

Repo

Best Practices

Issue 01
Date 2023-05-05



Copyright © Huawei Technologies Co., Ltd. 2023. All rights reserved.

No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Huawei Technologies Co., Ltd.

Trademarks and Permissions



HUAWEI and other Huawei trademarks are trademarks of Huawei Technologies Co., Ltd.

All other trademarks and trade names mentioned in this document are the property of their respective holders.

Notice

The purchased products, services and features are stipulated by the contract made between Huawei and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

Contents

1 Git on CodeArts Repo.....	1
1.1 Overview.....	1
1.2 CodeArts Repo Cloud Operations.....	3
1.3 Local Development on Git.....	8

1 Git on CodeArts Repo

[Overview](#)

[CodeArts Repo Cloud Operations](#)

[Local Development on Git](#)

1.1 Overview

Purpose

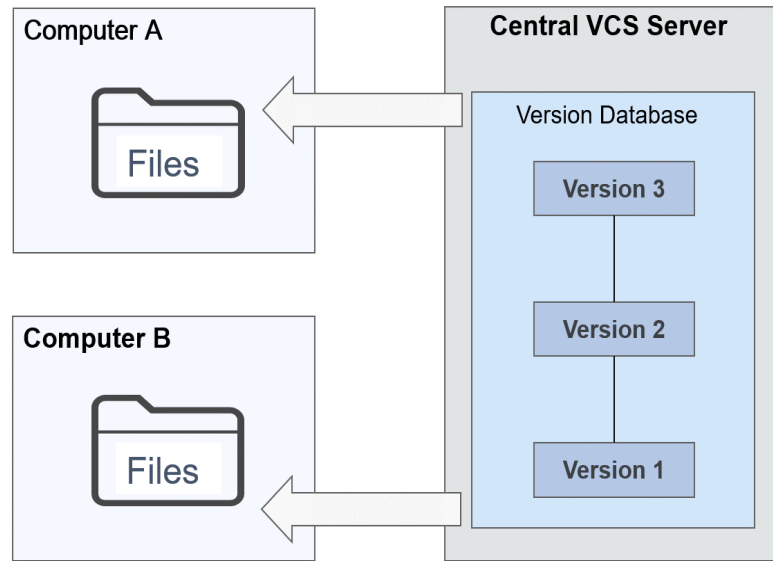
This document is intended to help developers who are interested in Git to better use Git and apply Git in the CodeArts practice.

Git Overview

Git is a distributed version control system (VCS). VCSs manage all code revisions during software development. They store and track changes to files, and records the development and maintenance of multiple versions. In fact, they can be used to manage any helpful documents apart from code files. VCSs are classified into centralized version control systems (CVCSs) and distributed version control systems (DVCSs).

Centralized Version Control Systems

A CVCS has a central server that contains all development data, and a number of clients that store snapshots of the files in the central server at one point. That means the change history of project files is kept only in the central server, but not on the clients. Therefore, developers must pull the latest version of files from the central server each time before starting their work.



Common CVCSs include CVS, VSS, SVN, and ClearCase.

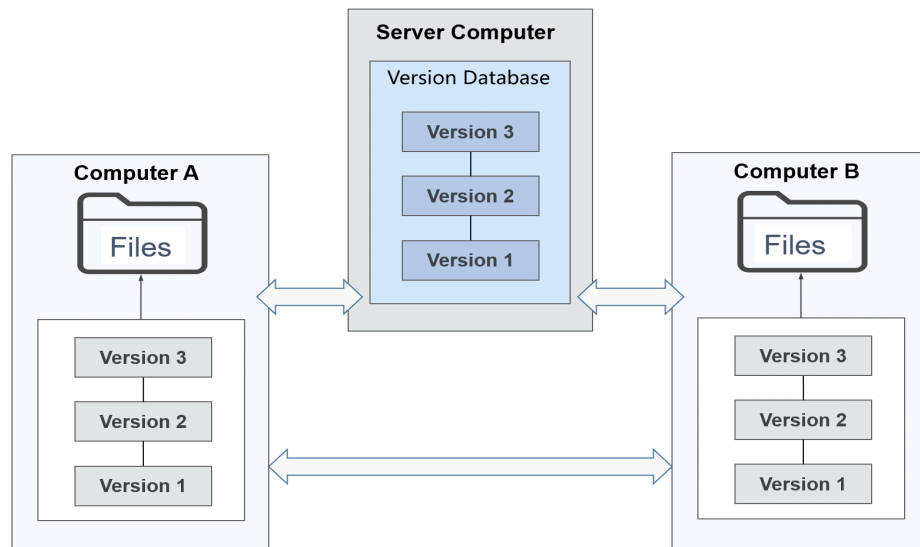
The advantages and disadvantages of CVCSs are listed below.

Table 1-1 Advantages and disadvantages of CVCSs

Advantages	Disadvantages
<ul style="list-style-type: none">• Easy to use.• Granular permission control on the directory level.• Large storage space is not required on the clients because they do not store the entire copy of the code repository.	<ul style="list-style-type: none">• A highly stable network is required since developers must work online.• If the server breaks down, the development work is suspended.• All data will be lost if the hard disk of the central server is corrupted and no proper backup is kept.

Distributed Version Control Systems

In DVCSs, every client is a complete mirror of the code repository. All data, including the change history of project files, is stored on each client. In other words, there is not a central server in this distributed system. Some companies which use Git may call a computer as the "central server". However, that "central server" is in nature the same as other clients except for the fact that it is used to manage collaboration.



Common DVCSs include Git, Mercurial, Bazaar, and BitKeeper. The advantages and disadvantages of DVCSs are listed below.

Table 1-2 Advantages and disadvantages of DVCSs

Advantages	Disadvantages
<ul style="list-style-type: none"> • Each client stores a complete copy of the code repository, including tags, branches, and version records. • Offline commits enable easy cross-distance collaboration. • Branches are cheap to create and destroy, and fast to check out. 	<ul style="list-style-type: none"> • High learning thresholds. • Branches can be created only for the entire repository but not for individual directories.

1.2 CodeArts Repo Cloud Operations

Preparations

- You have registered an account for CodeArts Repo.
- You already have a [Git client](#).
- A [project](#) has been created.

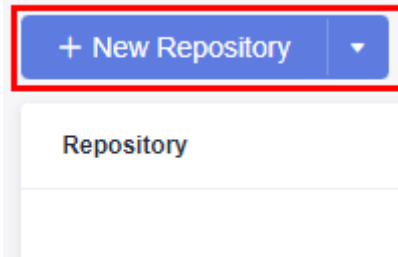
Cloud Repositories

CodeArts Repo allows you to create, clone, and manage cloud repositories. You can manage branches, tags, repository members, and keys, and perform

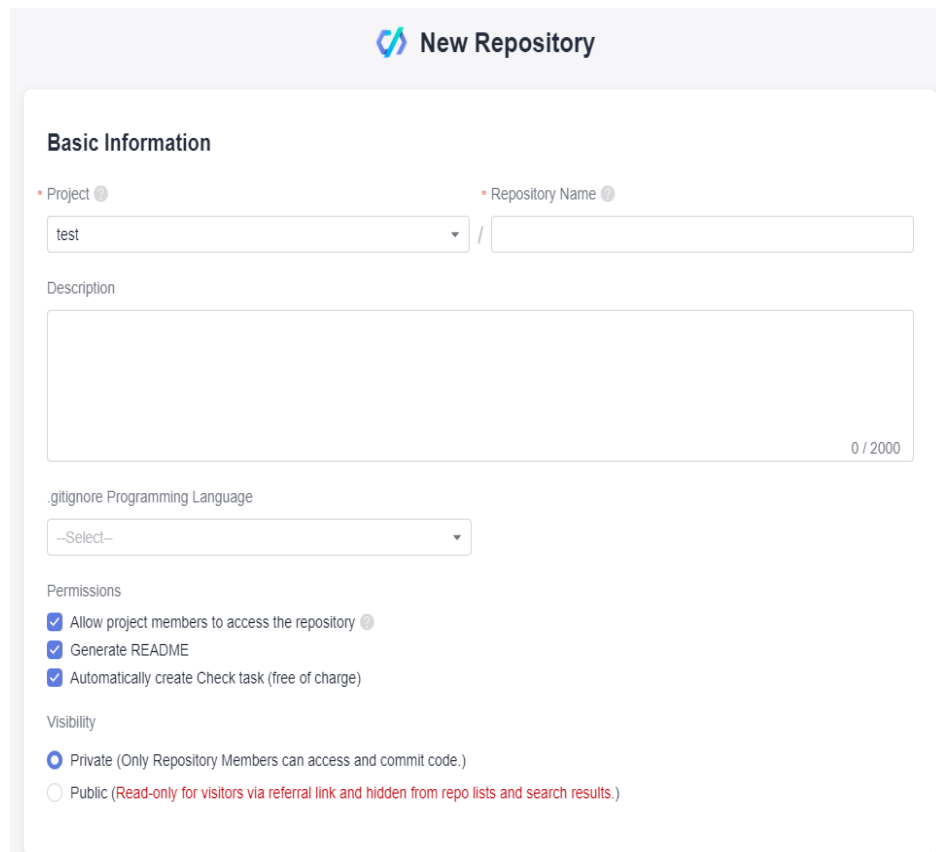
operations on code, including committing, pulling, pushing, viewing, and online editing. For more details about cloud repositories, see [Service Overview](#).

Creating an Empty Repository

1. On the CodeArts Repo homepage, click **New Repository**.



2. Enter the basic repository information, as shown in the following figure.

A screenshot of the 'New Repository' form. The form is titled 'New Repository' with a blue icon. It has a 'Basic Information' section. Under 'Project', there is a dropdown menu with 'test' selected. To the right is a 'Repository Name' input field. Below these is a 'Description' text area with a '0 / 2000' character count. There is a '.gitignore Programming Language' dropdown menu with '--Select--' selected. The 'Permissions' section has three checked checkboxes: 'Allow project members to access the repository', 'Generate README', and 'Automatically create Check task (free of charge)'. The 'Visibility' section has two radio buttons: 'Private (Only Repository Members can access and commit code.)' which is selected, and 'Public (Read-only for visitors via referral link and hidden from repo lists and search results.)'.

3. Click **OK** to create the repository. The repository list page is displayed.

Setting the SSH Keys or HTTPS Password

SSH keys and HTTPS password are credentials for communication between a client and server. Set them first before you clone or push a repository on your computer.

Setting SSH Keys

SSH keys are used when a client communicates with CodeArts Repo over the SSH protocol. If you have downloaded Git Bash for Windows and generated an SSH key pair in the process, skip this section.

- Step 1** Open the Git client (Git Bash or Linux CLI), enter the following command, and press **Enter** for three times.

```
ssh-keygen -t rsa -C "<email address>"
```

The generated SSH key pair is stored in `~/.ssh/id_rsa` and `~/.ssh/id_rsa.pub` by default.

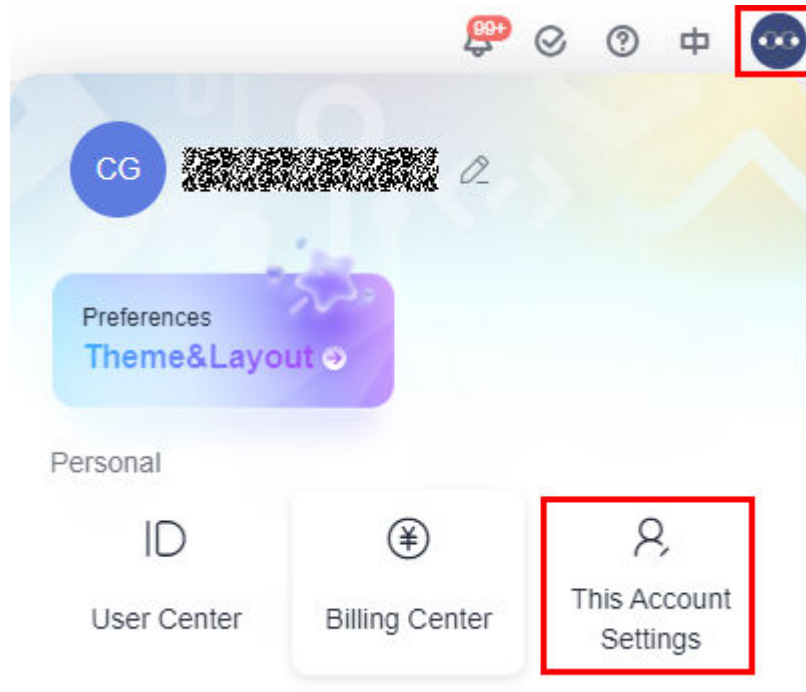


```
MINGW32 /
$ ssh-keygen -t rsa -C "devcloud@huaweicloud.com"
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/.../.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /c/Users/.../.ssh/id_rsa.
Your public key has been saved in /c/Users/.../.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:cgaQaYlU5pokqwtvJ2ZaTuyS+LwueWBpZgERzcfZ7mY devcloud@huaweicloud.com
The key's randomart image is:
+---[RSA 2048]-----+
|o*.++*
|. +00..
|o .o...
|= o ..
|. = .. S
|oB E+
|O++ o
|B0* .
|o@O+
+-----[SHA256]-----+
```

- Step 2** Add the SSH key to CodeArts Repo.

Open the Git client (Git Bash or Linux CLI) and run the following command to print the SSH key in `~/.ssh/id_rsa.pub`.

- Step 3** Copy the preceding SSH key content, log in to your CodeArts Repo repository list page, click the nickname in the upper right corner, and choose **This Account Settings > SSH Keys**.



Step 4 On the **SSK Keys** page, click **Add SSH Key**. In the displayed **Add SSH Key** page, enter the information shown in the following figure and click **OK**. A message is displayed, indicating that the operation is successful.

Add SSH Key

For details about how to generate an SSH key, see the guidance below.

The SSH key has been added. You can proceed to set an HTTPS password.

----End

Setting an HTTPS Password

An HTTPS password is used when a client communicates with CodeArts Repo over HTTPS. To set an HTTPS password, perform the following steps:

Step 1 Log in to the CodeArts Repo service repository list page, click the alias in the upper right corner, and choose **This Account Settings > HTTP Password**.

1.3 Local Development on Git

Background

After creating a repository with a README file in CodeArts Repo, an architect or project manager pushes the architecture code to the repository. Other developers then clone the architecture code to their local computers for incremental development.

NOTE

- Git supports code transmission over SSH and HTTPS. The SSH protocol is used as an example.
- If you want to use the HTTPS protocol, download the HTTPS password, and enter the HTTPS username and password when cloning or pushing code.
- The SSH URL and HTTPS URL of the same repository are different.

Pushing Architecture Code

1. Open the architecture code on the local computer. Ensure that the name of the root directory (CodeArts) is the same as that of the code repository created in the cloud. Right-click in the root directory and choose **Git Bash Here**.
2. Push local code to the cloud.

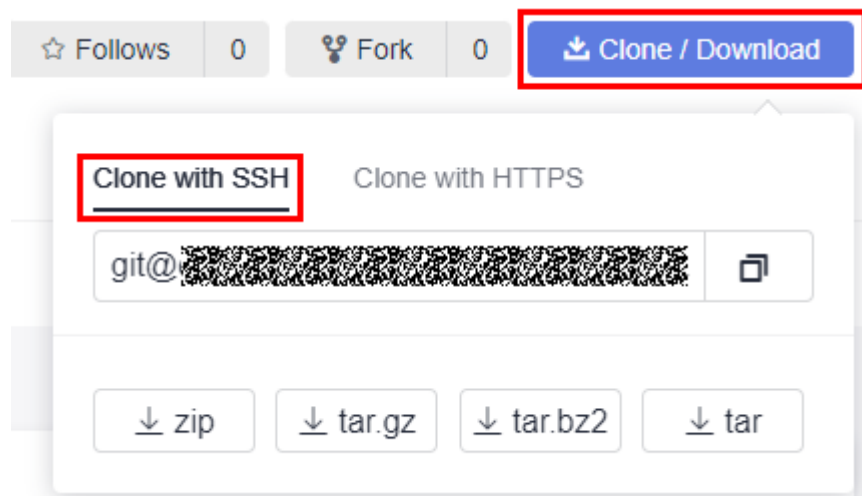
Run commands on Git Bash as instructed below.

- a. Initialize a local code repository. After this command is executed, a **.git** directory is generated in **D:/code/repo1/**.
`$ git init`

- b. Associate the local repository with the one in the cloud.

```
$ git remote add origin repoUrl
```

You can switch to the repository details page, click **Clone/Download**, and click the highlighted tab in the following figure to obtain the *repoUrl* value.



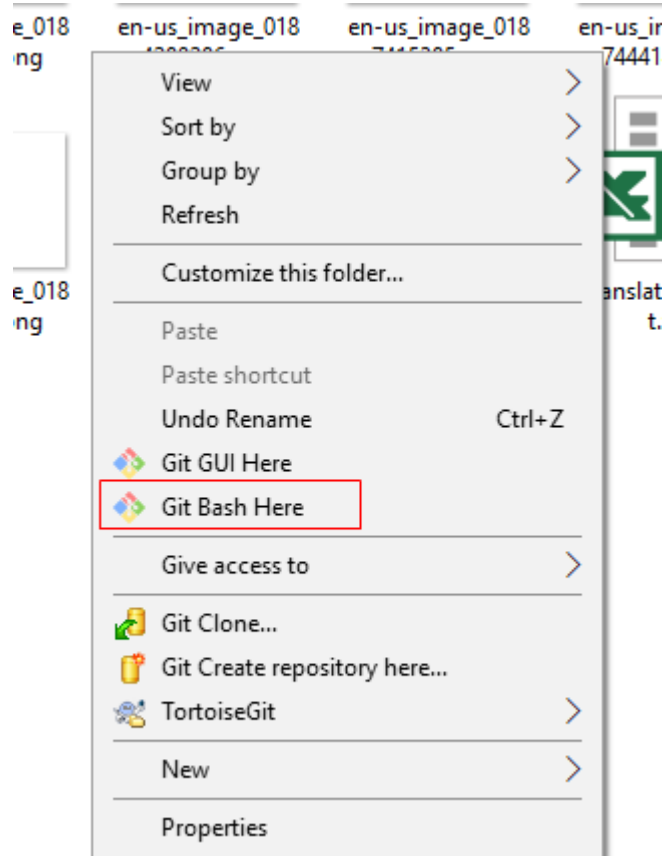
- c. Push code to the cloud repository.

```
$ git add .  
$ git commit -m "init project"  
$ git branch --set-upstream-to=origin/master master  
$ git pull --rebase  
$ git push
```

Cloning Code

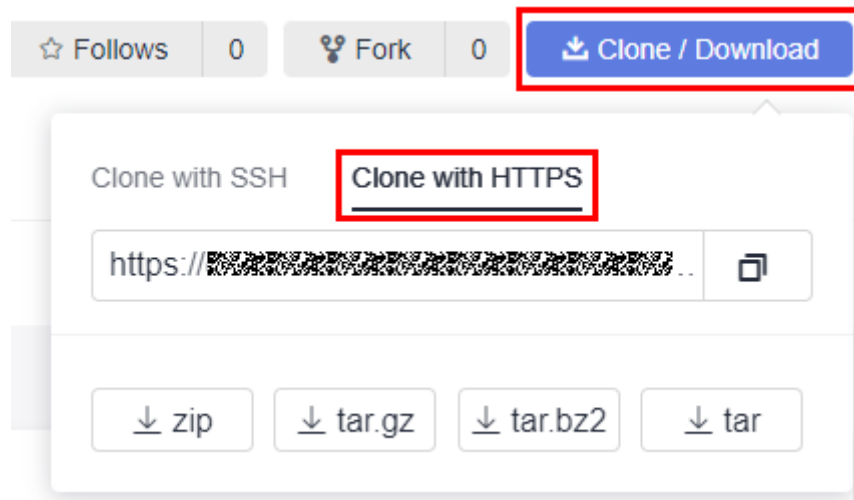
Clone the architecture code from the cloud to the local computer.

1. In the directory where you want to clone the code, right click and choose **Git Bash Here**.



2. Run the following command to clone the repository. Click **Clone/Download** and click the highlighted tab in the following figure to obtain the *repoUrl* value

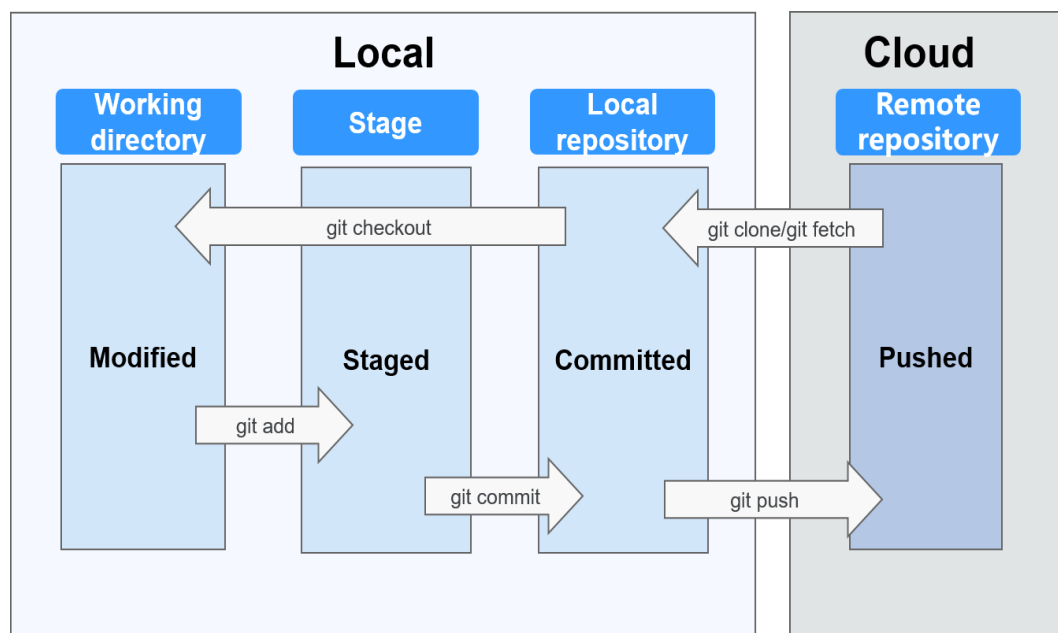
```
$ git clone repoUrl//Clone the code from the remote repository to the local computer.
```



Committing Code

A change travels from the working directory, stage, and local repository before being pushed to the remote repository.

Executing corresponding git commands can move a file between the four areas.



The following commands are involved:

1. **#git add/rm filename** //Add changes from the working directory to the stage after creating, editing, or deleting files.
2. **#git commit -m "commit message"** //Commit the files from the stage to the local repository.
3. **#git push** //Push the files from the local repository to the remote one.

Performing Branch Operations

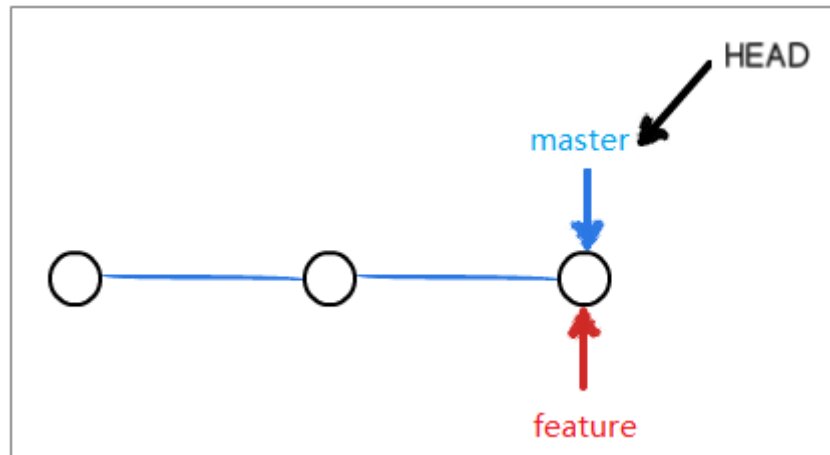
- Create a branch.

In Git, creating a branch is not to copy a repository, but to create a HEAD, a movable pointer pointing to the last commit. A branch in nature is a file that contains the 40-byte SHA-1 checksum of the commit it points to.

```
#git branch branchName commitID
```

A new branch is pulled based on the specified commit ID. If no commit ID is specified, the branch is pulled from the commit that HEAD points to.

For example, to create a feature branch, run **git branch feature**.

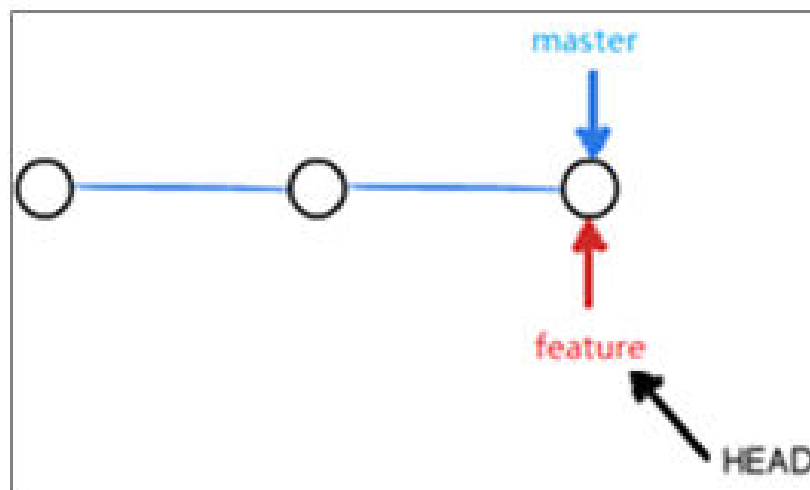


- Check out a branch.

Run the following command:

```
#git checkout branchName
```

For example, to check out the feature branch, run **git checkout feature**.



- Integrate branches.

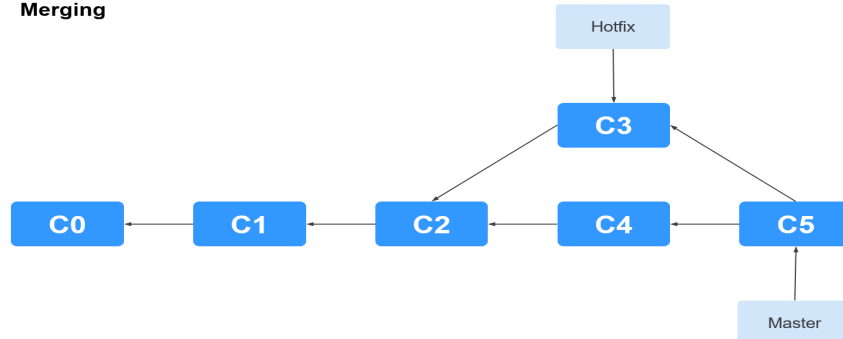
There are two ways to integrate changes from one branch to another: **git merge** and **git rebase**. The following describes the differences between them.

Assume that C4 and C3 are added to the master branch and hotfix branch respectively. The hotfix branch is now ready to be integrated to the master branch.

- a. Three-way merge integrates C3, C4, and their most recent common ancestor C2. Merging is simple to operate, but a new commit C5 is created, resulting in a less readable commit history.

```
#git checkout master
#git merge hotfix
```

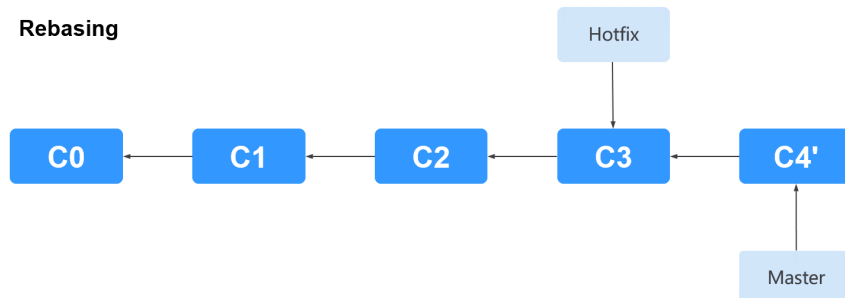
Merging



- b. Git rebase saves the changes introduced to C4 as a patch in the `.git/rebase` directory, synchronizes the patch C4' to the hotfix branch, and applies the patch on top of C3.

```
#git checkout master
#git rebase hotfix
```

Rebasing



- Resolve conflicts.
 - a. Scenario 1: The same line of code is changed in both the two branches to merge.

```
$ cat doc/README.txt
User1 hacked.
<<<<<< HEAD
Hello, user2. # Changes on the current branch
*****
Hello, user1. # Changes on the source branch
>>>>>> a123390b8936882bd53033a582ab540850b6b5fb
User2 hacked.
User2 hacked again.
```

Solution

- i. Manually merge the change that you think is proper.
- ii. Commit the change.

- b. Scenario 2: A file is renamed in two different ways.

Solution

- i. Check which name is correct and delete the incorrect one.
- ii. Commit the change.