1
Exploring AWS CodeCommit

In the era of Agile and DevOps, where everything (that is, software and infrastructure) is as-code, it is necessary to have a reliable and secure source code repository. The repository should fulfill the needs of a team of any size, from small to large. The team members should be able to work from anywhere as part of a team, but they should also be able to work independently. Working independently helps developers to contribute in coding on various versions, bug fixing, and feature releases of the software.

Operating or managing a Git server can be a tedious job, in terms of managing authentication and integrating with other DevOps tools to be agile (that is, **Continuous Integration** (**CI**)/**Continuous Delivery** (**CD**)). On top of that, especially when there are a lot of branches and a large number of developers are working at the same time, it may become a challenge to maintain performance—a large number of push, pull, commit, or other basic repository functions would be executed in parallel, which may require huge compute, RAM, disk, and network I/O.

If a team of developers is not working in parallel and just needs to maintain various versions of the object, it is best to use the AWS S3 bucket with enabled versioning.

AWS CodeCommit is a highly available, durable, and fully managed source code service. It is a Git-based source code repository. At the time of writing, AWS CodeCommit provides five free active users in a free tier. It can be used for various purposes, such as documents, source codes, and binary files. After the free tier, AWS charges $1 per active user. CodeCommit is also compliant with **Health Insurance Portability and Accountability Act** (**HIPAA**) and **Payment Card Industry Data Security Standard** (**PCI DSS**).

More details about pricing can be found at `https://aws.amazon.com/codecommit/pricing/`.

The aim of this chapter is to introduce you to AWS CodeCommit. The chapter covers the following topics with relevant reference URLs for further reading, where required:

- How CodeCommit works
- Configuring HTTPS users using Git credentials
- Creating a repository
- CodeCommit events
- Working with branches
- Working with pull requests

How CodeCommit works

AWS CodeCommit can be accessed using a web console and also using the Git command line or AWS CLI. Our interaction with AWS CodeCommit source code repositories is shown in the following diagram:

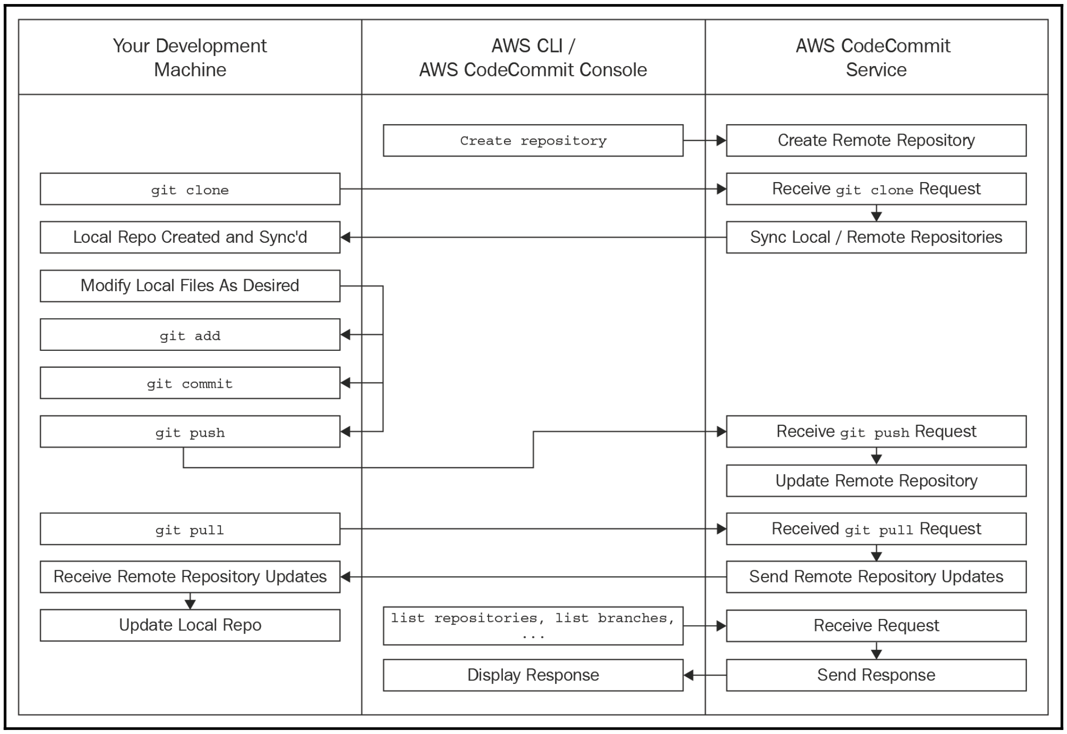

Your Development Machine
AWS CLI / AWS CodeCommit Console
AWS CodeCommit Service
Create repository
Create Remote Repository
git clone
Receive git clone Request
Local Repo Created and Sync'd
Sync Local / Remote Repositories
Modify Local Files As Desired
git add
git commit
git push
Receive git push Request
Update Remote Repository
git pull
Received git pull Request
Receive Remote Repository Updates
Send Remote Repository Updates
Update Local Repo
list repositories, list branches, ...
Receive Request
Display Response
Send Response

Figure 20.1: How CodeCommit works

The following section describes how to work with CodeCommit:

1. When we are starting fresh work on a new software development project (that is, starting a new repository), we can create it using the AWS CLI or web console.
2. Once a centralized remote source code repository is created, developers can start contributing code and files into it. Before contributing code and files to the remote repository, it is a best practice to create a local copy of the remote source code repository. As a result, the developer is no longer required to stay connected over the internet with the remote repository. It can be easily done with the `git clone` command on the software developer's local machine. Executing this command will send a request to the remote source code repository to clone (that is, copy) it to the local machine. It will also validate the request by username and password. For example, to clone a remote AWS CodeCommit repository named `LearnCodeCommit`, execute the following command:

```
$ git clone
https://git-codecommit.us-east-1.amazonaws.com/v1/repo
s/LearnCodeCommit
```

`git clone` a remote repository to a working directory on a local machine. The working directory shouldn't be `/tmp` or `C:\temp` on Linux or Windows OS, respectively.

Executing the preceding command will create a `LearnCodeCommit` directory on a local machine and place in it all the repository source code or files, along with the metadata and configuration. Once a remote source code repository is cloned to the developer's local machine, it is ready to start using Git commands from the local repository.

Before executing Git commands, make sure the Git CLI is installed and configured on the developer's local machine.
In the *Creating a repository* section, we explained how to obtain an HTTPS/SSH link to clone the remote repository to the local machine. It will prompt you for a username and password. AWS CodeCommit doesn't allow you to use your defined username and password. Your username and password can be obtained for a specific AWS IAM user from **HTTPS Git credentials for AWS CodeCommit** inside **Security credentials**. More details about the credentials are given in the subsequent section.

3. Once the developer has cloned a local copy for the remote repository, they can start contributing their work (usually in the form of programming: source code) or any binary files into the local repository. Storing a source code or a file on a remote source code repository is a three-step process:

    1. The developer creates or modifies several source codes using mostly the IDE and sometimes the CLI. When creating a new or modifying an existing file using CLI, they can also be added to the local repository using the `git add` command. This command tells the local repository that this new set of files has been added/updated and is ready to be stored in a local repository. This process is also called **staging**. For example, you are inside a directory called **local repository directory**, and you have created a new file called `login.py` and would like to add to the local code repository, we can do that as follows:

        ```
        $ touch login.py
        ```

        A blank file has been created using the `touch` command. Let's demonstrate the `git status` command:

        ```
        $ git status
        # On branch master
        #
        # Initial commit
        #
        # Untracked files:
        # (use "git add <file>..." to include in what
        will be committed)
        #
        # login.py
        nothing added to commit but untracked files
        present (use "git add" to track)
        ```

        The `git status` command will list all the files, whether they are newly created or modified existing ones. Based on the list, the appropriate decision can be made to add files to the staging for committing to the local repository. Therefore, we use `git add` to add files to the local repository:

        ```
        $ git add login.py
        ```

Now the `git status` command lists the newly created file, which is staged and ready to commit in the local repository:

```
$ git status
# On branch master
#
# Initial commit
#
# Changes to be committed:
# (use "git rm --cached <file>..." to unstage)
#
# new file: login.py
#
```

The preceding `git add` command will not add the newly created file to the local repository, but it will prepare a file or set of files to commit to the local repository when a subsequent `git commit` command is executed. There may be a situation where the developer has changed or created many source code files for a given task or project on their local machine. Out of these files, they just want to store (that is, commit) a few selected and relevant source code or files to the repository. This way, they have the liberty to select which files they would like to prepare for committing by staging them using the `git add` command.

When developing software, each of the developers may be using their preferred IDE. It is also possible to integrate an IDE (such as, AWS Cloud9, Microsoft Visual Studio, or Eclipse) with AWS CodeCommit. To learn how to set up connections from the preferred IDE, please refer to `https://docs.aws.amazon.com/codecommit/latest/userguide/setting-up-ide.html`.

These IDEs can directly add new files to the local repository and commit to the configured remote repository.

2. Once recently added or modified files are staged, they are ready to be stored (that is, committed) in a local repository, which can be achieved using the `git commit` command. This command will store staged files in a local repository, as in the following example:

   ```
   $ git commit -m "login.py module added"
   [master (root-commit) 32b0e1f] login.py module
   added
   1 file changed, 0 insertions(+), 0 deletions(-)
   create mode 100644 login.py
   ```

   The `-m` parameter will store a message along with unique ID. When we run the `git log` command to check commit history, we can use this message as a reference to understand what changes have been carried out in that particular commit. Now running `git status` will show `nothing to commit, working directory clean`:

   ```
   $ git status
   # On branch master
   nothing to commit, working directory clean
   ```

   The Git repository stores the snapshot of the files (that is, *original files* + *modification*).

3. Usually, once considerable progress for a software development task is achieved, and related files have been staged and committed to the local source code repository, it is time to push these changes to the remote source code repository, which can be done using the `git push` command. This command will ask the remote source code repository to accept the changes made by one of the developers in a team on a local machine. On successful execution of this remote source code repository, it is updated with the latest changes carried out on one of the developer's local machines. In the same way, each of the developers in a team can push their change to the remote source code repository:

   ```
   $ git push origin master
   Counting objects: 3, done.
   Delta compression using up to 4 threads.
   Compressing objects: 100% (2/2), done.
   Writing objects: 100% (2/2), 235 bytes | 0
   bytes/s, done.
   ```

```
Total 2 (delta 0), reused 0 (delta 0)
To
https://git-codecommit.us-east-1.amazonaws.co
m/v1/repos/LearnCodeCommit
 32b0e1f..394963d master -> master
```

When your local code repository is out of sync (behind) with the remote source code repository (usually called the **upstream repository**), you will get an error message saying, **Note about fast-forwards**.

The `git push` command takes two arguments: origin and branch name. Origin indicates the remote repository, which is from where the local repository was cloned. In this case, the branch name is `master`. The branch name depends on the bug fix, the feature, or the part of the software development project you are working on. By default, the AWS CodeCommit repository is blank and without any branches. Once a file has been stored automatically, it creates a master branch, and that is the default branch. Further down the chapter, the *Working with branches* section describes source code branching strategies at a high level.

In general, Git commands require authentication. Whenever you use Git commands, it prompts you for your username and password. More details on credentials are given in the subsequent section named *Configuring HTTPS users using Git credentials*.

4. In a development team, there may be multiple developers working on the same repository with multiple branches. When a developer starts work on a branch, they may need to sync up their local repository with the remote repository. Also, there may be situations wherein a developer would want to restore the changes done on the local repository. When the local repository deviates from the remote repository and the developer needs to restore the local repository, they can pull the remote source code repository. Developers can update their local source code repository using the `git pull` command.

Figure 20.1 and the following points explain various stages involved in a developer's day-to-day activities during a software development lifecycle when working with the source code repository. It is possible to carry out these steps using an AWS CodeCommit web console. While CodeCommit web console may be easy and quick for working with a couple of files, it is not easy to handle more files using web console. For working with a large number of files and doing constant development, it is recommended that you use CLI or IDE integration with CodeCommit.

To work with remote the AWS CodeCommit source code repository using the CLI, you need to configure a local Git client to communicate with it. This configuration requires authentication to be carried out. AWS IAM provides three types of credentials:

- **IAM-generated username and password**: IAM-generated username and password can be used for Git to communicate with the remote AWS CodeCommit repository over HTTPS. The IAM-generated username and password method is the most recommended method.
- **SSH key**: A public-private key pair that is generated manually and can be associated with an IAM user for communicating with the remote AWS CodeCommit source code repository over SSH. More details on this method can be found at `https://docs.aws.amazon.com/codecommit/latest/userguide/setting-up-without-cli.html`.
- **AWS access and secret keys**: These keys can be used with the credentials that are incorporated with the AWS CLI for communicating with the remote AWS CodeCommit repository over HTTPS.

Configuring HTTPS users using Git credentials

The easiest way to configure the AWS CodeCommit source code repository with the Git CLI on a local machine is to configure Git credentials for AWS CodeCommit in the IAM web console, and then use those credentials for HTTPS connections. These credentials can also be used with any third-party IDE. The HTTPS protocol requires a username and password for every connection. HTTPS connections require either Git credentials or an AWS access key.

The HTTPS connection for the Git CLI to use AWS CodeCommit can be made through the following steps.

Step 1 – creating the IAM user

We start off with creating an IAM user by executing the following steps:

1. Log into the AWS account and open the **IAM** web console.
2. Create a new user or use an existing IAM user. Make sure IAM users have access and a secret key.
3. Once the user is created or an existing user is selected, go to the **Permissions** tab and choose **Add permissions**.

TIP

In order for an IAM user to work with AWS CodeCommit, the user needs access to the AWS **Key Management Service** (**KMS**). If you're using an existing user, please make sure any policy attached doesn't have exclusive deny rules for the AWS KMS service.

4. Choose **Attach existing policies directly** from **Grant permissions**.
5. Select **AWSCodeCommitFullAccess** from the list of AWS-managed policies. Attaching this policy to the IAM user will provide full privileges for AWS CodeCommit. It is assumed that we are creating a source code repository administrator.
6. Click **Next: Review**, and then click on **Add permission**.

To make managing a large number of developers' privileges easy, it is a best practice to create a relevant IAM group (that is, developers), and make each IAM user part of the group. More details on sharing a repository with the developers team can be found at `https://docs.aws.amazon.com/codecommit/latest/userguide/how-to-share-repository.html`.

Step 2 – installing Git

Now make sure Git is installed on the developer's machine. AWS CodeCommit supports Git version 1.7.9 and later. Based on the OS (Microsoft Windows, Linux, or macOS), install Git on developer's local machine.

You can refer to following URLs for downloading and installing Git for your operating system:

- **Linux:** `https://git-scm.com/download/linux`
- **Windows:** `https://git-scm.com/download/win`
- **Mac OS:** `https://git-scm.com/download/mac`

TIP

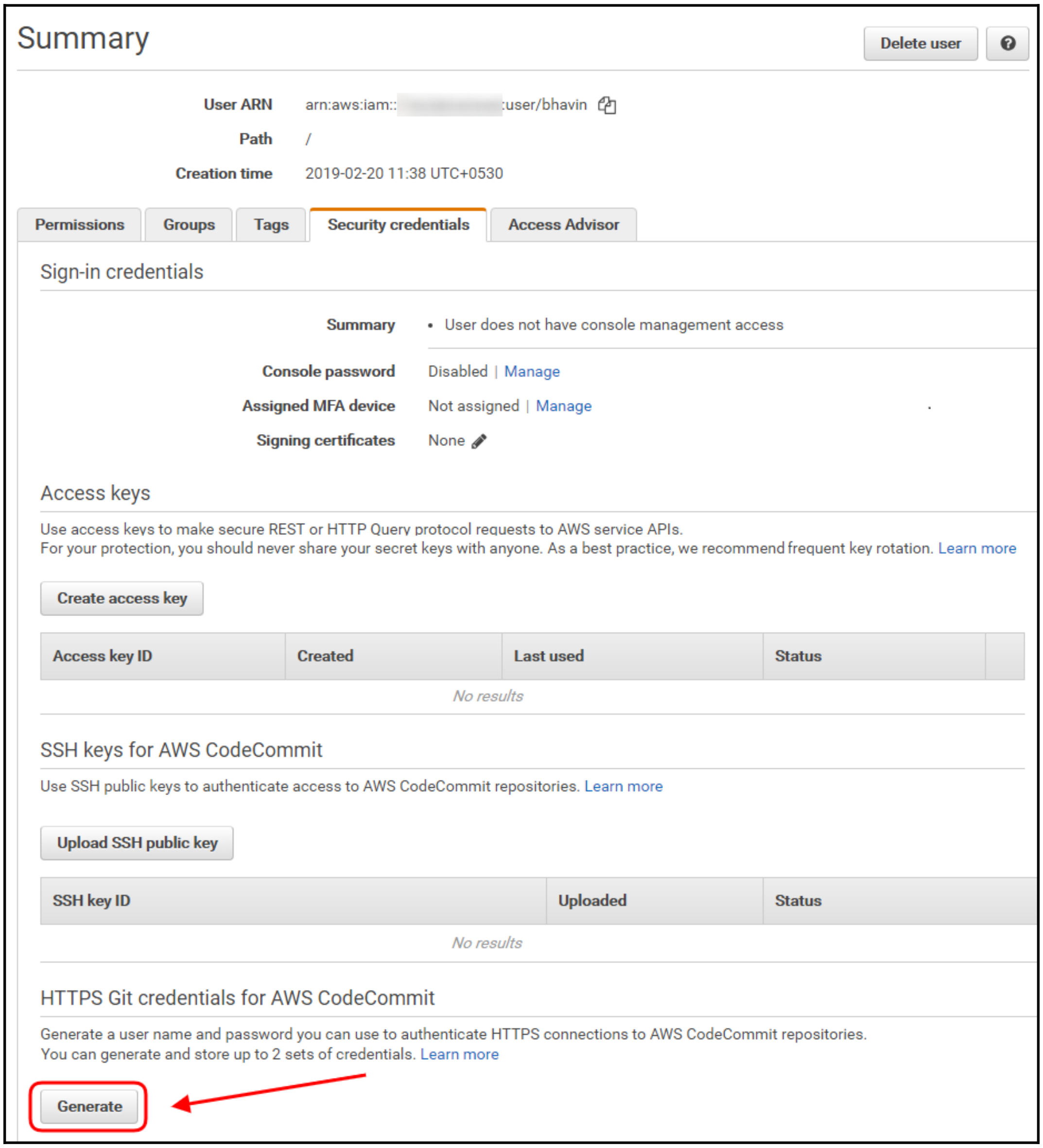
For a better user experience with the Git CLI and the remote source code repository, make sure basic Git configuration such as username and email is done on the developer's local machine.

Step 3 – creating Git credentials for HTTPS connections to AWS CodeCommit

Once the appropriate IAM user with sufficient privileges is created, you can install Git on the developer's machine and start using it. The steps to create the credentials are as follows:

1. Log into the AWS account and open the **IAM** web console.
2. Select the IAM user that was created in the s*tep 1 – creating the IAM user* section.

3. Choose the **Security credentials** tab, and select **Generate** under **HTTPS Git credentials for AWS CodeCommit**:



Summary

Delete user

User ARN	arn:aws:iam::[redacted]:user/bhavin
Path	/
Creation time	2019-02-20 11:38 UTC+0530

Permissions | Groups | Tags | **Security credentials** | Access Advisor

Sign-in credentials

Summary	• User does not have console management access
Console password	Disabled | Manage
Assigned MFA device	Not assigned | Manage
Signing certificates	None

Access keys

Use access keys to make secure REST or HTTP Query protocol requests to AWS service APIs.
For your protection, you should never share your secret keys with anyone. As a best practice, we recommend frequent key rotation. Learn more

Create access key

Access key ID	Created	Last used	Status
	No results		

SSH keys for AWS CodeCommit

Use SSH public keys to authenticate access to AWS CodeCommit repositories. Learn more

Upload SSH public key

SSH key ID	Uploaded	Status
	No results	

HTTPS Git credentials for AWS CodeCommit

Generate a user name and password you can use to authenticate HTTPS connections to AWS CodeCommit repositories.
You can generate and store up to 2 sets of credentials. Learn more

Generate

Figure 20.2: HTTP Git credentials for AWS CodeCommit

4. It is not possible to use your own username and password as Git credentials. Hence, download your credentials and keep them in a safe place.

Git credentials can be downloaded only during their creation. You cannot download them later on. If credentials are lost, you need to reset the password. To avoid this situation, make sure the downloaded credentials are stored in a safe place.

These recently created usernames and passwords are required for each HTTPS connection with the AWS CodeCommit repository.

More information on configuring *HTPPS connections to AWS CodeCommit repositories on Linux, macOS, or Unix with the AWS CLI Credential Helper* is available at `https://docs.aws.amazon.com/codecommit/latest/userguide/setting-up-https-unixes.html`. For Windows, it is available at `https://docs.aws.amazon.com/codecommit/latest/userguide/setting-up-https-windows.html`.

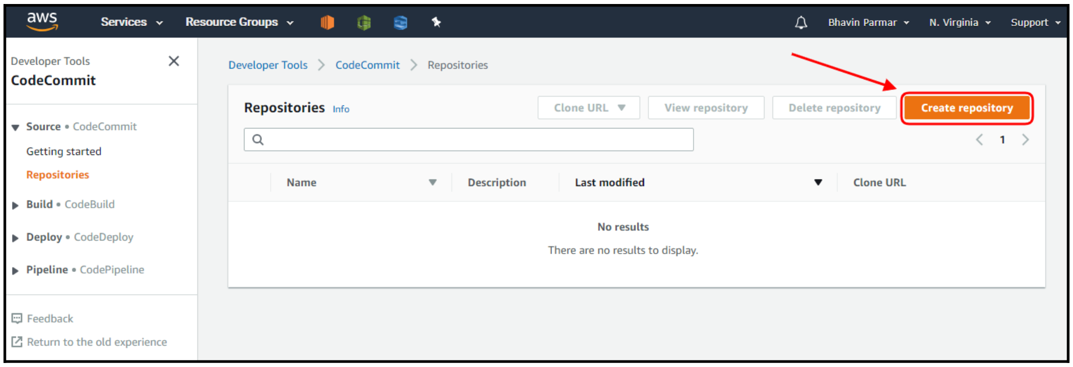
Creating a repository

Once the IAM user is configured with SSH keys or HTTPS Git credentials for AWS CodeCommit, the user gets the required privileges to interact with the remote source code repository. It is essential to create an IAM user for accessing the repository. The repository is a secured and reliable place to store code and files for the project. It stores the history of each commit, and can also sends notifications to the repository users to receive an email when any new commit has occurred.

To create a AWS CodeCommit repository using a web console, execute the following steps:

1. Log into the AWS account as a user that has sufficient privileges to work with AWS CodeCommit.
2. Select the appropriate AWS region from the top-right side of the AWS dashboard.

3. Go to the AWS CodeCommit web console, then click on **Create repository**:

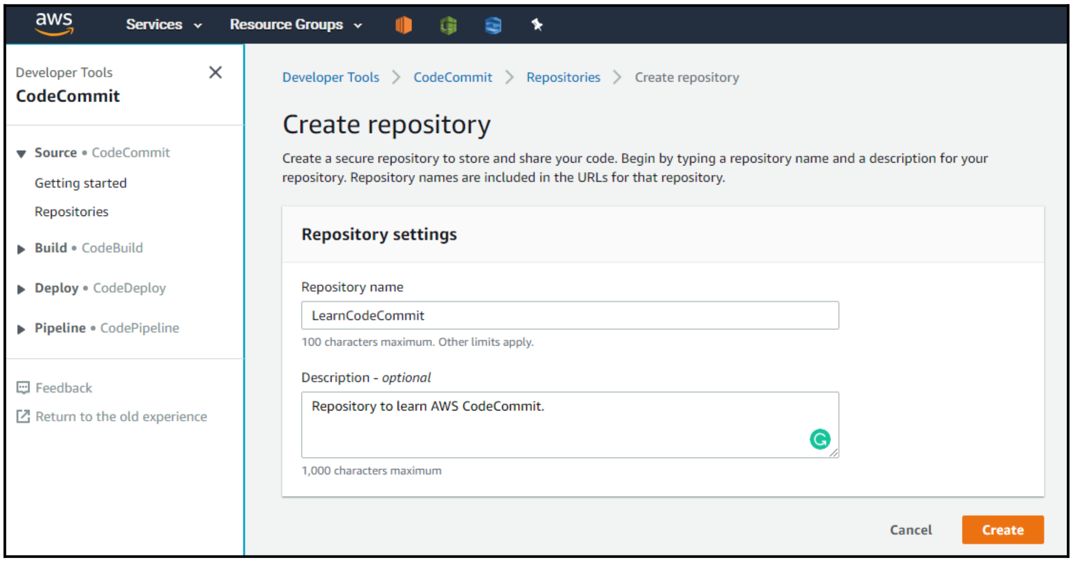


No results

There are no results to display.

Figure 20.3

4. Provide an appropriate **Repository name** and **Description**:



Create repository

Create a secure repository to store and share your code. Begin by typing a repository name and a description for your repository. Repository names are included in the URLs for that repository.

Repository settings

Repository name: LearnCodeCommit

100 characters maximum. Other limits apply.

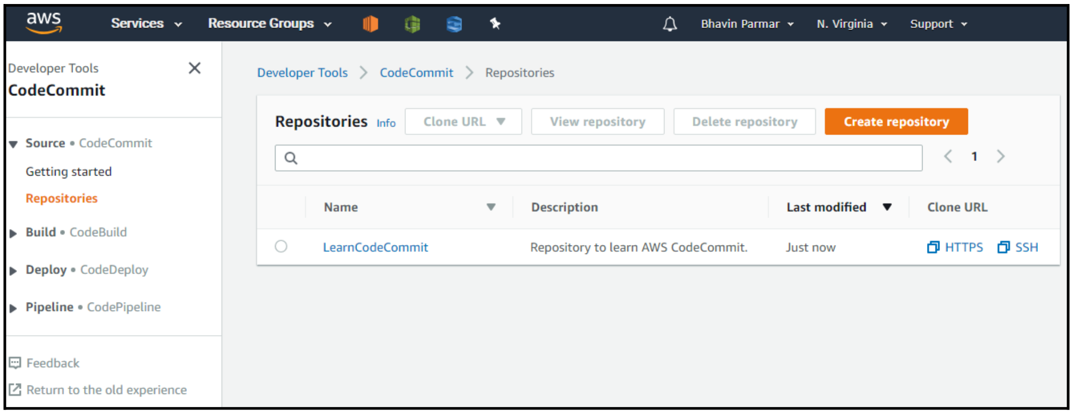
Description - *optional*: Repository to learn AWS CodeCommit.

1,000 characters maximum

Figure 20.4

The AWS CodeCommit repository name must be unique in the region for your AWS account.

5. Click on **Create** to create the remote source code repository. Once it is created, it is available:



	Name	Description	Last modified	Clone URL
○	LearnCodeCommit	Repository to learn AWS CodeCommit.	Just now	HTTPS SSH

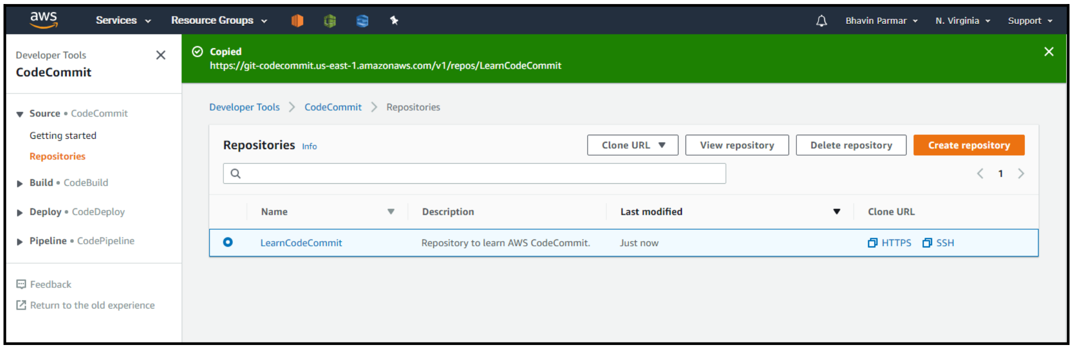
Figure 20.5

The AWS CodeCommit repository can also be created using the AWS CLI for CodeCommit by executing the following command:

```
aws codecommit create-repository --repository-name LearnCodeCommit --
repository-description " Repository to learn AWS CodeCommit"
```

Before executing the preceding command, make sure the `aws configure` command is executed with access, the secret key, and the region.

Now that the repository is created, it is time to connect it with the remote repository. A repository can be remotely connected over HTTPS or SSH protocol. It requires a URL to do so. The URL can be obtained by selecting the desired repository to connect to and clicking on **HTTPS** or **SSH**:



Copied
https://git-codecommit.us-east-1.amazonaws.com/v1/repos/LearnCodeCommit

Name	Description	Last modified	Clone URL
LearnCodeCommit	Repository to learn AWS CodeCommit.	Just now	HTTPS SSH

Figure 20.6

Selecting an appropriate repository and clicking the desired protocol (**HTTPS** or **SSH**) icon will not only show the URL on the top, but will also copy the URL into the memory directly to paste at the desired CLI or GUI.

Other developers in a team can access the same AWS CodeCommit repository by using the HTTPS/SSH protocol URL. This URL is also called a clone URL for the remote source code repository. Make sure developers have sufficient privileges to access the repository by creating an IAM user with appropriate minimum privileges to fulfill their day-to-day tasks.

CodeCommit events

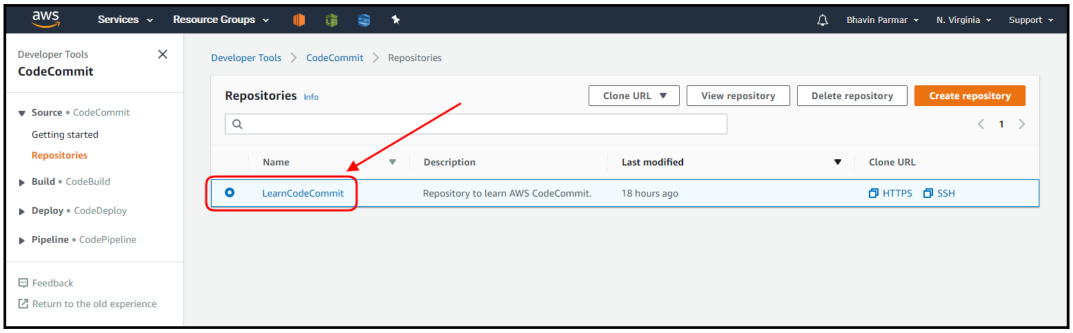
When a computer program or a service detects the occurrence of an action, it is called an **event**. Event-driven programming is a mechanism that determines the flow of the program based on the occurrence of events. CodeCommit supports event-driven programming. We will see some events and their details in this topic.

When a CodeCommit source code repository is ready with its credentials, each of the developers can access the repository to store project code and files. Developers can also configure an event notification and define their development, testing, or deployment workflow based on the events. Some of the notification events are described in the following points:

- **Pull request update events**: When you enable this event, subscribers receive an email notification in either of the following scenarios:
    - When a pull request is created or closed
    - When a pull request is updated with code changes
    - When a title or description of the pull request changes
- **Pull request comment events**: When you enable this event, subscribers receive an email notification when somebody comments or replies to a comment in a pull request.
- **Commit comment events**: When you enable this event, subscribers receive an email notification when somebody comments on a commit outside of a pull request. It includes comments on the following:
    - Line of code in a commit
    - Files in a commit
    - The commit itself

Enabling such events is helpful to update each of the developers in a team about the latest events, and accordingly they can perform their consecutive roles and tasks. Events can be enabled by executing the following steps:

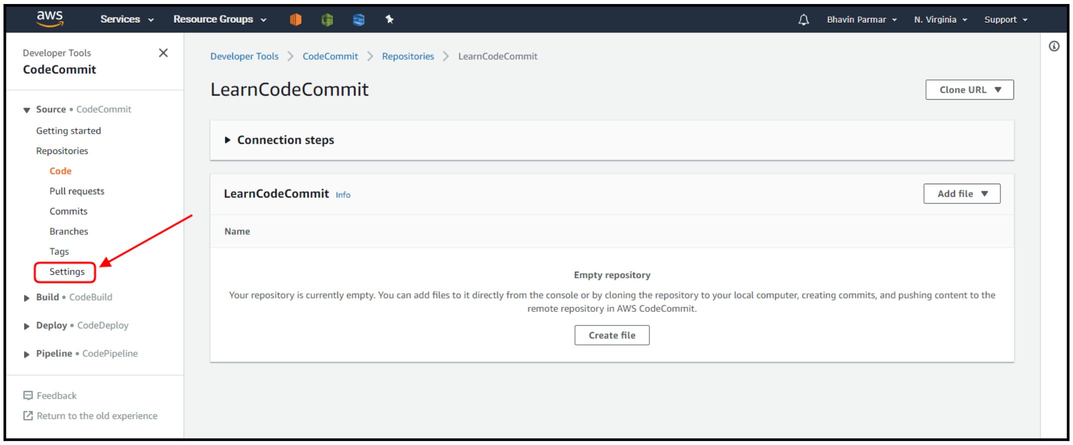
1. Log into the AWS account and open the AWS CodeCommit web console.
2. Click on the desired repository:



Name	Description	Last modified	Clone URL
LearnCodeCommit	Repository to learn AWS CodeCommit.	18 hours ago	HTTPS SSH

Figure 20.7

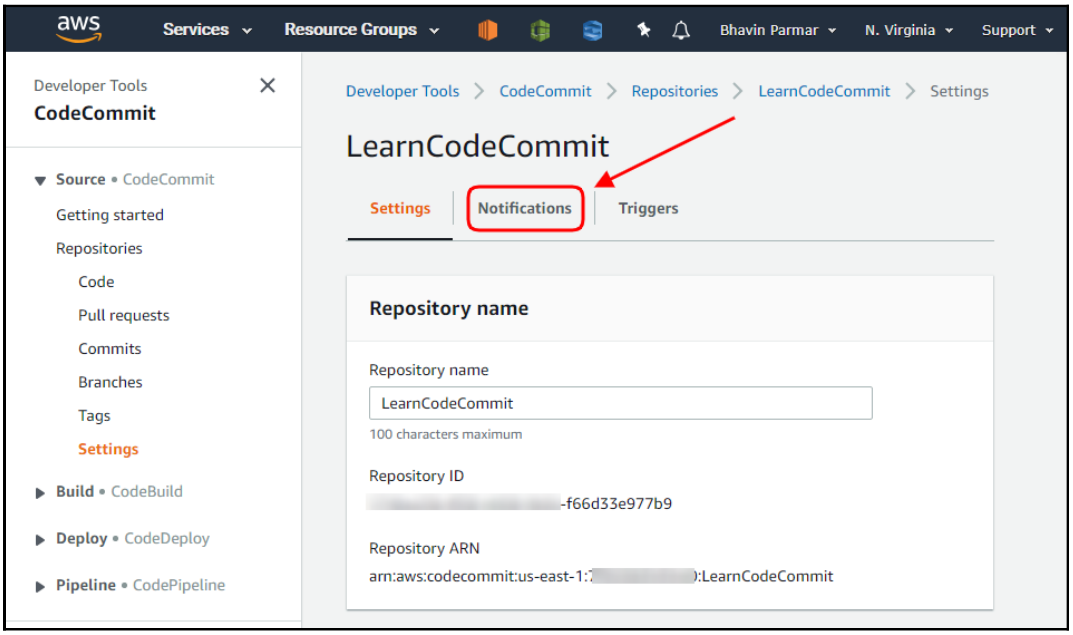
3. Select **Settings** from the left-hand pane:



Your repository is currently empty. You can add files to it directly from the console or by cloning the repository to your local computer, creating commits, and pushing content to the remote repository in AWS CodeCommit.

Figure 20.8

4. Select the **Notifications** tab:



Repository name: LearnCodeCommit

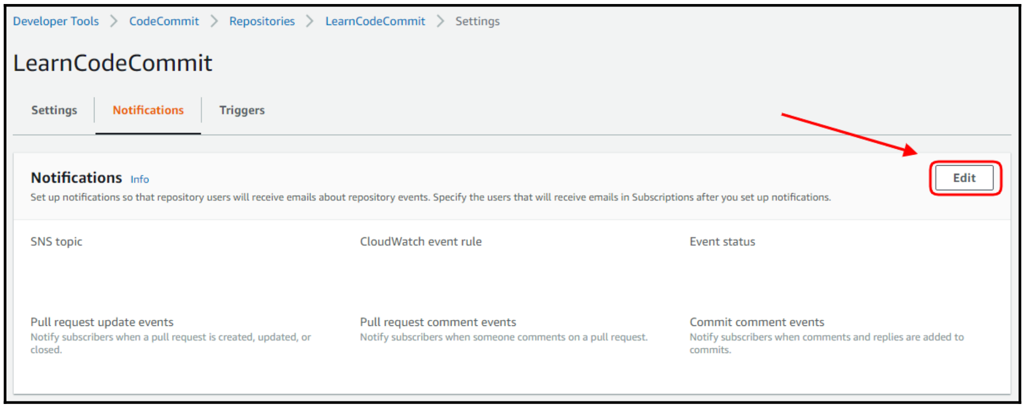
100 characters maximum

Repository ID: [redacted]-f66d33e977b9

Repository ARN: arn:aws:codecommit:us-east-1:[redacted]:LearnCodeCommit

Figure 20.9

5. Select **Edit** in the **Notifications** section:



Developer Tools > CodeCommit > Repositories > LearnCodeCommit > Settings

LearnCodeCommit

Settings | Notifications | Triggers

Notifications Info

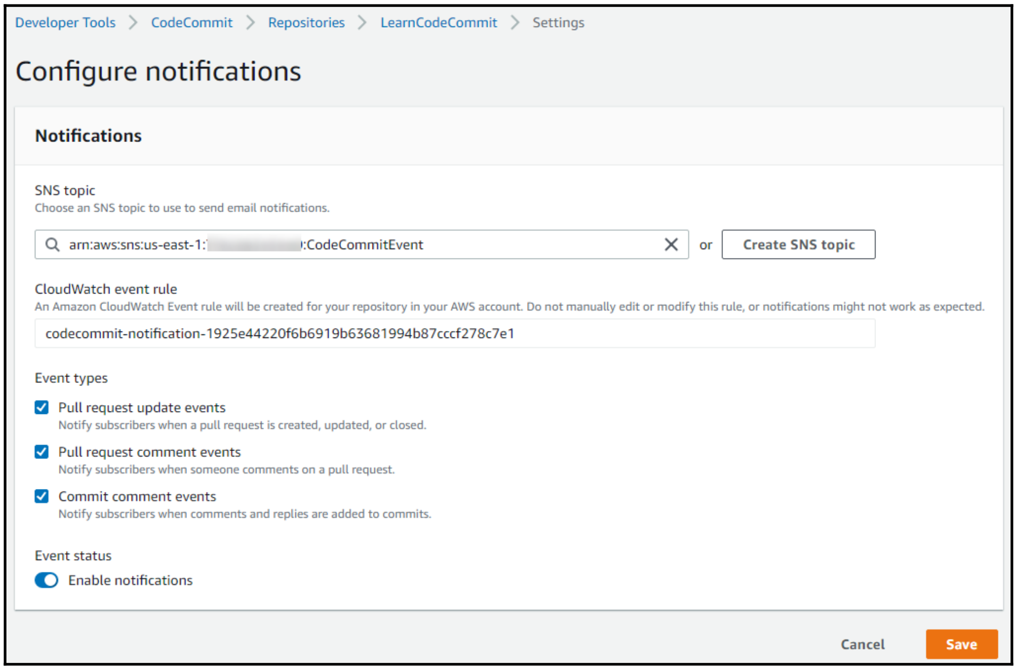
Edit

Set up notifications so that repository users will receive emails about repository events. Specify the users that will receive emails in Subscriptions after you set up notifications.

SNS topic	CloudWatch event rule	Event status
Pull request update events Notify subscribers when a pull request is created, updated, or closed.	Pull request comment events Notify subscribers when someone comments on a pull request.	Commit comment events Notify subscribers when comments and replies are added to commits.

Figure 20.10

6. Select the desired existing SNS topic. If a suitable SNS topic doesn't exist, it is also possible to create one. Enable the desired events:



Developer Tools > CodeCommit > Repositories > LearnCodeCommit > Settings

Configure notifications

Notifications

SNS topic
Choose an SNS topic to use to send email notifications.

arn:aws:sns:us-east-1:[redacted]:CodeCommitEvent or Create SNS topic

CloudWatch event rule
An Amazon CloudWatch Event rule will be created for your repository in your AWS account. Do not manually edit or modify this rule, or notifications might not work as expected.

codecommit-notification-1925e44220f6b6919b63681994b87cccf278c7e1

Event types

- ✓ Pull request update events
  Notify subscribers when a pull request is created, updated, or closed.
- ✓ Pull request comment events
  Notify subscribers when someone comments on a pull request.
- ✓ Commit comment events
  Notify subscribers when comments and replies are added to commits.

Event status

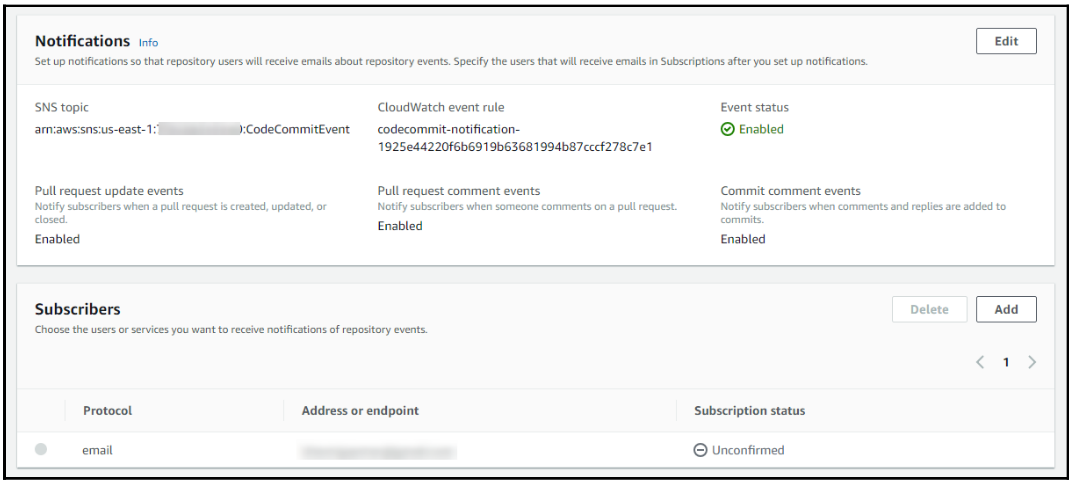
● Enable notifications

Cancel | Save

Figure 20.11

Once event notifications are enabled, use the **Enable notifications** toggle button to disable events temporarily if required. At present, to delete the events notification, you can directly delete the SNS topic from AWS SNS web console.

7. Click on **Save** to complete enabling events for the desired remote AWS CodeCommit source code repository. Once the events are successfully enabled, the list of **Subscribers** will be displayed:



Notifications Info

Set up notifications so that repository users will receive emails about repository events. Specify the users that will receive emails in Subscriptions after you set up notifications.

SNS topic
arn:aws:sns:us-east-1:[redacted]:CodeCommitEvent

CloudWatch event rule
codecommit-notification-1925e44220f6b6919b63681994b87cccf278c7e1

Event status
Enabled

Pull request update events
Notify subscribers when a pull request is created, updated, or closed.
Enabled

Pull request comment events
Notify subscribers when someone comments on a pull request.
Enabled

Commit comment events
Notify subscribers when comments and replies are added to commits.
Enabled

Subscribers

Choose the users or services you want to receive notifications of repository events.

Protocol	Address or endpoint	Subscription status
email	[redacted]	Unconfirmed

Figure 20.12

Working with branches

In real life, during a software development lifecycle, software development teams work in parallel on various versions, features, or bug fixes. Ultimately, each team of developers may be working on the same software but with separate plans, priorities, and deliverables. While they constantly develop their processes to achieve their designated task, they need a common source code repository. For this purpose, each repository (that is, Git repository) can have one main branch, which is usually called the **master** branch. The master branch contains production-ready software, ready to ship to the customer or within the business. For each version, feature, or bug fix, individual branches are created. While each team is working on their individual goals, they are also concerned with the software developed so far that they are using as a base for their own designated tasks.

While the development team works on new versions, features, or bug fixes, it may break. It is recommended that you keep each of the features, versions, and bug fixes independent and separate. Every team contributes individually contributing to their designated branch in the code repository. With a distributed version-control system, it becomes easy for developers to push or pull their code to keep their development in line with the overall development of the project.

Git branches are simple pointers or references to a commit. These branches make it possible for various developer teams to work in parallel on individual development tasks. Once they are done with their part, they can merge into the production branch.

The Git repository branching strategy may vary from project to project and organization to organization. It is beautifully explained by Vincent Driessen on his blog: `https://nvie.com/posts/a-successful-git-branching-model/`.

Once you have the remote repository cloned to a local machine, there may be a situation where you need to switch between various branches either to contribute your code or to refer to existing code.

To get a list of existing branches, we can use the `git branch` command:

```
$ git branch
* master
```

The `*` in the output indicates the present active branch. Logically, `git` is pointing to the master and all the file operations will be carried out against the same branch.

The preceding command only shows the branches in a local repository. To get the branch list of all the repositories (local and remote), we can run the `git branch --all` command:

```
$ git branch --all
* master
remotes/origin/HEAD -> origin/master
remotes/origin/master
```

To get the details only about remote branches, we can use the `git branch -r` command:

```
$ git branch -r
origin/HEAD -> origin/master
origin/master
```

Use the AWS CLI `aws codecommit list-branches –repository-name LearnCodeCommit` to returns information about a repository branch.

To create a new branch while keeping logical control over the present branch, use the `git branch` command:

```
$ git branch feature-X     # create a new branch named feature-X
$ git branch               # list branches in a current local repo
feature-X
* master
```

You can check out your code repository using the `git checkout` command with the –`b` parameter. The syntax is as follows:

```
$ git checkout -b feature-Y
Switched to a new branch 'feature-Y'
```

The preceding command clearly indicates that logical control has been switched to a newly created branch, `feature-Y`. The –`b` parameter in the preceding command is used to create a branch with the given name; in the previous example the branch name is given as `feature-Y`. By running the `git status` command, we can observe that `*` has been shifted from `master` to `feature-Y`. This indicates that the current branch is `feature-Y`:

```
$ git branch
feature-X
* feature-Y
master
```

Git has a special pointer called `HEAD` to remember on which branch—specifically, on which commit version—the developer was.

In general, the `git checkout` command can be used to switch from one branch to another or to bring a file to one of its earlier states during a software development on a local repository. To delete an existing branch from a local repository, we can run the following command:

```
$ git branch -d feature-Y
output: Deleted branch feature-Y (was 394963d).
```

The output indicates that the `feature-Y` branch with the `394963d` commit reference is deleted.

Working with pull requests

The `git pull` command is used to update the local version of a code repository with code from the remote repository:

```
*** Changes in progress
```

Once a developer or a group of developers completes development on a feature branch, they need to merge the feature branch with the original branch. The branch can be merged using the `git branch` or `git checkout` command. In a GitLab, this process is called a **merge request**, and the same process is called a **pull request** in AWS CodeCommit and GitHub. Both mean one and the same thing: pulling changes from another branch (that is, a child branch) to fork into a parent branch with existing updated code.

When you are merging a child branch with a parent branch after a long time, it is suggested that you pull the latest code on the master branch from the remote repository.

The `git merge` command can be used to merge the two branches (that is, the specified branch with the current branch).

Before executing the `git merge` command, make sure you are on the right branch and that `HEAD` is pointing to the correct merge-receiving branch. By executing a `git status` command, we can verify whether `HEAD` is pointing to the correct merge-branch. If required, use the `git checkout` command to switch the branch.

Ideally, it is recommended you merge branches many times during a day to keep a minimum deviation of code between the two branches. This command will combine multiple sequences of commits into one unified history. Usually, `git merge` will take two commit pointers at the branch tip. In our example, it is the **Master tip** and **Feature X**, as shown in the following diagram. It will find a common base commit between them:

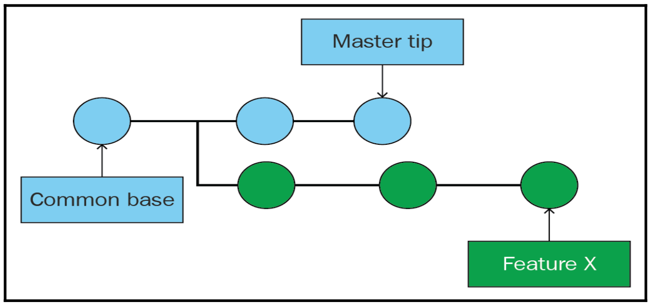


Figure 20.12

Once the **Common base** has found commit phase, the **Common base** will create a new merge commit to combine the changes in each queued merge commit sequence, as shown in the following diagram:

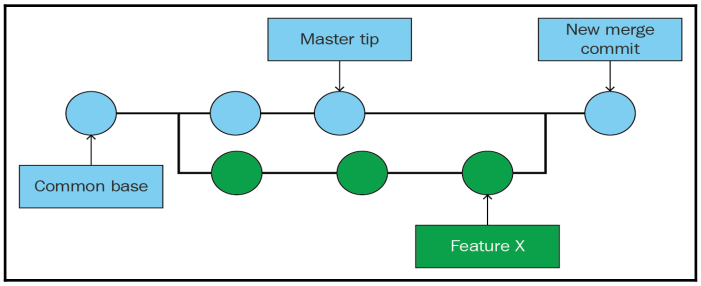


Figure 20.13

Consider the following command:

```
$ git merge feature-X
Merge made by the 'recursive' strategy.
file1 | 0
t1 | 0
2 files changed, 0 insertions(+), 0 deletions(-)
create mode 100644 file1
create mode 100644 t1
```

The preceding command's output clearly indicates a merge took place using the recursive method. The `git merge` command will automatically select a merge strategy unless it is explicitly specified using the `–s` parameter. The strategy types are recursive, resolve, octopus, ours, and subtree.

Merge commits are different from other commits. Since a merge commit merges two branches, it has two parent commits. When you create a merge commit, Git automatically tries to merge the histories of the commits you want to merge. If Git encounters some data that was changed in the history of both branches, it cannot combine this data in the merge. This scenario is called version-control conflict. Git needs manual intervention in order to continue merging the commits.

Summary

- AWS CodeCommit is highly available, durable, and a fully-managed source code service.
- CodeCommit is compliant with regulatory compliance standards such as HIPAA and PCI DSS.
- AWS CodeCommit can be accessed using a web console and also using the Git command line or AWS CLI.
- Developers can sync up their local repository with the remote repository using `git pull` command.
- Executing Git commands require authentication. Whenever you use Git commands, Git prompts you for your username and password.
- When working with a large number of files and doing constant development, it is recommended that you use CLI or IDE integration with CodeCommit.
- To work with remote the AWS CodeCommit source code repository using the CLI, you need to configure a local Git client to communicate with it.
- AWS IAM provides three types of credential: IAM-generated username and password, SSH key and AWS Access key ID and secret key.
- Git credentials can be downloaded only during the creation of the credentials. You cannot download them later on. If credentials are lost, you need to reset the password. To avoid this situation, make sure the downloaded credentials are stored in a safe place.

- The repository is a secure and reliable place to store code and files for the project.
- Developers can also configure an event notification and define their development, testing, or deployment workflow based on the events.
- When a computer program or a service detects occurrence of an action, it is called an event.
- Event-driven programming is a mechanism that determines the flow of the program based on the occurrence of events.
- A repository can be divided into multiple branches depending upon organizational requirement, for instance, feature, dev, QA, staging, production branch, and so on.
- The Git repository branching strategy may vary from project to project and from organization to organization.
- To get a list of existing branches, we can use the `git branch` command. The `git checkout` command can be used to switch from one branch to another or to bring a file to one of its earlier states.