr>
<td>git-codecommit.eu-west-2.amazonaws.com</td>
<td>SHA256</td>
<td>r0Rwz5k/IHp/QyrRnflMj02D5UEqMbtFNTuDG2hNbs</td>
</tr>
<tr>
<td>git-codecommit.ap-south-1.amazonaws.com</td>
<td>SHA256</td>
<td>hUKwnTj7+Xpx4Kdb6p45j4RaZ1J41hAMD8k</td>
</tr>
<tr>
<td>git-codecommit.ca-central-1.amazonaws.com</td>
<td>SHA256</td>
<td>Qz5puafQdANVprLl6r0Qyh4lCNSF6ob61dGo</td>
</tr>
<tr>
<td>git-codecommit.eu-west-3.amazonaws.com</td>
<td>SHA256</td>
<td>uw7c2FL564jVoGtce +ikzILnKBszZs7t9+CFdSjKbLI</td>
</tr>
</tbody>
</table>
IAM Error: 'Invalid format' when attempting to add a public key to IAM

Problem: In IAM, when attempting to set up to use SSH with AWS CodeCommit, an error message appears containing the phrase **Invalid format** when you attempt to add your public key.

Possible fixes: IAM accepts public keys in the OpenSSH format only. If you provide your public key in another format, or if the key does not contain the required number of bits, you will see this error. This problem most commonly occurs when the public/private key pairs are generated on Windows computers. To generate a key pair and copy the OpenSSH format required by IAM, see the section called “SSH and Windows: Set Up the Public and Private Keys for Git and AWS CodeCommit” (p. 32).

Git on Windows: Bash Emulator or Command Line Freezes When Attempting to Connect Using SSH

Problem: After you configure SSH access for Windows and confirm connectivity at the command line or terminal, you see a message that the server’s host key is not cached in the registry, and the prompt to store the key in the cache is frozen (does not accept y/n/return input) when you attempt to use commands such as **git pull**, **git push**, or **git clone** at the command prompt or Bash emulator.

Possible fixes: The most common cause for this error is that your Git environment is configured to use something other than OpenSSH for authentication (probably PuTTY). This is known to cause problems with the caching of keys in some configurations. To fix this problem, try one of the following:

- Open a Bash emulator and add the `GIT_SSH_COMMAND="ssh"` parameter before the Git command. For example, if you are attempting to push to a repository, instead of typing **git push**, type:

  ```plaintext
  GIT_SSH_COMMAND="ssh" git push
  ```

- If you have PuTTY installed, open PuTTY, and in **Host Name (or IP address)**, type the AWS CodeCommit endpoint you want to reach (for example, git-codecommit.us-east-2.amazonaws.com). Choose **Open**. When prompted by the PuTTY Security Alert, choose **Yes** to permanently cache the key.

- Rename or delete the `GIT_SSH` environment variable if you are no longer using it. Then open a new command prompt or Bash emulator session, and try your command again.

For other solutions, see **Git clone/pull continually freezing at Store key in cache** on Stack Overflow.

Troubleshooting the Credential Helper and HTTPS Connections to AWS CodeCommit

The following information might help you troubleshoot common issues when using the credential helper included with the AWS CLI and HTTPS to connect to AWS CodeCommit repositories.

Topics

- **Git for macOS: I Configured the Credential Helper Successfully, but Now I Am Denied Access to My Repository (403)** (p. 241)
- **Git for Windows: I Installed Git for Windows, but I Am Denied Access to My Repository (403)** (p. 242)
Git for macOS: I Configured the Credential Helper Successfully, but Now I Am Denied Access to My Repository (403)

**Problem:** On macOS, the credential helper does not seem to access or use your credentials as expected. This can be caused by two different problems:

- The AWS CLI is configured for a different AWS region than the one where the repository exists.
- The Keychain Access utility has saved credentials which have since expired.

**Possible fixes:** To verify whether the AWS CLI is configured for the correct region, run the `aws configure` command, and review the displayed information. If the AWS CodeCommit repository is in a different region than the one shown for the AWS CLI, you must run the `aws configure` command and change the values to the appropriate ones for that region. For more information, see Step 1: Initial Configuration for AWS CodeCommit (p. 35).

The default version of Git released on OS X and macOS uses the Keychain Access utility to save generated credentials. For security reasons, the password generated for access to your AWS CodeCommit repository is temporary, so the credentials stored in the keychain will stop working after about 15 minutes. If you are only accessing Git with AWS CodeCommit, try the following:

1. In Terminal, run the `git config` command to find the Git configuration file where the Keychain Access utility is defined:

   ```
   $ git config -l --show-origin
   ```

   In the output from this command, find a line that contains the following option:

   ```
   helper = osxkeychain
   ```

   The file listed at the beginning of this line is the Git configuration file you must edit.

2. To edit the Git configuration file, use a plain-text editor or run the following command:

   ```
   $ nano /usr/local/git/etc/gitconfig
   ```

3. Comment out the following line of text:

   ```
   # helper = osxkeychain
   ```

   Alternatively, if you want to continue to use the Keychain Access utility to cache credentials for other Git repositories, modify the header instead of commenting out the line. For example, to allow cached credentials for GitHub, you could modify the header as follows:

   ```
   [credential "https://github.com"]
   helper = osxkeychain
   ```

   If you are accessing other repositories with Git, you can configure the Keychain Access utility so that it does not supply credentials for your AWS CodeCommit repositories. To configure the Keychain Access utility:

   1. Open the Keychain Access utility. (You can use Finder to locate it.)
Git for Windows: I Installed Git for Windows, but I Am Denied Access to My Repository (403)

Problem: On Windows, the credential helper does not seem to access or use your credentials as expected. This can be caused by different problems:

- The AWS CLI is configured for a different AWS region than the one where the repository exists.
- By default, Git for Windows installs a Git Credential Manager utility that is not compatible with AWS CodeCommit connections that use the AWS credential helper. When installed, it will cause connections to repository to fail even though the credential helper has been installed with the AWS CLI and configured for connections to AWS CodeCommit.
- Some versions of Git for Windows might not be in full compliance with RFC 2617 and RFC 4559, which could potentially cause issues with both Git credentials and the credential helper included with the AWS CLI. For more information, see Version 2.11.0(3) does not ask for username/password.

Possible fixes:

- If you are attempting to use the credential helper included with the AWS CLI, consider connecting with Git credentials over HTTPS instead of using the credential helper. Git credentials configured for your IAM user are compatible with the Git Credential Manager for Windows, unlike the credential helper for AWS CodeCommit. For more information, see For HTTPS Users Using Git Credentials (p. 5).

If you want to use the credential helper, to verify whether the AWS CLI is configured for the correct region, run the `aws configure` command, and review the displayed information. If the AWS CodeCommit repository is in a different region than the one shown for the AWS CLI, you must run the `aws configure` command and change the values to the appropriate ones for that region. For more information, see Step 1: Initial Configuration for AWS CodeCommit (p. 39).
• If possible, uninstall and reinstall Git for Windows. When installing Git for Windows, clear the check box for the option for installing the Git Credential Manager utility. This credential manager is not compatible with the credential helper for AWS CodeCommit. If you installed the Git Credential Manager or another credential management utility and you do not want to uninstall it, you can modify your .gitconfig file and add specific credential management for AWS CodeCommit:

1. Open Control Panel, choose Credential Manager, and remove any stored credentials for AWS CodeCommit.
2. Open your .gitconfig file in any plain-text editor, such as Notepad.

   **Note**
   If you work with multiple Git profiles, you might have both local and global .gitconfig files. Be sure to edit the appropriate file.
3. Add the following section to your .gitconfig file:

   ```
   [credential "https://git-codecommit.*.amazonaws.com"]
   helper = !aws codecommit credential-helper $@
   UseHttpPath = true
   ```

4. Save the file, and then open a new command line session before you attempt to connect again.

You can also use this approach if you want to use the credential helper for AWS CodeCommit when connecting to AWS CodeCommit repositories and another credential management system when connecting to other hosted repositories, such as GitHub repositories.

To reset which credential helper is used as the default, you can use the `--system` option instead of `--global` or `--local` when running the `git config` command.

• If you are using Git credentials on a Windows computer, you can try to work around any RFC noncompliance issues by including your Git credential user name as part of the connection string. For example, to work around the issue and clone a repository named *MyDemoRepo* in the US East (Ohio) region:

   ```
   git clone https://Your-Git-Credential-Username@git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo my-demo-repo
   ```

   **Note**
   This approach will not work if you have an @ character in your Git credentials username. You must URL encode (also known as URL escaping or percent-encoding) the character before it will work.

---

**Troubleshooting Git Clients and AWS CodeCommit**

The following information might help you troubleshoot common issues when using Git with AWS CodeCommit repositories. For troubleshooting problems specific to Git clients when using HTTPS or SSH, also see Troubleshooting Git Credentials (HTTPS) (p. 235), Troubleshooting SSH Connections (p. 236), and Troubleshooting the Credential Helper (HTTPS) (p. 240).

**Topics**

- **Git Error: error: RPC failed; result=56, HTTP code = 200 fatal: The remote end hung up unexpectedly** (p. 244)
- **Git Error: Too many reference update commands** (p. 244)
- **Git Error: push via HTTPS is broken in some versions of Git** (p. 244)
- **Git Error: 'gnutls_handshake()' failed'** (p. 244)
Git Error: error: RPC failed; result=56, HTTP code = 200 fatal: The remote end hung up unexpectedly

**Problem:** When pushing a large change, a large number of changes, or a large repository, long-running HTTPS connections are often terminated prematurely due to networking issues or firewall settings.

**Possible fixes:** Push with SSH instead, or when migrating a large repository, follow the steps in Migrate a Repository in Increments (p. 227). Also, make sure you are not exceeding the size limits for individual files. For more information, see Limits (p. 282).

Git Error: Too many reference update commands

**Problem:** The maximum number of reference updates per push is 4,000. This error appears when the push contains more than 4,000 reference updates.

**Possible fixes:** Try pushing branches and tags individually with `git push --all` and `git push --tags`. If you have too many tags, split the tags into multiple pushes. For more information, see Limits (p. 282).

Git Error: push via HTTPS is broken in some versions of Git

**Problem:** An issue with the curl update to 7.41.0 causes SSPI-based digest authentication to fail. Known affected versions of Git include 1.9.5.msysgit.1. Additionally, some versions of Git for Windows might not be in full compliance with RFC 2617 and RFC 4559, which could potentially cause issues with HTTPS connections using either Git credentials or the credential helper included with the AWS CLI.

**Possible fixes:** Check your version of Git for known issues or use an earlier or later version. For more information about msysgit, see Push to HTTPS Is Broken in the GitHub forums. For more information about Git for Windows version issues, see Version 2.11.0(3) does not ask for username/password.

Git Error: 'gnutls_handshake() failed'

**Problem:** In Linux, when you try to use Git to communicate with an AWS CodeCommit repository, an error message appears containing the phrase `error: gnutls_handshake() failed.`

**Possible fixes:** Compile Git against OpenSSL. For one approach, see “Error: gnutls_handshake() failed” When Connecting to HTTPS Servers in the Ask Ubuntu forums.

Alternatively, use SSH instead of HTTPS to communicate with AWS CodeCommit repositories.

Git Error: Git cannot find the AWS CodeCommit repository or does not have permission to access the repository

**Problem:** A trailing slash in the connection string can cause connection attempts to fail.
Possible fixes: Make sure that you have provided the correct name and connection string for the repository, and that there are no trailing slashes. For more information, see Connect to a Repository (p. 78).

**Git on Windows: No Supported Authentication Methods Available (publickey)**

**Problem:** After you configure SSH access for Windows, you see an access denied error when you attempt to use commands such as `git pull`, `git push`, or `git clone`.

**Possible fixes:** The most common cause for this error is that a GIT_SSH environment variable exists on your computer and is configured to support another connection utility, such as PuTTY. To fix this problem, try one of the following:

- Open a Bash emulator and add the `GIT_SSH_COMMAND="ssh"` parameter before the Git command. For example, if you are attempting to clone a repository, instead of typing `git clone ssh://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo my-demo-repo`, type:

  ```
  GIT_SSH_COMMAND="ssh" git clone ssh://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo my-demo-repo
  ```

- Rename or delete the GIT_SSH environment variable if you are no longer using it. Then open a new command prompt or Bash emulator session, and try your command again.

  For more information about troubleshooting Git issues on Windows when using SSH, see Troubleshooting SSH Connections (p. 236).

**Troubleshooting Access Errors and AWS CodeCommit**

The following information might help you troubleshoot access errors you might see when connecting with AWS CodeCommit repositories.

**Topics**

- Access Error: Prompted for AWS User Name When Connecting to an AWS CodeCommit Repository (p. 245)
- Access Error: Prompted for User Name and Password When Connecting to an AWS CodeCommit Repository from Windows (p. 246)
- Access Error: Public Key Denied When Connecting to an AWS CodeCommit Repository (p. 246)

**Access Error: Prompted for AWS User Name When Connecting to an AWS CodeCommit Repository**

**Problem:** When you try to use Git to communicate with an AWS CodeCommit repository, a message appears prompting you for your AWS user name.

**Possible fixes:** Configure your AWS profile or make sure the profile you are using is the one you configured for working with AWS CodeCommit. For more information about setting up, see Setting Up...
Access Error: Prompted for User Name and Password When Connecting to an AWS CodeCommit Repository from Windows

**Problem:** When you try to use Git to communicate with an AWS CodeCommit repository, you see a pop-up dialog box asking for your user name and password.

**Possible fixes:** This might be the built-in credential management system for Windows. Depending on your configuration, do one of the following:

- If you are using HTTPS with Git credentials, your Git credentials are not yet stored in the system. Provide the Git credentials and continue. You should not be prompted again. For more information, see [For HTTPS Users Using Git Credentials](p. 5).
- If you are using HTTPS with the credential helper for AWS CodeCommit, it is not compatible with the Windows credential management system. Choose **Cancel**.

This might also be an indication that you installed the Git Credential Manager as part of installing Git for Windows. The Git Credential Manager is not compatible with AWS CodeCommit. Consider uninstalling it.

For more information, see [For HTTPS Connections on Windows with the AWS CLI Credential Helper](p. 39) and [Git for Windows: I Installed Git for Windows, but I Am Denied Access to My Repository](p. 242).

Access Error: Public Key Denied When Connecting to an AWS CodeCommit Repository

**Problem:** When you try to use an SSH endpoint to communicate with an AWS CodeCommit repository, an error message appears containing the phrase **Error: public key denied**.

**Possible fixes:** The most common reason for this error is that you have not completed set up for SSH connections. Configure a public and private SSH key pair, and then associate the public key with your IAM user. For more information about configuring SSH, see [For SSH Connections on Linux, macOS, or Unix](p. 26) and [For SSH Connections on Windows](p. 30).

Troubleshooting Configuration Errors and AWS CodeCommit

The following information might help you troubleshoot configuration errors you might see when connecting with AWS CodeCommit repositories.

**Topics**

- [Configuration Error: Cannot Configure AWS CLI Credentials on macOS](p. 247)
Configuration Error: Cannot Configure AWS CLI Credentials on macOS

Problem: When you run `aws configure` to configure the AWS CLI, you see a `ConfigParseError` message.

Possible fixes: The most common cause for this error is that a credentials file already exists. Browse to `~/.aws` and look for a file named `credentials`. Rename or delete that file, and then run `aws configure` again.

Troubleshooting Console Errors and AWS CodeCommit

The following information might help you troubleshoot console errors you might see when using AWS CodeCommit repositories.

Topics
- Access Error: Encryption Key Access Denied for an AWS CodeCommit Repository from the Console or the AWS CLI (p. 247)
- Console Error: Cannot Browse the Code in an AWS CodeCommit Repository from the Console (p. 247)

Access Error: Encryption Key Access Denied for an AWS CodeCommit Repository from the Console or the AWS CLI

Problem: When you try to access AWS CodeCommit from the console or the AWS CLI, an error message appears containing the phrase `EncryptionKeyAccessDeniedException` or `User is not authorized for the KMS default master key for CodeCommit 'aws/codecommit' in your account`.

Possible fixes: The most common cause for this error is that your AWS account is not subscribed to AWS Key Management Service, which is required for AWS CodeCommit. Open the IAM console, choose Encryption Keys, and then choose Get Started Now. If you see a message that you are not currently subscribed to the AWS Key Management Service service, follow the instructions on that page to subscribe. For more information about AWS CodeCommit and AWS Key Management Service, see AWS KMS and Encryption (p. 288).

Console Error: Cannot Browse the Code in an AWS CodeCommit Repository from the Console

Problem: When you try to browse the contents of a repository from the console, an error message appears denying access.

Possible fixes: The most common cause for this error is that an IAM policy applied to your AWS account denies one or more of the permissions required for browsing code from the AWS CodeCommit console. For more information about AWS CodeCommit access permissions and browsing, see Authentication and Access Control for AWS CodeCommit (p. 250).
Troubleshooting Triggers and AWS CodeCommit

The following information might help you troubleshoot issues with triggers you might see in AWS CodeCommit.

Topics
- Trigger Error: A Repository Trigger Does Not Run When Expected (p. 248)

Trigger Error: A Repository Trigger Does Not Run When Expected

**Problem:** One or more triggers configured for a repository does not appear to run or does not run as expected.

**Possible fixes:** If the target of the trigger is an AWS Lambda function, make sure you have configured the function’s resource policy for access by AWS CodeCommit. For more information, see Example 2: Create a Policy for AWS Lambda Integration (p. 267).

Alternatively, edit the trigger and make sure the events for which you want to trigger actions have been selected and that the branches for the trigger include the branch where you want to see responses to actions. Try changing the settings for the trigger to **All repository events** and **All branches** and then testing the trigger. For more information, see Edit Triggers for a Repository (p. 105).

Turn on Debugging

**Problem:** I want to turn on debugging to get more information about my repository and how Git is executing commands.

**Possible fixes:** Try the following:

1. At the terminal or command prompt, run the following commands on your local machine before running Git commands:

   **On Linux, macOS, or Unix:**
   ```
   export GIT_TRACE_PACKET=1
   export GIT_TRACE=1
   export GIT_CURL_VERBOSE=1
   ```

   **On Windows:**
   ```
   set GIT_TRACE_PACKET=1
   set GIT_TRACE=1
   set GIT_CURL_VERBOSE=1
   ```

   **Note**
   Setting `GIT_CURL_VERBOSE` is useful for HTTPS connections only. SSH does not use the libcurl library.

2. To get more information about your Git repository, create a shell script similar to the following, and then run the script:

   ```
   #!/bin/sh
   ```
3. If these steps do not provide enough information for you to resolve the issue on your own, ask for help on the AWS CodeCommit forum. Be sure to include relevant output from these steps in your post.
Authentication and Access Control for AWS CodeCommit

Access to AWS CodeCommit requires credentials. Those credentials must have permissions to access AWS resources, such as AWS CodeCommit repositories, and your IAM user, which you use to manage your Git credentials or the SSH public key that you use for making Git connections. The following sections provide details on how you can use AWS Identity and Access Management (IAM) and AWS CodeCommit to help secure access to your resources:

- Authentication (p. 250)
- Access Control (p. 251)

Authentication

Because AWS CodeCommit repositories are Git-based and support the basic functionality of Git, including Git credentials, we recommend that you use an IAM user when working with AWS CodeCommit. You can access AWS CodeCommit with other identity types, but the other identity types are subject to limitations, as described below.

Identity types:

- **IAM user** – An IAM user is simply an identity within your AWS account that has specific custom permissions. For example, an IAM user can have permissions to create and manage Git credentials for accessing AWS CodeCommit repositories. **This is the recommended user type for working with AWS CodeCommit.** You can use an IAM user name and password to sign in to secure AWS webpages like the AWS Management Console, AWS Discussion Forums, or the AWS Support Center.

  You can generate Git credentials or associate SSH public keys with your IAM user. These are the easiest ways to set up Git to work with your AWS CodeCommit repositories. With Git credentials, you generate a static user name and password in IAM. You then use these credentials for HTTPS connections with Git and any third-party tool that supports Git user name and password authentication. With SSH connections, you create public and private key files on your local machine that Git and AWS CodeCommit use for SSH authentication. You associate the public key with your IAM user, and you store the private key on your local machine.

  In addition, you can generate access keys for each user. Use access keys when you access AWS services programmatically, either through one of the AWS SDKs or by using the AWS Command Line Interface (AWS CLI). The SDK and CLI tools use the access keys to cryptographically sign your requests. If you don’t use the AWS tools, you must sign the requests yourself. AWS CodeCommit supports Signature Version 4, a protocol for authenticating inbound API requests. For more information about authenticating requests, see Signature Version 4 Signing Process in the AWS General Reference.

- **AWS account root user** – When you sign up for AWS, you provide an email address and password that is associated with your AWS account. These are your root credentials, and they provide complete access to all of your AWS resources. Certain AWS CodeCommit features are not available for root account users. In addition, the only way to use Git with your root account is to configure the AWS credential helper, which is included with the AWS CLI. You cannot use Git credentials or SSH public-private key pairs with your root account user. For these reasons, we do not recommend using your root account user when interacting with AWS CodeCommit.

  **Important**

  For security reasons, we recommend that you use the root credentials only to create an administrator user, which is an IAM user with full permissions to your AWS account. Then, you
can use this administrator user to create other IAM users and roles with limited permissions. For more information, see IAM Best Practices and Creating an Admin User and Group in the IAM User Guide.

- **IAM role** – Like an IAM user, an IAM role is an IAM identity that you can create in your account to grant specific permissions. It is similar to an IAM user, but it is not associated with a specific person. Unlike an IAM user identity, you cannot use Git credentials or SSH keys with this identity type. However, an IAM role enables you to obtain temporary access keys that you can use to access AWS services and resources. IAM roles with temporary credentials are useful in the following situations:

  - **Federated user access** – Instead of creating an IAM user, you can use preexisting user identities from AWS Directory Service, your enterprise user directory, or a web identity provider. These are known as federated users. AWS assigns a role to a federated user when access is requested through an identity provider. For more information about federated users, see Federated Users and Roles in the IAM User Guide.

    **Note**
    You cannot use Git credentials or SSH public-private key pairs with federated users. In addition, user preferences are not available for federated users.

  - **Cross-account access** – You can use an IAM role in your account to grant another AWS account permissions to access your account's resources. For an example, see Tutorial: Delegate Access Across AWS Accounts Using IAM Roles in the IAM User Guide.

  - **AWS service access** – You can use an IAM role in your account to grant an AWS service permissions to access your account's resources. For example, you can create a role that allows AWS Lambda to access an AWS CodeCommit repository on your behalf. For more information, see Creating a Role to Delegate Permissions to an AWS Service in the IAM User Guide.

  - **Applications running on Amazon EC2** – Instead of storing access keys within an EC2 instance for use by applications running on the instance and for making AWS API requests, you can use an IAM role to manage temporary credentials for these applications. To assign an AWS role to an EC2 instance and make it available to all of its applications, you can create an instance profile that is attached to the instance. An instance profile contains the role and enables programs running on the EC2 instance to get temporary credentials. For more information, see Using Roles for Applications on Amazon EC2 in the IAM User Guide.

---

### Access Control

You can have valid credentials to authenticate your requests, but unless you have permissions you cannot create or access AWS CodeCommit resources. For example, you must have permissions to view repositories, push code, create and manage Git credentials, and so on.

The following sections describe how to manage permissions for AWS CodeCommit. We recommend that you read the overview first.

- **Overview of Managing Access Permissions to Your AWS CodeCommit Resources (p. 251)**
- **Using Identity-Based Policies (IAM Policies) for AWS CodeCommit (p. 256)**
- **AWS CodeCommit Permissions Reference (p. 267)**

### Overview of Managing Access Permissions to Your AWS CodeCommit Resources

Every AWS resource is owned by an AWS account. Permissions to create or access a resource are governed by permissions policies. An account administrator can attach permissions policies to IAM identities (that is, users, groups, and roles). Some services, such as AWS Lambda, also support attaching permissions policies to resources.
Note
An account administrator (or administrator user) is a user with administrator privileges. For more information, see IAM Best Practices in the IAM User Guide.

When granting permissions, you decide who gets the permissions, the resources they get permissions for, and the specific actions that you want to allow on those resources.

Topics
- AWS CodeCommit Resources and Operations (p. 252)
- Understanding Resource Ownership (p. 253)
- Managing Access to Resources (p. 253)
- Resource Scoping in AWS CodeCommit (p. 254)
-Specifying Policy Elements: Resources, Actions, Effects, and Principals (p. 255)
- Specifying Conditions in a Policy (p. 255)

AWS CodeCommit Resources and Operations

In AWS CodeCommit, the primary resource is a repository. Each resource has a unique Amazon Resource Names (ARN) associated with it. In a policy, you use an Amazon Resource Name (ARN) to identify the resource that the policy applies to. For more information about ARNs, see Amazon Resource Names (ARN) and AWS Service Namespaces in the Amazon Web Services General Reference. AWS CodeCommit does not currently support other resource types, which are referred to as subresources.

The following table describes how to specify AWS CodeCommit resources.

<table>
<thead>
<tr>
<th>Resource Type</th>
<th>ARN Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Repository</td>
<td><code>arn:aws:codecommit:region:account-id:repository-name</code></td>
</tr>
<tr>
<td>All AWS CodeCommit repositories</td>
<td><code>arn:aws:codecommit:*</code></td>
</tr>
<tr>
<td>All AWS CodeCommit repositories owned by the specified account in the specified region</td>
<td><code>arn:aws:codecommit:region:account-id:*</code></td>
</tr>
</tbody>
</table>

Note
Most AWS services treat a colon (:) or a forward slash (/) in ARNs as the same character. However, AWS CodeCommit requires an exact match in resource patterns and rules. When creating event patterns, be sure to use the correct ARN characters so that they match the ARN syntax in the resource.

For example, you can indicate a specific repository (MyDemoRepo) in your statement using its ARN as follows:

```
```

To specify all repositories that belong to a specific account, use the wildcard character (*) as follows:

```
```
Understanding Resource Ownership

The AWS account owns the resources that are created in the account, regardless of who created them. Specifically, the resource owner is the AWS account of the principal entity (that is, the root account, an IAM user, or an IAM role) that authenticates the resource creation request. The following examples illustrate how this works:

- If you create an IAM user in your AWS account and grant permissions to create AWS CodeCommit resources to that user, the user can create AWS CodeCommit resources. However, your AWS account, to which the user belongs, owns the AWS CodeCommit resources.
- If you use the root account credentials of your AWS account to create a rule, your AWS account is the owner of the AWS CodeCommit resource.
- If you create an IAM role in your AWS account with permissions to create AWS CodeCommit resources, anyone who can assume the role can create AWS CodeCommit resources. Your AWS account, to which the role belongs, owns the AWS CodeCommit resources.

Managing Access to Resources

To manage access to AWS resources, you use permissions policies. A permissions policy describes who has access to what. The following section explains the options for creating permissions policies.

**Note**

This section discusses using IAM in the context of AWS CodeCommit. It doesn't provide detailed information about the IAM service. For more information about IAM, see What Is IAM? in the IAM User Guide. For information about IAM policy syntax and descriptions, see AWS IAM Policy Reference in the IAM User Guide.

Permissions policies that are attached to an IAM identity are referred to as identity-based policies (IAM policies). Permissions policies that are attached to a resource are referred to as resource-based policies. Currently, AWS CodeCommit supports only identity-based policies (IAM policies).

**Topics**
- Identity-Based Policies (IAM Policies) (p. 253)
- Resource-Based Policies (p. 254)

**Identity-Based Policies (IAM Policies)**

To manage access to AWS resources, you attach permissions policies to IAM identities. In AWS CodeCommit, you use identity-based policies to control access to repositories. For example, you can do the following:
• **Attach a permissions policy to a user or a group in your account** – To grant a user permissions to view AWS CodeCommit resources in the AWS CodeCommit console, attach an identity-based permissions policy to a user or group that the user belongs to.

• **Attach a permissions policy to a role (to grant cross-account permissions)** – Delegation, such as when you want to grant cross-account access, involves setting up a trust between the account that owns the resource (the trusting account), and the account that contains the users who need to access the resource (the trusted account). A permissions policy grants the user of a role the needed permissions to carry out the intended tasks on the resource. A trust policy specifies which trusted accounts are allowed to grant its users permissions to assume the role. For more information, see [IAM Terms and Concepts](https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles.html).

To grant cross-account permissions, attach an identity-based permissions policy to an IAM role. For example, the administrator in Account A can create a role to grant cross-account permissions to another AWS account (for example, Account B) or an AWS service as follows:

1. Account A administrator creates an IAM role and attaches a permissions policy to the role that grants permissions on resources in Account A.
2. Account A administrator attaches a trust policy to the role identifying Account B as the principal who can assume the role.
3. Account B administrator can then delegate permissions to assume the role to any users in Account B. Doing this allows users in Account B to create or access resources in Account A. If you want to grant an AWS service permission to assume the role, the principal in the trust policy can also be an AWS service principal. For more information, see [Delegation in IAM Terms and Concepts](https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles.html).

For more information about using IAM to delegate permissions, see [Access Management](https://docs.aws.amazon.com/IAM/latest/UserGuide/using-access-management.html) in the *IAM User Guide*.

The following example policy allows a user to create a branch in a repository named *MyDemoRepo*:

```json
{
   "Version": "2012-10-17",
   "Statement" : [
      {
         "Effect" : "Allow",
         "Action" : [ "codecommit:CreateBranch" ],
      }
   ]
}
```

To restrict the calls and resources that users in your account have access to, create specific IAM policies, and then attach those policies to IAM users. For more information about how to create IAM roles and to explore example IAM policy statements for AWS CodeCommit, see [Overview of Managing Access Permissions to Your AWS CodeCommit Resources](https://docs.aws.amazon.com/CodeCommit/latest/userguide/authoring-policy-statements.html) (p. 251).

### Resource-Based Policies

Some services, such as Amazon S3, also support resource-based permissions policies. For example, you can attach a resource-based policy to an S3 bucket to manage access permissions to that bucket. AWS CodeCommit doesn't support resource-based policies.

### Resource Scoping in AWS CodeCommit

In AWS CodeCommit, you can scope identity-based policies and permissions to resources, as described in [AWS CodeCommit Resources and Operations](https://docs.aws.amazon.com/CodeCommit/latest/userguide/authoring-policy-statements.html) (p. 252). However, you cannot scope the...
ListRepositories permission to a resource. Instead, you must scope it to all resources (using the wildcard `*`). Otherwise, the action fails.

All other AWS CodeCommit permissions can be scoped to resources.

### Specifying Policy Elements: Resources, Actions, Effects, and Principals

You can create policies to allow or deny users access to resources, or allow or deny users to take specific actions on those resources. AWS CodeCommit defines a set of public API operations that define how users work with the service, whether that is through the AWS CodeCommit console, the SDKs, the AWS CLI, or by directly calling those APIs. To grant permissions for these API operations, AWS CodeCommit defines a set of actions that you can specify in a policy.

Some API operations can require permissions for more than one action. For more information about resources and API operations, see [AWS CodeCommit Resources and Operations](p. 252) and [AWS CodeCommit Permissions Reference](p. 267).

The following are the basic elements of a policy:

- **Resource** – To identify the resource that the policy applies to, you use an Amazon Resource Name (ARN). For more information, see [AWS CodeCommit Resources and Operations](p. 252).
- **Action** – To identify resource operations that you want to allow or deny, you use action keywords. For example, depending on the specified **Effect**, the codecommit:GetBranch permission either allows or denies the user to perform the GetBranch operation, which gets details about a branch in an AWS CodeCommit repository.
- **Effect** – You specify the effect, either allow or deny, that takes place when the user requests the specific action. If you don't explicitly grant access to (allow) a resource, access is implicitly denied. You can also explicitly deny access to a resource to make sure that a user cannot access it, even if a different policy grants access.
- **Principal** – In identity-based policies (IAM policies), the only type of policies that AWS CodeCommit supports, the user that the policy is attached to is the implicit principal.

To learn more about IAM policy syntax, see [AWS IAM Policy Reference](in the IAM User Guide).

For a table showing all of the AWS CodeCommit API actions and the resources that they apply to, see [AWS CodeCommit Permissions Reference](p. 267).

### Specifying Conditions in a Policy

When you grant permissions, you use the access policy language for IAM to specify the conditions under which a policy should take effect. For example, you might want a policy to be applied only after a specific date. For more information about specifying conditions in a policy language, see [Condition](and Policy Grammar in the IAM User Guide).

To express conditions, you use predefined condition keys. There are no condition keys specific to AWS CodeCommit. However, there are AWS-wide condition keys that you can use as appropriate. For a complete list of AWS-wide keys, see [Available Keys for Conditions](in the IAM User Guide).
Using Identity-Based Policies (IAM Policies) for AWS CodeCommit

The following examples of identity-based policies demonstrate how an account administrator can attach permissions policies to IAM identities (users, groups, and roles) to grant permissions to perform operations on AWS CodeCommit resources.

**Important**

We recommend that you first review the introductory topics that explain the basic concepts and options available to manage access to your AWS CodeCommit resources. For more information, see Overview of Managing Access Permissions to Your AWS CodeCommit Resources (p. 251).

**Topics**

- Permissions Required to Use the AWS CodeCommit Console (p. 256)
- AWS Managed (Predefined) Policies for AWS CodeCommit (p. 258)
- Customer Managed Policy Examples (p. 263)

The following is an example of an identity-based permissions policy:

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codecommit:BatchGetRepositories"
      ],
      "Resource": [
      ]
    }
  ]
}
```

This policy has one statement that allows a user to get information about the AWS CodeCommit repository named MyDestinationRepo and all AWS CodeCommit repositories that start with the name MyDemo in the us-east-2 Region.

**Permissions Required to Use the AWS CodeCommit Console**

To see the required permissions for each AWS CodeCommit API operation, and for more information about AWS CodeCommit operations, see AWS CodeCommit Permissions Reference (p. 267).

To allow users to use the AWS CodeCommit console, the administrator must grant them permissions for AWS CodeCommit actions. For example, you could attach the AWSCodeCommitPowerUser managed policy or its equivalent to a user or group, as shown in the following permissions policy:

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codecommit:*"
      ],
      "Resource": [
      ]
    }
  ]
}
```
Permissions Required to Use the AWS CodeCommit Console

```json
{
    "codecommit:BatchGet*",
    "codecommit:Get*",
    "codecommit:List*",
    "codecommit:Create*",
    "codecommit:DeleteBranch",
    "codecommit:Describe*",
    "codecommit:Put*",
    "codecommit:Post*",
    "codecommit:Merge*",
    "codecommit:Test*",
    "codecommit:Update*",
    "codecommit:GitPull",
    "codecommit:GitPush"
},
"Resource": "*"
},
{
    "Sid": "CloudWatchEventsCodeCommitRulesAccess",
    "Effect": "Allow",
    "Action": [
        "events:DeleteRule",
        "events:DescribeRule",
        "events:DisableRule",
        "events:EnableRule",
        "events:PutRule",
        "events:PutTargets",
        "events:RemoveTargets",
        "events:ListTargetsByRule"
    ],
    "Resource": "arn:aws:events:*:*:rule/codecommit*"
},
{
    "Sid": "SNSTopicAndSubscriptionAccess",
    "Effect": "Allow",
    "Action": [
        "sns:Subscribe",
        "sns:Unsubscribe"
    ],
    "Resource": "arn:aws:sns:*:*:codecommit*"
},
{
    "Sid": "SNSTopicAndSubscriptionReadAccess",
    "Effect": "Allow",
    "Action": [
        "sns:ListTopics",
        "sns:ListSubscriptionsByTopic",
        "sns:GetTopicAttributes"
    ],
    "Resource": "*"
},
{
    "Sid": "LambdaReadOnlyListAccess",
    "Effect": "Allow",
    "Action": [
        "lambda:ListFunctions"
    ],
    "Resource": "*"
},
{
    "Sid": "IAMReadOnlyListAccess",
    "Effect": "Allow",
    "Action": [
        "iam:ListUsers"
    ],
    "Resource": "*"
}
```
In addition to permissions granted to users by identity-based polices, AWS CodeCommit requires permissions for AWS Key Management Service (AWS KMS) actions. An IAM user does not need explicit Allow permissions for these actions, but the user must not have any policies attached that set the following permissions to Deny:

- `kms:Encrypt`
- `kms:Decrypt`
- `kms:ReEncrypt`
- `kms:GenerateDataKey`
- `kms:GenerateDataKeyWithoutPlaintext`
- `kms:DescribeKey`

For more information about encryption and AWS CodeCommit, see AWS KMS and Encryption (p. 288).

**AWS Managed (Predefined) Policies for AWS CodeCommit**

AWS addresses many common use cases by providing standalone IAM policies that are created and administered by AWS. These AWS managed policies grant required permissions for common use cases. The managed policies for AWS CodeCommit also provide permissions to perform operations in other services, such as IAM, Amazon SNS, and Amazon CloudWatch Events, as required for the responsibilities for the users who have been granted the policy in question. For example, the AWSCodeCommitFullAccess
policy is an administrative-level user policy that allows users with this policy to create and manage CloudWatch Events rules for repositories (rules whose names are prefixed with codecommit) and Amazon SNS topics for notifications about repository-related events (topics whose names are prefixed with codecommit), as well as administer repositories in AWS CodeCommit.

The following AWS managed policies, which you can attach to users in your account, are specific to AWS CodeCommit:

- **AWSCodeCommitFullAccess** – Grants full access to AWS CodeCommit. Apply this policy only to administrative-level users to whom you want to grant full control over AWS CodeCommit repositories and related resources in your AWS account, including the ability to delete repositories.

The AWSCodeCommitFullAccess policy contains the following policy statement:

```json
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "codecommit:*"
         ],
         "Resource": "*"
      },
      {
         "Sid": "CloudWatchEventsCodeCommitRulesAccess",
         "Effect": "Allow",
         "Action": [
            "events:DeleteRule",
            "events:DescribeRule",
            "events:DisableRule",
            "events:EnableRule",
            "events:PutRule",
            "events:PutTargets",
            "events:RemoveTargets",
            "events:ListTargetsByRule"
         ],
         "Resource": "arn:aws:events:*:*:rule/codecommit*"
      },
      {
         "Sid": "SNSTopicAndSubscriptionAccess",
         "Effect": "Allow",
         "Action": [
            "sns:CreateTopic",
            "sns:DeleteTopic",
            "sns:Subscribe",
            "sns:Unsubscribe",
            "sns:SetTopicAttributes"
         ],
         "Resource": "arn:aws:sns:*:*:codecommit*"
      },
      {
         "Sid": "SNSTopicAndSubscriptionReadAccess",
         "Effect": "Allow",
         "Action": [
            "sns:ListTopics",
            "sns:ListSubscriptionsByTopic",
            "sns:GetTopicAttributes"
         ],
         "Resource": "*"
      }
   ],
   "Sid": "LambdaReadOnlyListAccess",
   "Effect": "Allow",
   "Action": [
      "lambda:List*"
   ],
   "Resource": "*"
}
```
AWS Managed (Predefined) Policies for AWS CodeCommit

- **AWSCodeCommitPowerUser** – Allows users access to all of the functionality of AWS CodeCommit and repository-related resources, except it does not allow them to delete AWS CodeCommit repositories or create or delete repository-related resources in other AWS services, such as Amazon CloudWatch Events. We recommend that you apply this policy to most users.

The AWSCodeCommitPowerUser policy contains the following policy statement:

```json
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": ["sts:AssumeRole"]
      }
   ]
}
```
"codecommit:BatchGet",
"codecommit:Get",
"codecommit:List",
"codecommit:Create",
"codecommit:DeleteBranch",
"codecommit:Describe",
"codecommit:Put",
"codecommit:Post",
"codecommit:Merge",
"codecommit:Test",
"codecommit:Update",
"codecommit:GitPull",
"codecommit:GitPush"
],
"Resource": "*"
},
{
"Sid": "CloudWatchEventsCodeCommitRulesAccess",
"Effect": "Allow",
"Action": [
"events:DeleteRule",
"events:DescribeRule",
"events:DisableRule",
"events:EnableRule",
"events:PutRule",
"events:PutTargets",
"events:RemoveTargets",
"events:ListTargetsByRule"
],
"Resource": "arn:aws:events:*:*:rule/codecommit*"
},
{
"Sid": "SNSTopicAndSubscriptionAccess",
"Effect": "Allow",
"Action": [
"sns:Subscribe",
"sns:Unsubscribe"
],
"Resource": "arn:aws:sns:*:*:codecommit*"
},
{
"Sid": "SNSTopicAndSubscriptionReadAccess",
"Effect": "Allow",
"Action": [
"sns:ListTopics",
"sns:ListSubscriptionsByTopic",
"sns:GetTopicAttributes"
],
"Resource": "*"
},
{
"Sid": "LambdaReadOnlyListAccess",
"Effect": "Allow",
"Action": [
"lambda:ListFunctions"
],
"Resource": "*"
},
{
"Sid": "IAMReadOnlyListAccess",
"Effect": "Allow",
"Action": [
"iam:ListUsers"
],
"Resource": "*"
}
AWS CodeCommit User Guide

AWS Managed (Predefined) Policies for AWS CodeCommit

- **AWSCodeCommitReadOnly** – Grants read-only access to AWS CodeCommit and repository-related resources in other AWS services, as well as the ability to create and manage their own AWS CodeCommit-related resources (such as Git credentials and SSH keys for their IAM user to use when accessing repositories). Apply this policy to users to whom you want to grant the ability to read the contents of a repository, but not make any changes to its contents.

The AWSCodeCommitReadOnly policy contains the following policy statement:

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codecommit:BatchGet*",
                "codecommit:Get*",
                "codecommit:Describe*",
                "codecommit:List*",
                "codecommit:GitPull"
            ],
            "Resource": "*"
        },
        {
            "Sid": "CloudWatchEventsCodeCommitRulesReadOnlyAccess",
            "Effect": "Allow",
            "Action": [
```
"events:DescribeRule",
"events:ListTargetsByRule"
],
"Resource": "arn:aws:events:*:*:rule/codecommit**"
},
{
"Sid": "SNSSubscriptionAccess",
"Effect": "Allow",
"Action": [
"sns:ListTopics",
"sns:ListSubscriptionsByTopic",
"sns:GetTopicAttributes"
],
"Resource": "*"
},
{
"Sid": "LambdaReadOnlyListAccess",
"Effect": "Allow",
"Action": [
"lambda:ListFunctions"
],
"Resource": "*"
},
{
"Sid": "IAMReadOnlyListAccess",
"Effect": "Allow",
"Action": [
"iam:ListUsers"
],
"Resource": "*"
},
{
"Sid": "IAMReadOnlyConsoleAccess",
"Effect": "Allow",
"Action": [
"iam:ListAccessKeys",
"iam:ListSSHPublicKeys",
"iam:ListServiceSpecificCredentials",
"iam:ListAccessKeys",
"iam:GetSSHPublicKey"
],
"Resource": "arn:aws:iam::*:user/${aws:username}" }
}

For more information, see AWS Managed Policies in the IAM User Guide.

Customer Managed Policy Examples

You can create your own custom IAM policies to allow permissions for AWS CodeCommit actions and resources. You can attach these custom policies to the IAM users or groups that require those permissions. You can also create your own custom IAM policies for integration between AWS CodeCommit and other AWS services.

Topics

- Customer Managed Identity Policy Examples (p. 264)
- Customer Managed Integration Policy Examples (p. 266)
Customer Managed Identity Policy Examples

The following example IAM policies grant permissions for various AWS CodeCommit actions. Use them to limit AWS CodeCommit access for your IAM users and roles. These policies control the ability to perform actions with the AWS CodeCommit console, API, AWS SDKs, or the AWS CLI.

**Note**

All examples use the US West (Oregon) Region (us-west-2) and contain fictitious account IDs.

**Examples**

- Example 1: Allow a User to Perform AWS CodeCommit Operations in a Single Region (p. 264)
- Example 2: Allow a User to Use Git for a Single Repository (p. 264)
- Example 3: Allow a User Connecting From a Specific IP Address Range Access to a Repository (p. 265)

**Example 1: Allow a User to Perform AWS CodeCommit Operations in a Single Region**

The following permissions policy uses a wildcard character ("codecommit:*") to allow users to perform all AWS CodeCommit actions in the us-east-2 Region.

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["codecommit:*"],
            "Resource": "arn:aws:codecommit:us-east-2:*"
        }
    ]
}
```

**Example 2: Allow a User to Use Git for a Single Repository**

In AWS CodeCommit, the GitPull IAM policy permissions apply to any Git client command where data is retrieved from AWS CodeCommit, including `git fetch`, `git clone`, and so on. Similarly, the GitPush IAM policy permissions apply to any Git client command where data is sent to AWS CodeCommit. For example, if the GitPush IAM policy permission is set to `Allow`, a user can push the deletion of a branch using the Git protocol. That push is unaffected by any permissions applied to the `DeleteBranch` operation for that IAM user. The `DeleteBranch` permission applies to actions performed with the console, the AWS CLI, the SDKs, and the API, but not the Git protocol.

The following example allows the specified user to pull from, and push to, the AWS CodeCommit repository named MyDemoRepo:

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["codecommit:GitPull", "codecommit:GitPush"],
        }
    ]
}
```
Example 3: Allow a User Connecting From a Specific IP Address Range Access to a Repository

You can create a policy that only allows users to connect to an AWS CodeCommit repository if their IP address is within a certain IP address range. There are two equally valid approaches to this. You can create a Deny policy that disallows AWS CodeCommit operations if the IP address for the user is not within a specific block, or you can create an Allow policy that allows AWS CodeCommit operations if the IP address for the user is within a specific block.

You can create a Deny policy that denies access to all users who are not within a certain IP range. For example, you could attach the AWSCodeCommitPowerUser managed policy and a customer-managed policy to all users who require access to your repository. The following example policy denies all AWS CodeCommit permissions to users whose IP addresses are not within the specified IP address block of 203.0.113.0/16:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": ["codecommit:*"],
            "Resource": "*",
            "Condition": {
                "NotIpAddress": {
                    "aws:SourceIp": ["203.0.113.0/16"]
                }
            }
        }
    ]
}
```

The following example policy allows the specified user to access an AWS CodeCommit repository named MyDemoRepo with the equivalent permissions of the AWSCodeCommitPowerUser managed policy only if their IP address is within the specified address block of 203.0.113.0/16:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Condition": {
                "IpAddress": {
                    "aws:SourceIp": ["203.0.113.0/16"]
                }
            }
        }
    ]
}
```
Customer Managed Integration Policy Examples

This section provides example customer-managed user policies that grant permissions for integrations between AWS CodeCommit and other AWS services. For specific examples of policies that allow cross-account access to an AWS CodeCommit repository, see Configure Cross-Account Access to an AWS CodeCommit Repository (p. 120).

Note
All examples use the US West (Oregon) Region (us-west-2) when a region is required, and contain fictitious account IDs.

Examples

- Example 1: Create a Policy That Enables Cross-Account Access to an Amazon SNS Topic (p. 266)
- Example 2: Create a Policy for AWS Lambda Integration (p. 267)

Example 1: Create a Policy That Enables Cross-Account Access to an Amazon SNS Topic

You can configure an AWS CodeCommit repository so that code pushes or other events trigger actions, such as sending a notification from Amazon Simple Notification Service (Amazon SNS). If you create the Amazon SNS topic with the same account used to create the AWS CodeCommit repository, you do not need to configure additional IAM policies or permissions. You can create the topic, and then create the trigger for the repository. For more information, see Create a Trigger for an Amazon SNS Topic (p. 92).

However, if you want to configure your trigger to use an Amazon SNS topic in another AWS account, you must first configure that topic with a policy that allows AWS CodeCommit to publish to that topic. From that other account, open the Amazon SNS console, choose the topic from the list, and for Other topic actions, choose Edit topic policy. On the Advanced tab, modify the policy for the topic to allow AWS CodeCommit to publish to that topic. For example, if the policy is the default policy, you would modify the policy as follows, changing the items in red italic text to match the values for your repository, Amazon SNS topic, and account:

```json
{
  "Version": "2008-10-17",
  "Id": "__default_policy_ID",
  "Statement": [
    {
      "Sid": "__default_statement_ID",
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Action": [
        "SNS:Subscribe",
        "SNS:ListSubscriptionsByTopic",
        "SNS:DeleteTopic",
        "SNS:GetTopicAttributes",
        "SNS:Publish",
        "SNS:RemovePermission",
        "SNS:AddPermission",
      ]
    }
  ]
}
```
"SNS:Receive",
"SNS:SetTopicAttributes"
],
"Condition": {
"StringEquals": {
"AWS:SourceOwner": "111111111111"
}
},
"Sid": "CodeCommit-Policy_ID",
"Effect": "Allow",
"Principal": {
"Service": "codecommit.amazonaws.com"
},
"Action": "SNS:Publish",
"Condition": {
"StringEquals": {
"AWS:SourceAccount": "80398EXAMPLE"
}
}
"

Example 2: Create a Policy for AWS Lambda Integration

You can configure an AWS CodeCommit repository so that code pushes or other events trigger actions, such as invoking a function in AWS Lambda. For more information, see Create a Trigger for a Lambda Function (p. 97).

If you want your trigger to run a Lambda function directly (instead of using an Amazon SNS topic to invoke the Lambda function), and you do not configure the trigger in the Lambda console, you must include a policy similar to the following in the function’s resource policy:

```
{
  "Statement" : {
    "StatementId" : "Id-1",
    "Action" : "lambda:InvokeFunction",
    "Principal" : "codecommit.amazonaws.com",
    "SourceAccount" : "80398EXAMPLE"
  }
}
```

When manually configuring an AWS CodeCommit trigger that invokes a Lambda function, you must also use the Lambda AddPermission command to grant permission for AWS CodeCommit to invoke the function. For an example, see the To allow AWS CodeCommit to run a Lambda function (p. 100) section of Create a Trigger for an Existing Lambda Function (p. 100).

For more information about resource policies for Lambda functions, see AddPermission and The Pull/Push Event Models in the AWS Lambda Developer Guide.

AWS CodeCommit Permissions Reference

The following tables list each AWS CodeCommit API operation, the corresponding actions for which you can grant permissions, and the format of the resource ARN to use for granting permissions. The AWS
CodeCommit APIs are grouped into tables based on the scope of the actions allowed by that API. Refer to it when setting up Access Control (p. 251) and writing permissions policies that you can attach to an IAM identity (identity-based policies).

When you create a permissions policy, you specify the actions in the policy's Action field. You specify the resource value in the policy's Resource field as an ARN, with or without a wildcard character (*).

To express conditions in your AWS CodeCommit policies, use AWS-wide condition keys. For a complete list of AWS-wide keys, see Available Keys in the IAM User Guide.

**Note**
To specify an action, use the codecommit: prefix followed by the API operation name (for example, codecommit:GetRepository or codecommit:CreateRepository.

**Using Wildcards**

To specify multiple actions or resources, use a wildcard character (*) in your ARN. For example, codecommit:* specifies all AWS CodeCommit actions and codecommit:Get* specifies all AWS CodeCommit actions that begin with the word Get. The following example grants access to all repositories with names that begin with MyDemo.

```
arn:aws:codecommit:us-west-2:111111111111:MyDemo*
```

You can use wildcards only with the repository-name resources listed in the following table. You can’t use wildcards with region or account-id resources. For more information about wildcards, see IAM Identifiers in IAM User Guide.

**Topics**
- Required Permissions for Git Client Commands (p. 268)
- Permissions for Actions on Branches (p. 269)
- Permissions for Actions on Pull Requests (p. 270)
- Permissions for Actions on Individual Files (p. 271)
- Permissions for Actions on Comments (p. 272)
- Permissions for Actions on Committed Code (p. 273)
- Permissions for Actions on Repositories (p. 274)
- Permissions for Actions on Triggers (p. 275)
- Permissions for Actions on AWS CodePipeline Integration (p. 276)

**Required Permissions for Git Client Commands**

In AWS CodeCommit, the GitPull IAM policy permissions apply to any Git client command where data is retrieved from AWS CodeCommit, including git fetch, git clone, and so on. Similarly, the GitPush IAM policy permissions apply to any Git client command where data is sent to AWS CodeCommit. For example, if the GitPush IAM policy permission is set to Allow, a user can push the deletion of a branch using the Git protocol. That push is unaffected by any permissions applied to the DeleteBranch operation for that IAM user. The DeleteBranch permission applies to actions performed with the console, the AWS CLI, the SDKs, and the API, but not the Git protocol.

GitPull and GitPush are IAM policy permissions. They are not API actions.

**AWS CodeCommit Required Permissions for Actions for Git Client Commands**

**GitPull**

**Action(s):** codecommit:GitPull
Permissions for Actions on Branches

The following permissions allow or deny actions on branches in AWS CodeCommit repositories. These permissions pertain only to actions performed in the AWS CodeCommit console and with the AWS CodeCommit API, and to commands performed using the AWS CLI. They do not pertain to similar actions that can be performed using the Git protocol. For example, the `git show-branch -r` command displays a list of remote branches for a repository and its commits using the Git protocol. It's not affected by any permissions for the AWS CodeCommit ListBranches operation.

**AWS CodeCommit API Operations and Required Permissions for Actions on Branches**

**CreateBranch**

- **Action(s):** `codecommit:CreateBranch`
- Required to create a branch in an AWS CodeCommit repository.

  **Resource:** `arn:aws:codecommit:region:account-id:repository-name`

**DeleteBranch**

- **Action(s):** `codecommit:DeleteBranch`
- Required to delete a branch from an AWS CodeCommit repository.

  **Resource:** `arn:aws:codecommit:region:account-id:repository-name`

**GetBranch**

- **Action(s):** `codecommit:GetBranch`
- Required to get details about a branch in an AWS CodeCommit repository.

  **Resource:** `arn:aws:codecommit:region:account-id:repository-name`

**ListBranches**

- **Action(s):** `codecommit:ListBranches`
- Required to get a list of branches in an AWS CodeCommit repository.

  **Resource:** `arn:aws:codecommit:region:account-id:repository-name`

**UpdateDefaultBranch**

- **Action(s):** `codecommit:UpdateDefaultBranch`
- Required to change the default branch in an AWS CodeCommit repository.

  **Resource:** `arn:aws:codecommit:region:account-id:repository-name`
Permissions for Actions on Pull Requests

The following permissions allow or deny actions on pull requests in AWS CodeCommit repositories. These permissions pertain to actions performed with the AWS CodeCommit console and the AWS CodeCommit API, and commands performed using the AWS CLI. They do not pertain to similar actions that can be performed using the Git protocol. For related permissions on comments, see Permissions for Actions on Comments (p. 272).

AWS CodeCommit API Operations and Required Permissions for Actions on Pull Requests

BatchGetPullRequests

**Action(s):** codecommit:BatchGetPullRequests

Required to return information about one or more pull requests in an AWS CodeCommit repository. This is an IAM policy permission only, not an API action that you can call.

**Resource:** arn:aws:codecommit:region:account-id:repository-name

CreatePullRequest

**Action(s):** codecommit:CreatePullRequest

Required to create a pull request in an AWS CodeCommit repository.

**Resource:** arn:aws:codecommit:region:account-id:repository-name

DescribePullRequestEvents

**Action(s):** codecommit:DescribePullRequestEvents

Required to return information about one or more pull request events.

**Resource:** arn:aws:codecommit:region:account-id:repository-name

GetCommentsForPullRequest

**Action(s):** codecommit:GetCommentsForPullRequest

Required to return comments made on a pull request.

**Resource:** arn:aws:codecommit:region:account-id:repository-name

GetCommitsFromMergeBase

**Action(s):** codecommit:GetCommitsFromMergeBase

Required to return information about the difference between commits in the context of a potential merge. This is an IAM policy permission only, not an API action that you can call.

**Resource:** arn:aws:codecommit:region:account-id:repository-name

GetMergeConflicts

**Action(s):** codecommit:GetMergeConflicts

Required to return information about merge conflicts between the source and destination branch in a pull request.

**Resource:** arn:aws:codecommit:region:account-id:repository-name

GetPullRequest

**Action(s):** codecommit:GetPullRequest
Permissions for Actions on Individual Files

The following permissions allow or deny actions on individual files in AWS CodeCommit repositories. These permissions pertain only to actions performed in the AWS CodeCommit console, the AWS CodeCommit API, and to commands performed using the AWS CLI. They do not pertain to similar actions that can be performed using the Git protocol. For example, the `git push` command pushes new and changed files to an AWS CodeCommit repository by using the Git protocol. It's not affected by any permissions for the AWS CodeCommit PutFile operation.

## Permissions for Actions on Individual Files

- **ListPullRequests**
  - **Action(s)**: `codecommit:ListPullRequests`
  - Required to list pull requests in a repository.

- **MergePullRequestByFastForward**
  - **Action(s)**: `codecommit:MergePullRequestByFastForward`
  - Required to close a pull request and attempt to merge the source branch into the destination branch of a pull request using the fast-forward merge option.

- **PostCommentForPullRequest**
  - **Action(s)**: `codecommit:PostCommentForPullRequest`
  - Required to post a comment on a pull request in an AWS CodeCommit repository.

- **UpdatePullRequestDescription**
  - **Action(s)**: `codecommit:UpdatePullRequestDescription`
  - Required to change the description of a pull request in an AWS CodeCommit repository.

- **UpdatePullRequestStatus**
  - **Action(s)**: `codecommit:UpdatePullRequestStatus`
  - Required to change the status of a pull request in an AWS CodeCommit repository.

- **UpdatePullRequestTitle**
  - **Action(s)**: `codecommit:UpdatePullRequestTitle`
  - Required to change the title of a pull request in an AWS CodeCommit repository.
Permissions for Actions on Comments

The following permissions allow or deny actions on comments in AWS CodeCommit repositories. These permissions pertain to actions performed with the AWS CodeCommit console and the AWS CodeCommit API, and to commands performed using the AWS CLI. For related permissions on comments in pull requests, see Permissions for Actions on Pull Requests (p. 270).

AWS CodeCommit API Operations and Required Permissions for Actions on Repositories

DeleteCommentContent

**Action(s):** codecommit:DeleteCommentContent

Required to delete the content of a comment made on a change, file, or commit in a repository. Comments cannot be deleted, but the content of a comment can be removed if the user has this permission.

**Resource:** arn:aws:codecommit:region:account-id:repository-name

GetComment

**Action(s):** codecommit:GetComment

Required to return information about a comment made on a change, file, or commit in an AWS CodeCommit repository.

**Resource:** arn:aws:codecommit:region:account-id:repository-name

GetCommentsForComparedCommit

**Action(s):** codecommit:GetCommentsForComparedCommit

Required to return information about comments made on the comparison between two commits in an AWS CodeCommit repository.

**Resource:** arn:aws:codecommit:region:account-id:repository-name

PostCommentForComparedCommit

**Action(s):** codecommit:PostCommentForComparedCommit

Required to comment on the comparison between two commits in an AWS CodeCommit repository.

**Resource:** arn:aws:codecommit:region:account-id:repository-name
Permissions for Actions on Committed Code

The following permissions allow or deny actions on code committed to AWS CodeCommit repositories. These permissions pertain to actions performed with the AWS CodeCommit console and the AWS CodeCommit API, and commands performed using the AWS CLI. They do not pertain to similar actions that can be performed using the Git protocol. For related permissions on comments on committed code, see Permissions for Actions on Comments (p. 272).

Explicitly denying some of these permissions might result in unexpected consequences in the AWS CodeCommit console. For example, setting GetTree to Deny prevents users from navigating the contents of a repository in the console, but does not block users from viewing the contents of a file in the repository (if they are sent a link to the file in email, for example). Setting GetBlob to Deny prevents users from viewing the contents of files, but does not block users from browsing the structure of a repository. Setting GetObjectIdentifier to Deny blocks most of the functionality of code browsing. If you set all three of these actions to Deny in a policy, a user with that policy cannot browse code in the AWS CodeCommit console.

AWS CodeCommit API Operations and Required Permissions for Actions on Committed Code

BatchGetCommits

Action(s): codecommit:BatchGetCommits

Required to return information about one or more commits in an AWS CodeCommit repository. This is an IAM policy permission only, not an API action that you can call.


GetCommit

Action(s): codecommit:GetCommit

Required to return information about a commit.


GetCommitHistory

Action(s): codecommit:GetCommitHistory

Required to return information about the history of commits in a repository. This is an IAM policy permission only, not an API action that you can call.
Permissions for Actions on Repositories

The following permissions allow or deny actions on AWS CodeCommit repositories. These permissions pertain to actions performed with the AWS CodeCommit console and the AWS CodeCommit API, and to commands performed using the AWS CLI. They do not pertain to similar actions that can be performed using the Git protocol.

AWS CodeCommit API Operations and Required Permissions for Actions on Repositories

BatchGetRepositories

Action(s): codecommit:BatchGetRepositories

Required to get information about multiple AWS CodeCommit repositories in that are in an AWS account. In Resource, you must specify the names of all of the AWS CodeCommit repositories for which a user is allowed (or denied) information.


CreateRepository

Action(s): codecommit:CreateRepository

Required to create an AWS CodeCommit repository.

Permissions for Actions on Triggers

The following permissions allow or deny actions on triggers for AWS CodeCommit repositories.

AWS CodeCommit API Operations and Required Permissions for Actions on Triggers

GetRepositoryTriggers

**Action(s):** codecommit:GetRepositoryTriggers

Required to return information about triggers configured for a repository.

**Resource:** arn:aws:codecommit:region:account-id:repository-name

PutRepositoryTriggers

**Action(s):** codecommit:PutRepositoryTriggers

Required to create, edit, or delete triggers for a repository.

**Resource:** arn:aws:codecommit:region:account-id:repository-name
Permissions for Actions on AWS CodePipeline Integration

In order for AWS CodePipeline to use an AWS CodeCommit repository in a source action for a pipeline, you must grant all of the permissions listed in the following table to the service role for AWS CodePipeline. If these permissions are not set in the service role or are set to Deny, the pipeline does not run automatically when a change is made to the repository, and changes cannot be released manually.

AWS CodeCommit API Operations and Required Permissions for Actions on AWS CodePipeline Integration

GetBranch

Action(s): codecommit:GetBranch

Required to get details about a branch in an AWS CodeCommit repository.


GetCommit

Action(s): codecommit:GetCommit

Required to return information about a commit.


UploadArchive

Action(s): codecommit:UploadArchive

Required to allow the service role for AWS CodePipeline to upload repository changes into a pipeline. This is an IAM policy permission only, not an API action that you can call.


GetUploadArchiveStatus

Action(s): codecommit:GetUploadArchiveStatus

Required to determine the status of an archive upload: whether it is in progress, complete, cancelled, or if an error occurred. This is an IAM policy permission only, not an API action that you can call.


CancelUploadArchive

Action(s): codecommit:CancelUploadArchive

Required to cancel the uploading of an archive to a pipeline. This is an IAM policy permission only, not an API action that you can call.
Regions and Git Connection Endpoints for AWS CodeCommit

Each AWS CodeCommit repository is associated with an AWS region. AWS CodeCommit offers regional endpoints to make your requests to the service. In addition, AWS CodeCommit provides Git connection endpoints for both SSH and HTTPS protocols in every region where AWS CodeCommit is available.

All the examples in this guide use the same endpoint URL for Git in US East (Ohio): git-codecommit.us-east-2.amazonaws.com. However, when you use Git and configure your connections, make sure you choose the Git connection endpoint that matches the region that hosts your AWS CodeCommit repository. For example, if you want to make a connection to a repository in US East (N. Virginia), use the endpoint URL of git-codecommit.us-east-1.amazonaws.com. This is also true for API calls. When you make connections to an AWS CodeCommit repository with the AWS CLI or the SDKs, make sure you use the correct regional endpoint for the repository.

Supported Regions for AWS CodeCommit

You can create and use AWS CodeCommit repositories in the following AWS regions:

- US East (Ohio)
- US East (N. Virginia)
- US West (N. California)
- US West (Oregon)
- EU (Ireland)
- EU (London)
- EU (Paris)
- EU (Frankfurt)
Git Connection Endpoints

For more information about regional endpoints for AWS CLI, service, and API calls to AWS CodeCommit, see AWS Regions and Endpoints.

Git Connection Endpoints

Use the following URLs when you configure Git connections to AWS CodeCommit repositories:

Git connection endpoints for AWS CodeCommit

<table>
<thead>
<tr>
<th>Region Name</th>
<th>Region</th>
<th>Endpoint URL</th>
<th>Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>US East (Ohio)</td>
<td>us-east-2</td>
<td><a href="https://git-codecommit.us-east-2.amazonaws.com">https://git-codecommit.us-east-2.amazonaws.com</a></td>
<td>HTTPS</td>
</tr>
<tr>
<td>US East (Ohio)</td>
<td>us-east-2</td>
<td>ssh://git-codecommit.us-east-2.amazonaws.com</td>
<td>SSH</td>
</tr>
<tr>
<td>US East (N. Virginia)</td>
<td>us-east-1</td>
<td><a href="https://git-codecommit.us-east-1.amazonaws.com">https://git-codecommit.us-east-1.amazonaws.com</a></td>
<td>HTTPS</td>
</tr>
<tr>
<td>US East (N. Virginia)</td>
<td>us-east-1</td>
<td>ssh://git-codecommit.us-east-1.amazonaws.com</td>
<td>SSH</td>
</tr>
<tr>
<td>US West (Oregon)</td>
<td>us-west-2</td>
<td><a href="https://git-codecommit.us-west-2.amazonaws.com">https://git-codecommit.us-west-2.amazonaws.com</a></td>
<td>HTTPS</td>
</tr>
<tr>
<td>US West (Oregon)</td>
<td>us-west-2</td>
<td>ssh://git-codecommit.us-west-2.amazonaws.com</td>
<td>SSH</td>
</tr>
<tr>
<td>EU (Ireland)</td>
<td>eu-west-1</td>
<td><a href="https://git-codecommit.eu-west-1.amazonaws.com">https://git-codecommit.eu-west-1.amazonaws.com</a></td>
<td>HTTPS</td>
</tr>
<tr>
<td>EU (Ireland)</td>
<td>eu-west-1</td>
<td>ssh://git-codecommit.eu-west-1.amazonaws.com</td>
<td>SSH</td>
</tr>
<tr>
<td>Asia Pacific (Tokyo)</td>
<td>ap-northeast-1</td>
<td><a href="https://git-codecommit.ap-northeast-1.amazonaws.com">https://git-codecommit.ap-northeast-1.amazonaws.com</a></td>
<td>HTTPS</td>
</tr>
<tr>
<td>Asia Pacific (Tokyo)</td>
<td>ap-northeast-1</td>
<td>ssh://git-codecommit.ap-northeast-1.amazonaws.com</td>
<td>SSH</td>
</tr>
<tr>
<td>Asia Pacific (Singapore)</td>
<td>ap-southeast-1</td>
<td><a href="https://git-codecommit.ap-southeast-1.amazonaws.com">https://git-codecommit.ap-southeast-1.amazonaws.com</a></td>
<td>HTTPS</td>
</tr>
</tbody>
</table>
## Git Connection Endpoints

<table>
<thead>
<tr>
<th>Region Name</th>
<th>Region</th>
<th>Endpoint URL</th>
<th>Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>Asia Pacific (Singapore)</td>
<td>ap-southeast-1</td>
<td>ssh://git-codecommit.ap-southeast-1.amazonaws.com</td>
<td>SSH</td>
</tr>
<tr>
<td>Asia Pacific (Sydney)</td>
<td>ap-southeast-2</td>
<td><a href="https://git-codecommit.ap-southeast-2.amazonaws.com">https://git-codecommit.ap-southeast-2.amazonaws.com</a></td>
<td>HTTPS</td>
</tr>
<tr>
<td>Asia Pacific (Sydney)</td>
<td>ap-southeast-2</td>
<td>ssh://git-codecommit.ap-southeast-2.amazonaws.com</td>
<td>SSH</td>
</tr>
<tr>
<td>EU (Frankfurt)</td>
<td>eu-central-1</td>
<td><a href="https://git-codecommit.eu-central-1.amazonaws.com">https://git-codecommit.eu-central-1.amazonaws.com</a></td>
<td>HTTPS</td>
</tr>
<tr>
<td>EU (Frankfurt)</td>
<td>eu-central-1</td>
<td>ssh://git-codecommit.eu-central-1.amazonaws.com</td>
<td>SSH</td>
</tr>
<tr>
<td>Asia Pacific (Seoul)</td>
<td>ap-northeast-2</td>
<td><a href="https://git-codecommit.ap-northeast-2.amazonaws.com">https://git-codecommit.ap-northeast-2.amazonaws.com</a></td>
<td>HTTPS</td>
</tr>
<tr>
<td>Asia Pacific (Seoul)</td>
<td>ap-northeast-2</td>
<td>ssh://git-codecommit.ap-northeast-2.amazonaws.com</td>
<td>SSH</td>
</tr>
<tr>
<td>South America (São Paulo)</td>
<td>sa-east-1</td>
<td><a href="https://git-codecommit.sa-east-1.amazonaws.com">https://git-codecommit.sa-east-1.amazonaws.com</a></td>
<td>HTTPS</td>
</tr>
<tr>
<td>South America (São Paulo)</td>
<td>sa-east-1</td>
<td>ssh://git-codecommit.sa-east-1.amazonaws.com</td>
<td>SSH</td>
</tr>
<tr>
<td>US West (N. California)</td>
<td>us-west-1</td>
<td><a href="https://git-codecommit.us-west-1.amazonaws.com">https://git-codecommit.us-west-1.amazonaws.com</a></td>
<td>HTTPS</td>
</tr>
<tr>
<td>US West (N. California)</td>
<td>us-west-1</td>
<td>ssh://git-codecommit.us-west-1.amazonaws.com</td>
<td>SSH</td>
</tr>
<tr>
<td>EU (London)</td>
<td>eu-west-2</td>
<td><a href="https://git-codecommit.eu-west-2.amazonaws.com">https://git-codecommit.eu-west-2.amazonaws.com</a></td>
<td>HTTPS</td>
</tr>
<tr>
<td>EU (London)</td>
<td>eu-west-2</td>
<td>ssh://git-codecommit.eu-west-2.amazonaws.com</td>
<td>SSH</td>
</tr>
<tr>
<td>Asia Pacific (Mumbai)</td>
<td>ap-south-1</td>
<td><a href="https://git-codecommit.ap-south-1.amazonaws.com">https://git-codecommit.ap-south-1.amazonaws.com</a></td>
<td>HTTPS</td>
</tr>
<tr>
<td>Asia Pacific (Mumbai)</td>
<td>ap-south-1</td>
<td>ssh://git-codecommit.ap-south-1.amazonaws.com</td>
<td>SSH</td>
</tr>
<tr>
<td>Canada (Central)</td>
<td>ca-central-1</td>
<td><a href="https://git-codecommit.ca-central-1.amazonaws.com">https://git-codecommit.ca-central-1.amazonaws.com</a></td>
<td>HTTPS</td>
</tr>
<tr>
<td>Canada (Central)</td>
<td>ca-central-1</td>
<td>ssh://git-codecommit.ca-central-1.amazonaws.com</td>
<td>SSH</td>
</tr>
</tbody>
</table>
## Server Fingerprints for AWS CodeCommit

The following table lists the public fingerprints for Git connection endpoints in AWS CodeCommit. These server fingerprints are displayed as part of the verification process for adding an endpoint to your known hosts file.

### Public fingerprints for AWS CodeCommit

<table>
<thead>
<tr>
<th>Server</th>
<th>Cryptographic hash type</th>
<th>Fingerprint</th>
</tr>
</thead>
<tbody>
<tr>
<td>git-codecommit.us-east-2.amazonaws.com</td>
<td>SHA256</td>
<td>31BlW2g5xn/NA2Ck6dyeJIRQOWvn7n8UEs56fG6ZiZQ</td>
</tr>
</tbody>
</table>
### Limits in AWS CodeCommit

The following table describes limits in AWS CodeCommit. For information about limits that can be changed, see [AWS Service Limits](#).

<table>
<thead>
<tr>
<th>Server</th>
<th>Cryptographic hash type</th>
<th>Fingerprint</th>
</tr>
</thead>
<tbody>
<tr>
<td>git-codecommit.ap-southeast-2.amazonaws.com</td>
<td>SHA256</td>
<td>nYp+gHas80Y3Dqbp4yanCDFhqDVjseefVbHEXq</td>
</tr>
<tr>
<td>git-codecommit.eu-central-1.amazonaws.com</td>
<td>SHA256</td>
<td>MwGrkiEki8QkkBtlAgXbyto2YBnZf62YV5</td>
</tr>
<tr>
<td>git-codecommit.ap-northeast-2.amazonaws.com</td>
<td>SHA256</td>
<td>eegAPqRwY9YsYo92HIQmxetfXBHzA2d8Eys</td>
</tr>
<tr>
<td>git-codecommit.sa-east-1.amazonaws.com</td>
<td>SHA256</td>
<td>kW+VKb0jpRaG/ZbXkgbtMqbRgEDK7nISV3SVoyCmzu</td>
</tr>
<tr>
<td>git-codecommit.us-west-1.amazonaws.com</td>
<td>SHA256</td>
<td>gzauWTWXDK2u5KumMi5vbKtmyerdlwgSzY</td>
</tr>
<tr>
<td>git-codecommit.eu-west-2.amazonaws.com</td>
<td>SHA256</td>
<td>r0Rwz5k/IHp/QyrRnfiM9j02D5UEqMbtFNTuDG2hNbs</td>
</tr>
<tr>
<td>git-codecommit.ap-south-1.amazonaws.com</td>
<td>SHA256</td>
<td>hUKwnTj7+Xpx4Ktdd6p45j4Raz1J4hAMD8k</td>
</tr>
<tr>
<td>git-codecommit.ca-central-1.amazonaws.com</td>
<td>SHA256</td>
<td>Qz5puafQdANVprLl6j60Qyh41CNsF6ob61D0</td>
</tr>
<tr>
<td>git-codecommit.eu-west-3.amazonaws.com</td>
<td>SHA256</td>
<td>uw7c2FL564jVoFgtc+ikZlKnKbsZzt9+CFdSjJkbli</td>
</tr>
</tbody>
</table>
## Limits

<table>
<thead>
<tr>
<th><strong>Number of repositories</strong></th>
<th>Maximum of 1,000 per AWS account. This limit can be changed. For more information, see <a href="https://aws.amazon.com/service-quotas/">AWS Service Limits</a>.</th>
</tr>
</thead>
</table>
| **Regions**                | AWS CodeCommit is available in the following regions:  
  - US East (Ohio)  
  - US East (N. Virginia)  
  - US West (N. California)  
  - US West (Oregon)  
  - EU (Ireland)  
  - EU (London)  
  - EU (Paris)  
  - EU (Frankfurt)  
  - Asia Pacific (Tokyo)  
  - Asia Pacific (Singapore)  
  - Asia Pacific (Sydney)  
  - Asia Pacific (Seoul)  
  - Asia Pacific (Mumbai)  
  - South America (São Paulo)  
  - Canada (Central)  |
| **Number of references in a single push** | Maximum of 4,000, including create, delete, and update. There is no limit on the overall number of references in the repository. |
| **Number of triggers in a repository** | Maximum of 10. |
| **Repository names**       | Any combination of letters, numbers, periods, underscores, and dashes between 1 and 100 characters in length. Repository names cannot end in .git and cannot contain any of the following characters: ! ? @ # % ^ & * ( ) + = { } | \ / > < ~ ` " ; : |
| **Branch names**           | Any combination of allowed characters between 1 and 256 characters in length. Branch names cannot:  
  - begin or end with a slash (/) or period (.)  
  - consist of the single character @  
  - contain two or more consecutive periods (..), slashes (//), or the following character combination: @{  
  - contain spaces or any of the following characters: ? ^ * [ \ ~ : |
<table>
<thead>
<tr>
<th>Limits</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Branch names are references. Many of the limitations on branch names</td>
<td>Branch names are references. Many of the limitations on branch names are based on the Git reference standard. For more information, see Git Internals and git-check-ref-format.</td>
</tr>
<tr>
<td>names are based on the Git reference standard. For more information,</td>
<td></td>
</tr>
<tr>
<td>see Git Internals and git-check-ref-format.</td>
<td></td>
</tr>
<tr>
<td>Trigger names</td>
<td>Any combination of letters, numbers, periods, underscores, and</td>
</tr>
<tr>
<td></td>
<td>dashes between 1 and 100 characters in length. Trigger names</td>
</tr>
<tr>
<td></td>
<td>cannot contain spaces or commas.</td>
</tr>
<tr>
<td>User names in commits made in the console</td>
<td>Any combination of allowed characters between 1 and 1,024</td>
</tr>
<tr>
<td></td>
<td>characters in length.</td>
</tr>
<tr>
<td>Email addresses in commits made in the console</td>
<td>Any combination of allowed characters between 1 and 256</td>
</tr>
<tr>
<td></td>
<td>characters in length. Email addresses are not validated.</td>
</tr>
<tr>
<td>Repository descriptions</td>
<td>Any combination of characters between 0 and 1,000 characters</td>
</tr>
<tr>
<td></td>
<td>in length. Repository descriptions are optional.</td>
</tr>
<tr>
<td>Metadata for a commit</td>
<td>Maximum of 20 MB for the combined metadata for a commit (for</td>
</tr>
<tr>
<td></td>
<td>example, the combination of author information, date, parent</td>
</tr>
<tr>
<td></td>
<td>commit list, and commit messages).</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong></td>
</tr>
<tr>
<td></td>
<td>There is no limit on the number or the total size of all files</td>
</tr>
<tr>
<td></td>
<td>in a single commit, as long as the metadata does not exceed 2</td>
</tr>
<tr>
<td></td>
<td>MB and a single blob does not exceed 2 GB.</td>
</tr>
<tr>
<td>File size display limit in the AWS CodeCommit console</td>
<td>Maximum of 6 MB.</td>
</tr>
</tbody>
</table>
### File paths

Any combination of allowed characters between 1 and 4,096 characters in length. File paths must be an unambiguous name that specifies the file and the exact location of the file. File paths cannot exceed 20 directories in depth. Additionally, file paths cannot:

- contain empty strings
- be a relative file path
- include any of the following character combinations:
  - `./`\n  - `/../`
  - `//`
- end with a trailing slash or backslash

Filenames and paths must be fully qualified. The name and path to a file on your local computer must follow the standards for that operating system. When specifying the path to a file in an AWS CodeCommit repository, use the standards for Amazon Linux.

### Git blob size

Maximum of 2 GB.

**Note**

There is no limit on the number or the total size of all files in a single commit, as long as the metadata does not exceed 6 MB and a single blob does not exceed 2 GB.

### Custom data for triggers

This is a string field limited to 1,000 characters. It cannot be used to pass any dynamic parameters.

### Graph display of branches in the Commit Visualizer

35 per page. If there are more than 35 branches on a single page, the graph is not displayed.

## Temporary Access to AWS CodeCommit Repositories

You can allow users temporary access to your AWS CodeCommit repositories. For example, you might do this to allow IAM users to access AWS CodeCommit repositories in separate AWS accounts (a technique known as *cross-account access*). For a walkthrough of configuring cross-account access to a repository, see [Configure Cross-Account Access to an AWS CodeCommit Repository](p. 120).

You can also configure access for users who want or must authenticate through methods such as:

- Security Assertion Markup Language (SAML)
- Multi-factor authentication (MFA)
- Federation
• Login with Amazon
• Amazon Cognito
• Facebook
• Google
• OpenID Connect (OIDC)-compatible identity provider

**Note**
The following information applies only to the use of the AWS CLI credential helper to connect to AWS CodeCommit repositories. You cannot use SSH or Git credentials and HTTPS to connect to AWS CodeCommit repositories with temporary access credentials.

To give users temporary access to your AWS CodeCommit repositories, complete the following steps.

Do not complete these steps if all of the following requirements are true:

• You are signed in to an Amazon EC2 instance.
• You are using Git and HTTPS with the AWS CLI credential helper to connect from the Amazon EC2 instance to AWS CodeCommit repositories.
• The Amazon EC2 instance has an attached IAM instance profile that contains the access permissions described in For HTTPS Connections on Linux, macOS, or Unix with the AWS CLI Credential Helper (p. 35) or For HTTPS Connections on Windows with the AWS CLI Credential Helper (p. 39).
• You have installed and configured the Git credential helper on the Amazon EC2 instance, as described in For HTTPS Connections on Linux, macOS, or Unix with the AWS CLI Credential Helper (p. 35) or For HTTPS Connections on Windows with the AWS CLI Credential Helper (p. 39).

Amazon EC2 instances that meet the preceding requirements are already set up to communicate temporary access credentials to AWS CodeCommit on your behalf.

### Step 1: Complete the Prerequisites

Complete the setup steps to provide a user with temporary access to your AWS CodeCommit repositories:

• For cross-account access, see Walkthrough: Delegating Access Across AWS Accounts Using IAM Roles.
• For SAML and federation, see Using Your Organization's Authentication System to Grant Access to AWS Resources and About AWS STS SAML 2.0-based Federation.
• For MFA, see Using Multi-Factor Authentication (MFA) Devices with AWS and Creating Temporary Security Credentials to Enable Access for IAM Users.
• For Login with Amazon, Amazon Cognito, Facebook, Google, or any OIDC-compatible identity provider, see About AWS STS Web Identity Federation.

Use the information in Authentication and Access Control for AWS CodeCommit (p. 250) to specify the AWS CodeCommit permissions you want to temporarily grant the user.

### Step 2: Get Temporary Access Credentials

Depending on the way you set up temporary access, your user can get temporary access credentials in one of the following ways:

• For cross-account access, call the AWS CLI `assume-role` command or call the AWS STS AssumeRole API.
• For SAML, call the AWS CLI `assume-role-with-saml` command or the AWS STS AssumeRoleWithSAML API.
• For federation, call the AWS CLI assume-role or get-federation-token commands or the AWS STS AssumeRole or GetFederationToken APIs.

• For MFA, call the AWS CLI get-session-token command or the AWS STS GetSessionToken API.

• For Login with Amazon, Amazon Cognito, Facebook, Google, or any OIDC-compatible identity provider, call the AWS CLI assume-role-with-web-identity command or the AWS STS AssumeRoleWithWebIdentity API.

Your user should receive a set of temporary access credentials, which include an AWS access key ID, a secret access key, and a session token. Your user should make a note of these three values because they are used in the next step.

### Step 3: Configure the AWS CLI with Your Temporary Access Credentials

Your user must configure the development machine to use those temporary access credentials.

1. Follow the instructions in Setting Up (p. 4) to set up the AWS CLI. Use the `aws configure` command to configure a profile.

   **Note**
   Before you continue, make sure the git config file is configured to use the AWS profile you configured in the AWS CLI.

2. You can associate the temporary access credentials with the user's AWS CLI named profile in one of the following ways. Do not use the `aws configure` command.

   • In the `~/.aws/credentials` file (for Linux) or the `%UserProfile%.aws\credentials` file (for Windows), add to the user's AWS CLI named profile the `aws_access_key_id`, `aws_secret_access_key`, and `aws_session_token` setting values:

     ```
     [CodeCommitProfileName]
     aws_access_key_id=TheAccessKeyID
     aws_secret_access_key=TheSecretAccessKey
     aws_session_token=TheSessionToken
     ```

   -OR-

   • Set the `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`, and `AWS_SESSION_TOKEN` environment variables:

     For Linux, macOS, or Unix:

     ```
     export AWS_ACCESS_KEY_ID=TheAccessKey
     export AWS_SECRET_ACCESS_KEY=TheSecretAccessKey
     export AWS_SESSION_TOKEN=TheSessionToken
     ```

     For Windows:

     ```
     set AWS_ACCESS_KEY_ID=TheAccessKey
     set AWS_SECRET_ACCESS_KEY=TheSecretAccessKey
     set AWS_SESSION_TOKEN=TheSessionToken
     ```

     For more information, see Configuring the AWS Command Line Interface in the AWS Command Line Interface User Guide.

API Version 2015-04-13

287
3. Set up the Git credential helper with the AWS CLI named profile associated with the temporary access credentials.
   - Linux, macOS, or Unix (p. 35)
   - Windows (p. 39)

   As you follow these steps, do not call the `aws configure` command. You already specified temporary access credentials through the credentials file or the environment variables. If you use environment variables instead of the credentials file, in the Git credential helper, specify `default` as the profile name.

**Step 4: Access the AWS CodeCommit Repositories**

Assuming your user has followed the instructions in Connect to a Repository (p. 78) to connect to the AWS CodeCommit repositories, the user then uses Git to call `git clone`, `git push`, and `git pull` to clone, push to, and pull from, the AWS CodeCommit repositories to which he or she has temporary access.

When the user uses the AWS CLI and specifies the AWS CLI named profile associated with the temporary access credentials, results scoped to that profile are returned.

If the user receives the `403: Forbidden` error in response to calling a Git command or a command in the AWS CLI, it’s likely the temporary access credentials have expired. The user must go back to step 2 (p. 286) and get a new set of temporary access credentials.

**AWS Key Management Service and Encryption for AWS CodeCommit Repositories**

Data in AWS CodeCommit repositories is encrypted in transit and at rest. When data is pushed into an AWS CodeCommit repository (for example, by calling `git push`), AWS CodeCommit encrypts the received data as it is stored in the repository. When data is pulled from an AWS CodeCommit repository (for example, by calling `git pull`), AWS CodeCommit decrypts the data and then sends it to the caller. This assumes the IAM user associated with the push or pull request has been authenticated by AWS. Data sent or received is transmitted using the HTTPS or SSH encrypted network protocols.

The first time you create an AWS CodeCommit repository in a new region in your AWS account, AWS CodeCommit creates an AWS-managed key in that same region in AWS Key Management Service (AWS KMS) that is used only by AWS CodeCommit (the `aws/codecommit` key). This key is created and stored in your AWS account. AWS CodeCommit uses this AWS-managed key to encrypt and decrypt the data in this and all other AWS CodeCommit repositories within that region in your AWS account.

**Important**

AWS CodeCommit performs the following AWS KMS actions against the default key `aws/codecommit`. An IAM user does not need explicit permissions for these actions, but the user must not have any attached policies that deny these actions for the `aws/codecommit` key. Specifically, your AWS account must not have any of the following permissions set to deny when creating your first repository:

- "kms:Encrypt"
- "kms:Decrypt"
- "kms:ReEncrypt"
- "kms:GenerateDataKey"
- "kms:GenerateDataKeyWithoutPlaintext"
Encryption Context

Each service integrated with AWS KMS specifies an encryption context for both the encryption and decryption operations. The encryption context is additional authenticated information AWS KMS uses to check for data integrity. When specified for the encryption operation, it must also be specified in the decryption operation or decryption will fail. AWS CodeCommit uses the AWS CodeCommit repository ID for the encryption context. You can find the repository ID by using the `get-repository` command or by viewing repository details in the AWS CodeCommit console. Search for the AWS CodeCommit repository ID in AWS CloudTrail logs to understand which encryption operations were taken on which key in AWS KMS to encrypt or decrypt data in the AWS CodeCommit repository.

For more information about AWS KMS, see the AWS Key Management Service Developer Guide.

Logging AWS CodeCommit API Calls with AWS CloudTrail

AWS CodeCommit is integrated with CloudTrail, a service that captures all of the AWS CodeCommit API calls and delivers the log files to an Amazon S3 bucket that you specify. CloudTrail captures API calls from the AWS CodeCommit console, your Git client, and from code calls to the AWS CodeCommit APIs. Using the information collected by CloudTrail, you can determine the request that was made to AWS CodeCommit, the source IP address from which the request was made, who made the request, when it was made, and so on.

To learn more about CloudTrail, including how to configure and enable it, see the AWS CloudTrail User Guide.

AWS CodeCommit Information in CloudTrail

When CloudTrail logging is enabled in your AWS account, API calls made to AWS CodeCommit actions are tracked in CloudTrail log files, where they are written with other AWS service records. CloudTrail determines when to create and write to a new file based on a time period and file size.

All AWS CodeCommit actions are logged by CloudTrail, including some (such as `GetObjectIdentifier`) that are not currently documented in the AWS CodeCommit API Reference but are instead referenced as access permissions and documented in AWS CodeCommit Permissions.
For example, calls to the `ListRepositories` (in the AWS CLI, `aws codecommit list-repositories`), `CreateRepository` (`aws codecommit create-repository`) and `PutRepositoryTriggers` (`aws codecommit put-repository-triggers`) actions generate entries in the CloudTrail log files, as well as Git client calls to `GitPull` and `GitPush`. In addition, if you have an AWS CodeCommit repository configured as a source for a pipeline in AWS CodePipeline, you will see calls to AWS CodeCommit access permission actions such as `UploadArchive` from AWS CodePipeline. Since AWS CodeCommit uses AWS Key Management Service to encrypt and decrypt repositories, you will also see calls from AWS CodeCommit to `Encrypt` and `Decrypt` actions from AWS KMS in CloudTrail logs.

Every log entry contains information about who generated the request. The user identity information in the log entry helps you determine the following:

- Whether the request was made with root or IAM user credentials
- Whether the request was made with temporary security credentials for a role or federated user, or made by an assumed role
- Whether the request was made by another AWS service

For more information, see the CloudTrail `userIdentity` element.

You can store your log files in your Amazon S3 bucket for as long as you want, but you can also define Amazon S3 lifecycle rules to archive or delete log files automatically. By default, your log files are encrypted with Amazon S3 server-side encryption (SSE).

If you want to be notified upon log file delivery, you can configure CloudTrail to publish Amazon SNS notifications when new log files are delivered. For more information, see Configuring Amazon SNS Notifications for CloudTrail.

You can also aggregate AWS CodeCommit log files from multiple AWS regions and multiple AWS accounts into a single Amazon S3 bucket.

For more information, see Receiving CloudTrail Log Files from Multiple Regions and Receiving CloudTrail Log Files from Multiple Accounts.

Understanding AWS CodeCommit Log File Entries

CloudTrail log files can contain one or more log entries. Each entry lists multiple JSON-formatted events. A log event represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. Log entries are not an ordered stack trace of the public API calls, so they do not appear in any specific order.

**Note**

This example has been formatted for improved readability. In a CloudTrail log file, all entries and events are concatenated into a single line. In addition, this example has been limited to a single AWS CodeCommit entry. In a real CloudTrail log file, you will see entries and events from multiple AWS services.

**Contents**

- Example: A log entry for listing AWS CodeCommit repositories (p. 290)
- Example: A log entry for creating an AWS CodeCommit repository (p. 291)
- Examples: Log entries for pull requests to an AWS CodeCommit repository (p. 292)
- Example: A log entry for a successful push to an AWS CodeCommit repository (p. 293)

**Example: A log entry for listing AWS CodeCommit repositories**

The following example shows a CloudTrail log entry that demonstrates the `ListRepositories` action.
Note
Although ListRepositories returns a list of repositories, non-mutable responses are not recorded in CloudTrail logs, so responseElements is shown as null in the log file.

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AIDACKCEVSQ6C2EXAMPLE",
    "arn": "arn:aws:iam::444455556666:user/Mary_Major",
    "accountId": "444455556666",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "userName": "Mary_Major"
  },
  "eventTime": "2016-12-14T17:57:36Z",
  "eventSource": "codecommit.amazonaws.com",
  "eventName": "ListRepositories",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "203.0.113.12",
  "userAgent": "aws-cli/1.10.53 Python/2.7.9 Windows/8 botocore/1.4.43",
  "requestParameters": null,
  "responseElements": null,
  "requestID": "cb8c167e-EXAMPLE",
  "eventID": "e3c6f4ce-EXAMPLE",
  "readOnly": true,
  "eventType": "AwsApiCall",
  "apiVersion": "2015-04-13",
  "recipientAccountId": "444455556666"
}
```

Example: A log entry for creating an AWS CodeCommit repository

The following example shows a CloudTrail log entry that demonstrates the CreateRepository action in the US East (Ohio) Region region.

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AIDACKCEVSQ6C2EXAMPLE",
    "arn": "arn:aws:iam::444455556666:user/Mary_Major",
    "accountId": "444455556666",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "userName": "Mary_Major"
  },
  "eventTime": "2016-12-14T18:19:15Z",
  "eventSource": "codecommit.amazonaws.com",
  "eventName": "CreateRepository",
  "awsRegion": "us-east-2",
  "sourceIPAddress": "203.0.113.12",
  "userAgent": "aws-cli/1.10.53 Python/2.7.9 Windows/8 botocore/1.4.43",
  "requestParameters": {
    "repositoryDescription": "Creating a demonstration repository.",
    "repositoryName": "MyDemoRepo"
  },
  "responseElements": {
    "repositoryMetadata": {
      "creationDate": "Dec 14, 2016 6:19:14 PM",
      "repositoryId": "8afe792d-EXAMPLE",
      "cloneUrlSsh": "ssh://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo",
```

API Version 2015-04-13
291
Examples: Log entries for pull requests to an AWS CodeCommit repository

The following example shows a CloudTrail log entry that demonstrates the GitPull action where the local repo is already up-to-date.

```json
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE",
        "arn": "arn:aws:iam::444455556666:user/Mary_Major",
        "accountId": "444455556666",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "userName": "Mary_Major"
    },
    "eventTime": "2016-12-14T18:19:15Z",
    "eventSource": "codecommit.amazonaws.com",
    "eventName": "GitPull",
    "awsRegion": "us-east-2",
    "sourceIPAddress": "203.0.113.12",
    "userAgent": "git/2.11.0.windows.1",
    "requestParameters": null,
    "responseElements": null,
    "additionalEventData": {
        "protocol": "HTTP",
        "dataTransferred": false,
        "repositoryName": "MyDemoRepo",
        "repositoryId": "8afe792d-EXAMPLE"
    },
    "requestID": "d148de46-EXAMPLE",
    "eventID": "740f179d-EXAMPLE",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::CodeCommit::Repository"
        }
    ],
    "eventType": "AwsApiCall",
    "apiVersion": "2015-04-13",
    "recipientAccountId": "111122223333"
}
```
The following example shows a CloudTrail log entry that demonstrates the GitPull action where the local repo is not up-to-date and so data is transferred from the AWS CodeCommit repository to the local repo.

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AIDACKCEVSQ6C2EXAMPLE",
    "arn": "arn:aws:iam::444455556666:user/Mary_Major",
    "accountId": "444455556666",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "userName": "Mary_Major"
  },
  "eventTime": "2016-12-14T18:19:15Z",
  "eventSource": "codecommit.amazonaws.com",
  "eventName": "GitPull",
  "awsRegion": "us-east-2",
  "sourceIPAddress": "203.0.113.12",
  "userAgent": "git/2.10.1",
  "requestParameters": null,
  "responseElements": null,
  "additionalEventData": {
    "protocol": "HTTP",
    "capabilities": [
      "multi_ack_detailed",
      "side-band-64k",
      "thin-pack"
    ],
    "dataTransferred": true,
    "repositoryName": "MyDemoRepo",
    "repositoryId": "8afe792d-EXAMPLE",
    "shallow": false
  },
  "requestID": "d148de46-EXAMPLE",
  "eventID": "740f179d-EXAMPLE",
  "readOnly": true,
  "resources": [
    {
      "accountId": "111122223333",
      "type": "AWS::CodeCommit::Repository"
    }
  ],
  "eventType": "AwsApiCall",
  "recipientAccountId": "111122223333"
}
```

Example: A log entry for a successful push to an AWS CodeCommit repository

The following example shows a CloudTrail log entry that demonstrates a successful GitPush action. The GitPush action appears twice in a log entry for a successful push.

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AIDACKCEVSQ6C2EXAMPLE",
    "arn": "arn:aws:iam::444455556666:user/Mary_Major",
    "accountId": "444455556666",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "userName": "Mary_Major"
  },
  "eventTime": "2016-12-14T18:19:15Z",
  "eventSource": "codecommit.amazonaws.com",
  "eventName": "GitPush",
  "awsRegion": "us-east-2",
  "sourceIPAddress": "203.0.113.12",
  "userAgent": "git/2.10.1",
  "requestParameters": null,
  "responseElements": null,
  "additionalEventData": {
    "protocol": "HTTP",
    "capabilities": [
      "multi_ack_detailed",
      "side-band-64k",
      "thin-pack"
    ],
    "dataTransferred": true,
    "repositoryName": "MyDemoRepo",
    "repositoryId": "8afe792d-EXAMPLE",
    "shallow": false
  },
  "requestID": "d148de46-EXAMPLE",
  "eventID": "740f179d-EXAMPLE",
  "readOnly": true,
  "resources": [
    {
      "accountId": "111122223333",
      "type": "AWS::CodeCommit::Repository"
    }
  ],
  "eventType": "AwsApiCall",
  "recipientAccountId": "111122223333"
}
```
"arn": "arn:aws:iam::444455556666:user/Mary_Major",
"accountId": "444455556666",
"accessKeyId": "AKIAIOSFODNN7EXAMPLE",
"userName": "Mary_Major",
"eventTime": "2016-12-14T18:19:15Z",
"eventSource": "codecommit.amazonaws.com",
"eventName": "GitPush",
"awsRegion": "us-east-2",
"sourceIPAddress": "203.0.113.12",
"userAgent": "git/2.10.1",
"requestParameters": null,
"responseElements": null,
"additionalEventData": {
  "protocol": "HTTP",
  "dataTransferred": false,
  "repositoryName": "MyDemoRepo",
  "repositoryId": "8afe792d-EXAMPLE",
},
"requestID": "d148de46-EXAMPLE",
"eventID": "740f179d-EXAMPLE",
"readOnly": true,
"resources": [
  {
    "accountId": "111122223333",
    "type": "AWS::CodeCommit::Repository"
  }
],
"eventType": "AwsApiCall",
"recipientAccountId": "111122223333",

"eventVersion": "1.05",
"userIdentity": {
  "type": "IAMUser",
  "principalId": "AIDACKCEVSQ6C2EXAMPLE",
  "arn": "arn:aws:iam::444455556666:user/Mary_Major",
  "accountId": "444455556666",
  "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
  "userName": "Mary_Major"
},
"eventTime": "2016-12-14T18:19:15Z",
"eventSource": "codecommit.amazonaws.com",
"eventName": "GitPush",
"awsRegion": "us-east-2",
"sourceIPAddress": "203.0.113.12",
"userAgent": "git/2.10.1",
"requestParameters": {
  "references": [
    {
      "commit": "100644EXAMPLE",
      "ref": "refs/heads/master"
    }
  ],

  "responseElements": null,
  "additionalEventData": {
    "protocol": "HTTP",
    "capabilities": [
      "report-status",
      "side-band-64k"
    ],
    "dataTransferred": true,
    "repositoryName": "MyDemoRepo",
    "repositoryId": "8afe792d-EXAMPLE",
  }
}
AWS CodeCommit Command Line Reference

This reference will help you learn how to use AWS CLI.

To install and configure the AWS CLI

1. On your local machine, download and install the AWS CLI. This is a prerequisite for interacting with AWS CodeCommit from the command line. For more information, see Getting Set Up with the AWS Command Line Interface.

   **Note**
   AWS CodeCommit works only with AWS CLI versions 1.7.38 and later. To determine which version of the AWS CLI you have installed, run the `aws --version` command. To upgrade an older version of the AWS CLI to the latest version, see Installing the AWS Command Line Interface.

2. Run this command to verify the AWS CodeCommit commands for the AWS CLI are installed:

   ```
   aws codecommit help
   ```

   This command should return a list of AWS CodeCommit commands.

3. Configure the AWS CLI with the `configure` command, as follows:

   ```
   aws configure
   ```

   When prompted, specify the AWS access key and AWS secret access key of the IAM user you will use with AWS CodeCommit. Also, be sure to specify the region where the repository exists, such as `us-east-2`. When prompted for the default output format, specify `json`. For example:

   ```
   AWS Access Key ID [None]: Type your target AWS access key ID here, and then press Enter
   AWS Secret Access Key [None]: Type your target AWS secret access key here, and then press Enter
   Default region name [None]: Type a supported region for AWS CodeCommit here, and then press Enter
   Default output format [None]: Type json here, and then press Enter
   ```

   To connect to a repository or a resource in another region, you must re-configure the AWS CLI with the default region name for that region. Supported default region names for AWS CodeCommit include:

   - `us-east-2`
   - `us-east-1

   API Version 2015-04-13

   295
For more information about AWS CodeCommit and regions, see Regions and Git Connection Endpoints (p. 278). For more information about IAM, access keys, and secret keys, see How Do I Get Credentials? and Managing Access Keys for IAM Users.

To view a list of all available AWS CodeCommit commands, run the following command:

```
aws codecommit help
```

To view information about a specific AWS CodeCommit command, run the following command, where `command-name` is the name of the command (for example, `create-repository`):

```
aws codecommit command-name help
```

To learn how to use the commands in AWS CLI, go to one or more of the following sections to view descriptions and example usage:

- `batch-get-repositories` (p. 112)
- `create-branch` (p. 196)
- `create-pull-request` (p. 143)
- `create-repository` (p. 77)
- `delete-branch` (p. 206)
- `delete-comment-content` (p. 187)
- `delete-repository` (p. 129)
- `describe-pull-request-events` (p. 147)
- `get-blob` (p. 171)
- `get-branch` (p. 199)
- `get-comment` (p. 186)
- `get-comments-for-compared-commit` (p. 185)
- `get-comments-for-pull-request` (p. 153)
- `get-commit` (p. 169)
- `get-differences` (p. 170)
- `get-merge-conflicts` (p. 147)
- `get-pull-request` (p. 146)
- `get-repository` (p. 112)
- `get-repository-triggers` (p. 105)
Basic Git Commands

You can use Git to work with a local repo and the AWS CodeCommit repository to which you've connected the local repo.

The following are some basic examples of frequently used Git commands.

For more options, see your Git documentation.

Topics

- Configuration Variables (p. 297)
- Remote Repositories (p. 298)
- Commits (p. 299)
- Branches (p. 299)
- Tags (p. 300)

Configuration Variables

<table>
<thead>
<tr>
<th>Description</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lists all configuration variables.</td>
<td><code>git config --list</code></td>
</tr>
<tr>
<td>Lists only local configuration variables.</td>
<td><code>git config --local -l</code></td>
</tr>
<tr>
<td>Lists only system configuration variables.</td>
<td><code>git config --system -l</code></td>
</tr>
<tr>
<td>Lists only global configuration variables.</td>
<td><code>git config --global -l</code></td>
</tr>
<tr>
<td>Sets a configuration variable in the specified configuration file.</td>
<td>`git config [--local</td>
</tr>
<tr>
<td>Edits a configuration file directly. Can also be used to discover the</td>
<td>`git config [--local</td>
</tr>
<tr>
<td>location of a specific configuration file. To exit edit mode, typically</td>
<td></td>
</tr>
<tr>
<td>you type :q (to</td>
<td></td>
</tr>
</tbody>
</table>
Remote Repositories

<table>
<thead>
<tr>
<th>Action</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initializes a local repo in preparation for connecting it to an AWS CodeCommit repository.</td>
<td><code>git init</code></td>
</tr>
<tr>
<td>Can be used to set up a connection between a local repo and a remote repository (such as an AWS CodeCommit repository) using the specified nickname the local repo has for the AWS CodeCommit repository and the specified URL to the AWS CodeCommit repository.</td>
<td><code>git remote add remote-name remote-url</code></td>
</tr>
<tr>
<td>Creates a local repo by making a copy of an AWS CodeCommit repository at the specified URL, in the specified subfolder of the current folder on the local machine. This command also creates a remote tracking branch for each branch in the cloned AWS CodeCommit repository and creates and checks out an initial branch that is forked from the current default branch in the cloned AWS CodeCommit repository.</td>
<td><code>git clone remote-url local-subfolder-name</code></td>
</tr>
<tr>
<td>Shows the nickname the local repo uses for the AWS CodeCommit repository.</td>
<td><code>git remote</code></td>
</tr>
<tr>
<td>Shows the nickname and the URL the local repo uses for fetches and pushes to the AWS CodeCommit repository.</td>
<td><code>git remote -v</code></td>
</tr>
<tr>
<td>Pushes finalized commits from the local repo to the AWS CodeCommit repository, using the specified nickname the local repo has for the AWS CodeCommit repository and the specified branch. Also sets up upstream tracking information for the local repo during the push.</td>
<td><code>git push -u remote-name branch-name</code></td>
</tr>
<tr>
<td>Pushes finalized commits from the local repo to the AWS CodeCommit repository after upstream tracking information is set.</td>
<td><code>git push</code></td>
</tr>
<tr>
<td>Pulls finalized commits to the local repo from the AWS CodeCommit repository, using the specified nickname the local repo has for the AWS CodeCommit repository and the specified branch</td>
<td><code>git pull remote-name branch-name</code></td>
</tr>
<tr>
<td>Pulls finalized commits to the local repo from the AWS CodeCommit repository after upstream tracking information is set.</td>
<td><code>git pull</code></td>
</tr>
<tr>
<td>Disconnects the local repo from the AWS CodeCommit repository, using the specified nickname the local repo has for the AWS CodeCommit repository.</td>
<td><code>git remote rm remote-name</code></td>
</tr>
</tbody>
</table>
### Commits

<table>
<thead>
<tr>
<th>Description</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shows what has or hasn't been added to the pending commit in the local repo.</td>
<td><code>git status</code></td>
</tr>
<tr>
<td>Shows what has or hasn't been added to the pending commit in the local repo in a concise format.</td>
<td><code>git status -sb</code></td>
</tr>
<tr>
<td>(M = modified, A = added, D = deleted, and so on)</td>
<td></td>
</tr>
<tr>
<td>Shows changes between the pending commit and the latest commit in the local repo.</td>
<td><code>git diff HEAD</code></td>
</tr>
<tr>
<td>Adds specific files to the pending commit in the local repo.</td>
<td>`git add [file-name-1 file-name-2 file-name-N</td>
</tr>
<tr>
<td>Adds all new, modified, and deleted files to the pending commit in the local repo.</td>
<td><code>git add</code></td>
</tr>
<tr>
<td>Begins finalizing the pending commit in the local repo, which displays an editor to provide a commit message. After the message is entered, the pending commit is finalized.</td>
<td><code>git commit</code></td>
</tr>
<tr>
<td>Finalizes the pending commit in the local repo, including specifying a commit message at the same time.</td>
<td><code>git commit -m &quot;Some meaningful commit comment&quot;</code></td>
</tr>
<tr>
<td>Lists recent commits in the local repo.</td>
<td><code>git log</code></td>
</tr>
<tr>
<td>Lists recent commits in the local repo in a graph format.</td>
<td><code>git log --graph</code></td>
</tr>
<tr>
<td>Lists recent commits in the local repo in a predefined condensed format.</td>
<td><code>git log --pretty=oneline</code></td>
</tr>
<tr>
<td>Lists recent commits in the local repo in a predefined condensed format, with a graph.</td>
<td><code>git log --graph --pretty=oneline</code></td>
</tr>
<tr>
<td>Lists recent commits in the local repo in a custom format, with a graph.</td>
<td><code>git log --graph --pretty=format:&quot;%H (%h) : %cn : %ar : %s&quot;</code></td>
</tr>
<tr>
<td>(For more options, see <a href="#">Git Basics - Viewing the Commit History</a>)</td>
<td></td>
</tr>
</tbody>
</table>

### Branches

<table>
<thead>
<tr>
<th>Description</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lists all branches in the local repo with an asterisk (*) displayed next to your current branch.</td>
<td><code>git branch</code></td>
</tr>
<tr>
<td>Pulls information about all existing branches in the AWS CodeCommit repository to the local repo.</td>
<td><code>git fetch</code></td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td><code>git branch -a</code></td>
<td>Lists all branches in the local repo and remote tracking branches in the local repo.</td>
</tr>
<tr>
<td><code>git branch -r</code></td>
<td>Lists only remote tracking branches in the local repo.</td>
</tr>
<tr>
<td><code>git branch new-branch-name</code></td>
<td>Creates a new branch in the local repo using the specified branch name.</td>
</tr>
<tr>
<td><code>git checkout other-branch-name</code></td>
<td>Switches to another branch in the local repo using the specified branch name.</td>
</tr>
<tr>
<td><code>git checkout -b new-branch-name</code></td>
<td>Creates a new branch in the local repo using the specified branch name, and then switches to it.</td>
</tr>
<tr>
<td><code>git push -u remote-name new-branch-name</code></td>
<td>Pushes a new branch from the local repo to the AWS CodeCommit repository using the specified nickname the local repo has for the AWS CodeCommit repository and the specified branch name. Also sets up upstream tracking information for the branch in the local repo during the push.</td>
</tr>
<tr>
<td><code>git branch --track new-branch-name remote-name/remote-branch-name</code></td>
<td>Creates a new branch in the local repo using the specified branch name. Then connects the new branch in the local repo to an existing branch in the AWS CodeCommit repository, using the specified nickname the local repo has for the AWS CodeCommit repository and the specified branch name.</td>
</tr>
<tr>
<td><code>git merge from-other-branch-name</code></td>
<td>Merges changes from another branch in the local repo to the current branch in the local repo.</td>
</tr>
<tr>
<td><code>git branch -d branch-name</code></td>
<td>Deletes a branch in the local repo unless it contains work that has not been merged.</td>
</tr>
<tr>
<td><code>git push remote-name :branch-name</code></td>
<td>Deletes a branch in the AWS CodeCommit repository using the specified nickname the local repo has for the AWS CodeCommit repository and the specified branch name. (Note the use of the colon (:).)</td>
</tr>
</tbody>
</table>

### Tags

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>git tag</code></td>
<td>Lists all tags in the local repo.</td>
</tr>
<tr>
<td><code>git fetch --tags</code></td>
<td>Pulls all tags from the AWS CodeCommit repository to the local repo.</td>
</tr>
<tr>
<td><code>git show tag-name</code></td>
<td>Shows information about a specific tag in the local repo.</td>
</tr>
<tr>
<td><code>git tag tag-name commit-id-to-point-tag-at</code></td>
<td>Creates a &quot;lightweight&quot; tag in the local repo.</td>
</tr>
<tr>
<td><code>git push remote-name tag-name</code></td>
<td>Pushes a specific tag from the local repo to the AWS CodeCommit repository using the specified nickname the local repo has for the AWS CodeCommit repository and the specified branch name.</td>
</tr>
<tr>
<td>Action</td>
<td>Command</td>
</tr>
<tr>
<td>-----------------------------------------------------------------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>Specifies a tag to the local repo</td>
<td><code>git tag -d tag-name</code></td>
</tr>
<tr>
<td>Deletes a tag in the AWS CodeCommit repository using the specified</td>
<td><code>git push remote-name :tag-name</code></td>
</tr>
<tr>
<td>nickname the local repo has for the AWS CodeCommit repository and the</td>
<td></td>
</tr>
<tr>
<td>specified tag name.</td>
<td></td>
</tr>
<tr>
<td>Pushes all tags from the local repo to the AWS CodeCommit repository</td>
<td><code>git push remote-name --tags</code></td>
</tr>
<tr>
<td>using the specified nickname the local repo has for the AWS CodeCommit repository.</td>
<td></td>
</tr>
</tbody>
</table>
# AWS CodeCommit User Guide

Document History

The following table describes the important changes to the documentation since the last release of the *AWS CodeCommit User Guide*.

- **API version**: 2015-04-13
- **Latest documentation update**: February 21, 2018

<table>
<thead>
<tr>
<th>Change</th>
<th>Description</th>
<th>Date Changed</th>
</tr>
</thead>
<tbody>
<tr>
<td>New section</td>
<td>The Working with Files in AWS CodeCommit Repositories (p. 130) section has been added. The AWS CodeCommit Permissions Reference (p. 267) and Getting Started with AWS CodeCommit Tutorial (p. 44) topics have been updated.</td>
<td>February 21, 2018</td>
</tr>
<tr>
<td>New topic</td>
<td>The Configure Cross-Account Access to an AWS CodeCommit Repository (p. 120) topic has been added.</td>
<td>February 21, 2018</td>
</tr>
<tr>
<td>New topic</td>
<td>The Integrate AWS Cloud9 with AWS CodeCommit (p. 12) topic has been added. The Product and Service Integrations (p. 68) topic has been updated with information about AWS Cloud9.</td>
<td>December 1, 2017</td>
</tr>
<tr>
<td>New section</td>
<td>The Working with Pull Requests in AWS CodeCommit Repositories (p. 138) section has been added. The Authentication and Access Control for AWS CodeCommit (p. 250) section has been updated with information about permissions for pull requests and commenting. It also includes updated managed policy statements.</td>
<td>November 20, 2017</td>
</tr>
<tr>
<td>Updated topics</td>
<td>The Product and Service Integrations (p. 68) topic has been updated to include links for customers who want to update their existing pipelines to use Amazon CloudWatch Events to start pipelines in response to changes in an AWS CodeCommit repository.</td>
<td>October 11, 2017</td>
</tr>
<tr>
<td>New topics</td>
<td>The Authentication and Access Control for AWS CodeCommit (p. 250) section has been added. It replaces the Access Permissions Reference topic.</td>
<td>September 11, 2017</td>
</tr>
<tr>
<td>Updated topics</td>
<td>The Manage Triggers for a Repository (p. 91) section has been updated to reflect changes in trigger configuration. Topics and images have been updated throughout the guide to reflect changes in the navigation bar.</td>
<td>August 29, 2017</td>
</tr>
<tr>
<td>New topic</td>
<td>The Working with User Preferences (p. 208) topic has been added. The View Tag Details (p. 190) topic has been updated. The Product and Service Integrations (p. 68) topics have been</td>
<td>August 3, 2017</td>
</tr>
<tr>
<td>Change</td>
<td>Description</td>
<td>Date Changed</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>Updated topic</td>
<td>AWS CodeCommit is now available in two additional regions: Asia Pacific (Mumbai), and Canada (Central). The Regions and Git Connection Endpoints (p. 278) topic has been updated.</td>
<td>June 29, 2017</td>
</tr>
<tr>
<td>Updated topic</td>
<td>AWS CodeCommit is now available in four additional regions: Asia Pacific (Tokyo), Asia Pacific (Singapore), Asia Pacific (Sydney), and EU (Frankfurt). The Regions and Git Connection Endpoints (p. 278) topic has been updated to provide information about Git connection endpoints and supported regions for AWS CodeCommit.</td>
<td>June 6, 2017</td>
</tr>
<tr>
<td>Updated topic</td>
<td>AWS CodeCommit is now available in four additional regions: Asia Pacific (Seoul), South America (São Paulo), US West (N. California), and EU (London). The Regions and Git Connection Endpoints (p. 278) topic has been updated.</td>
<td>May 25, 2017</td>
</tr>
<tr>
<td>New topic</td>
<td>The Compare Branches (p. 200) topic has been added. The contents of the Working with Branches (p. 194) section have been updated with information about using the AWS CodeCommit console to work with branches in a repository.</td>
<td>May 18, 2017</td>
</tr>
<tr>
<td>New topic</td>
<td>The Compare Commits (p. 173) topic has been added with information about comparing commits. The structure of the user guide has been updated for working with repositories (p. 75), commits, (p. 163), and branches (p. 194).</td>
<td>March 28, 2017</td>
</tr>
<tr>
<td>Updated topic</td>
<td>The View Commit Details (p. 166) topic has been updated with information about viewing the difference between a commit and its parent in the console, and using the get-differences command to view differences between commits using the AWS CLI.</td>
<td>January 24, 2017</td>
</tr>
<tr>
<td>New topic</td>
<td>The Logging AWS CodeCommit API Calls with AWS CloudTrail (p. 289) topic has been added with information about logging connections to AWS CodeCommit using AWS CloudFormation.</td>
<td>January 11, 2017</td>
</tr>
<tr>
<td>New topic</td>
<td>The For HTTPS Users Using Git Credentials (p. 5) topic has been added with information about setting up connections to AWS CodeCommit using Git credentials over HTTPS.</td>
<td>December 22, 2016</td>
</tr>
<tr>
<td>Updated topic</td>
<td>The Product and Service Integrations (p. 68) topic has been updated to include information about integration with AWS CodeBuild.</td>
<td>December 5, 2016</td>
</tr>
<tr>
<td>Change</td>
<td>Description</td>
<td>Date Changed</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>Updated topic</td>
<td>AWS CodeCommit is now available in another region, EU (Ireland). The <em>Regions and Git Connection Endpoints (p. 278)</em> topic has been updated to provide information about Git connection endpoints and supported regions for AWS CodeCommit.</td>
<td>November 16, 2016</td>
</tr>
<tr>
<td>Updated topic</td>
<td>AWS CodeCommit is now available in another region, US West (Oregon). The <em>Regions and Git Connection Endpoints (p. 278)</em> topic has been updated to provide information about Git connection endpoints and supported regions for AWS CodeCommit.</td>
<td>November 14, 2016</td>
</tr>
<tr>
<td>New topic</td>
<td>The <em>Create a Trigger for a Lambda Function (p. 97)</em> topic has been updated to reflect the ability to create AWS CodeCommit triggers as part of creating the Lambda function. This simplified process streamlines trigger creation and automatically configures the trigger with the permissions required for AWS CodeCommit to invoke the Lambda function. The <em>Create a Trigger for an Existing Lambda Function (p. 100)</em> topic has been added to include information about creating triggers for existing Lambda functions in the AWS CodeCommit console.</td>
<td>October 19, 2016</td>
</tr>
<tr>
<td>New topic</td>
<td>AWS CodeCommit is now available in another region, US East (Ohio). The <em>Regions and Git Connection Endpoints (p. 278)</em> topic has been added to provide information about Git connection endpoints and supported regions for AWS CodeCommit.</td>
<td>October 17, 2016</td>
</tr>
<tr>
<td>Topic update</td>
<td>The <em>Product and Service Integrations (p. 68)</em> topic has been updated to include information about integration with AWS Elastic Beanstalk.</td>
<td>October 13, 2016</td>
</tr>
<tr>
<td>Topic update</td>
<td>The <em>Product and Service Integrations (p. 68)</em> topic has been updated to include information about integration with AWS CloudFormation.</td>
<td>October 6, 2016</td>
</tr>
<tr>
<td>Topic update</td>
<td>The <em>For SSH Connections on Windows (p. 30)</em> topic has been revised to provide guidance for using a Bash emulator for SSH connections on Windows instead of the PuTTY suite of tools.</td>
<td>September 29, 2016</td>
</tr>
<tr>
<td>Topic update</td>
<td>The <em>View Commit Details (p. 166)</em> and <em>AWS CodeCommit Tutorial (p. 44)</em> topics have been updated to include information about the Commit Visualizer in the AWS CodeCommit console. The <em>Limits (p. 282)</em> topic has been updated with the increase to the number of references allowed in a single push.</td>
<td>September 14, 2016</td>
</tr>
<tr>
<td>Topic update</td>
<td>The <em>View Commit Details (p. 166)</em> and <em>AWS CodeCommit Tutorial (p. 44)</em> topics have been updated to include information about viewing the history of commits in the AWS CodeCommit console.</td>
<td>July 28, 2016</td>
</tr>
<tr>
<td>Change</td>
<td>Description</td>
<td>Date Changed</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>New topics</td>
<td>The <a href="#">Migrate a Git Repository to AWS CodeCommit</a> (p. 211) and <a href="#">Migrate Local or Unversioned Content to AWS CodeCommit</a> (p. 219) topics have been added.</td>
<td>June 29, 2016</td>
</tr>
<tr>
<td>Topic update</td>
<td>Minor updates have been made to the <a href="#">Troubleshooting</a> (p. 235) and <a href="#">For HTTPS Connections on Windows with the AWS CLI Credential Helper</a> (p. 39) topics.</td>
<td>June 22, 2016</td>
</tr>
<tr>
<td>Topic update</td>
<td>The <a href="#">Product and Service Integrations</a> (p. 68) and <a href="#">Access Permissions Reference</a> topics have been updated to include information about integration with AWS CodePipeline.</td>
<td>April 18, 2016</td>
</tr>
<tr>
<td>New topics</td>
<td>The <a href="#">Manage Triggers for a Repository</a> (p. 91) section has been added. New topics include examples, including policy and code samples, of how to create, edit, and delete triggers.</td>
<td>March 7, 2016</td>
</tr>
<tr>
<td>New topic</td>
<td>The <a href="#">Product and Service Integrations</a> (p. 68) topic has been added. Minor updates have been made to <a href="#">Troubleshooting</a> (p. 235).</td>
<td>March 7, 2016</td>
</tr>
<tr>
<td>Topic update</td>
<td>In addition to the MD5 server fingerprint, the SHA256 server fingerprint for AWS CodeCommit has been added to <a href="#">For SSH Connections on Linux, macOS, or Unix</a> (p. 26) and <a href="#">For SSH Connections on Windows</a> (p. 30).</td>
<td>December 9, 2015</td>
</tr>
<tr>
<td>New topic</td>
<td>The <a href="#">Browse Files in a Repository</a> (p. 131) topic has been added. New issues have been added to <a href="#">Troubleshooting</a> (p. 235). Minor improvements and fixes have been made throughout the user guide.</td>
<td>October 5, 2015</td>
</tr>
<tr>
<td>New topic</td>
<td>The <a href="#">For SSH Users Not Using the AWS CLI</a> (p. 24) topic has been added. The topics in the <a href="#">Setting Up</a> (p. 4) section have been streamlined. Guidance to help users determine which steps to follow for their operating systems and preferred protocols has been provided.</td>
<td>August 5, 2015</td>
</tr>
<tr>
<td>Topic update</td>
<td>Clarification and examples have been added to the SSH key ID steps in <a href="#">SSH and Linux, macOS, or Unix: Set Up the Public and Private Keys for Git and AWS CodeCommit</a> (p. 27) and <a href="#">SSH and Windows: Set Up the Public and Private Keys for Git and AWS CodeCommit</a> (p. 32).</td>
<td>July 24, 2015</td>
</tr>
<tr>
<td>Topic update</td>
<td>Steps in <a href="#">SSH and Windows: Set Up the Public and Private Keys for Git and AWS CodeCommit</a> (p. 32) have been updated to address an issue with IAM and saving the public key file.</td>
<td>July 22, 2015</td>
</tr>
<tr>
<td>Topic update</td>
<td><a href="#">Troubleshooting</a> (p. 235) has been updated with navigation aids. More troubleshooting information for credential keychain issues has been added.</td>
<td>July 20, 2015</td>
</tr>
<tr>
<td>Topic update</td>
<td>More information about AWS Key Management Service permissions has been added to the <a href="#">AWS KMS and Encryption</a> (p. 288) and the <a href="#">Access Permissions Reference</a> topics.</td>
<td>July 17, 2015</td>
</tr>
<tr>
<td>Change</td>
<td>Description</td>
<td>Date Changed</td>
</tr>
<tr>
<td>-----------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>Topic update</td>
<td>Another section has been added to Troubleshooting (p. 235) with information about troubleshooting issues with AWS Key Management Service.</td>
<td>July 10, 2015</td>
</tr>
<tr>
<td>Initial release</td>
<td>This is the initial release of the AWS CodeCommit User Guide.</td>
<td>July 9, 2015</td>
</tr>
</tbody>
</table>
AWS Glossary

For the latest AWS terminology, see the AWS Glossary in the AWS General Reference.