

CodeArts Repo

Best Practices

Issue 01
Date 2023-09-05



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.

Security Declaration

Vulnerability

Huawei's regulations on product vulnerability management are subject to "Vul. Response Process". For details about the policy, see the following website:<https://www.huawei.com/en/psirt/vul-response-process>
For enterprise customers who need to obtain vulnerability information, visit:<https://securitybulletin.huawei.com/enterprise/en/security-advisory>

Contents

1 Git on CodeArts Repo.....	1
1.1 Overview.....	1
1.2 Cloud Repository Operations.....	3
1.3 Local Development on Git.....	8
2 Migrating the Repository to CodeArts Repo.....	14
3 Code Review Practice.....	20

1 Git on CodeArts Repo

[Overview](#)

[Cloud Repository Operations](#)

[Local Development on Git](#)

1.1 Overview

Purpose

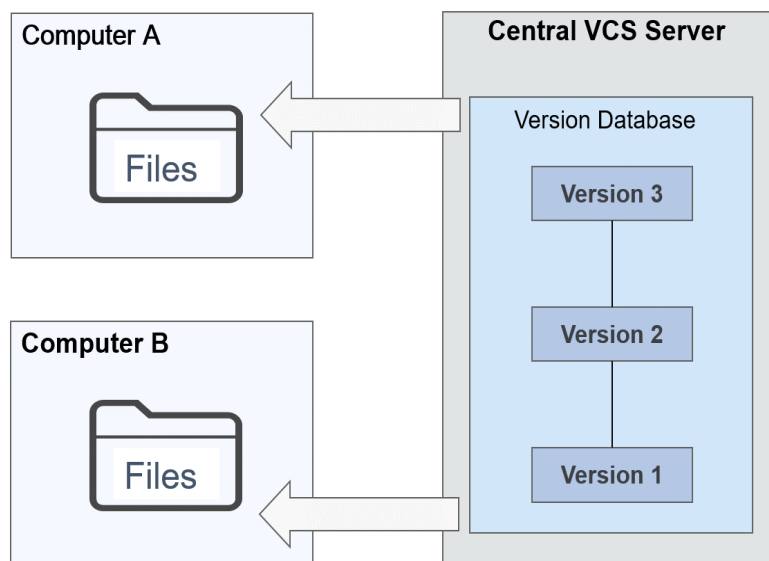
This document is intended to help developers who are interested in Git to better use Git and apply Git in the CodeArts practices.

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 record the development and maintenance of multiple versions. 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 (computers) 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 Concurrent Versions System (CVS), Visual SourceSafe (VSS), Subversion (SVN), and ClearCase.

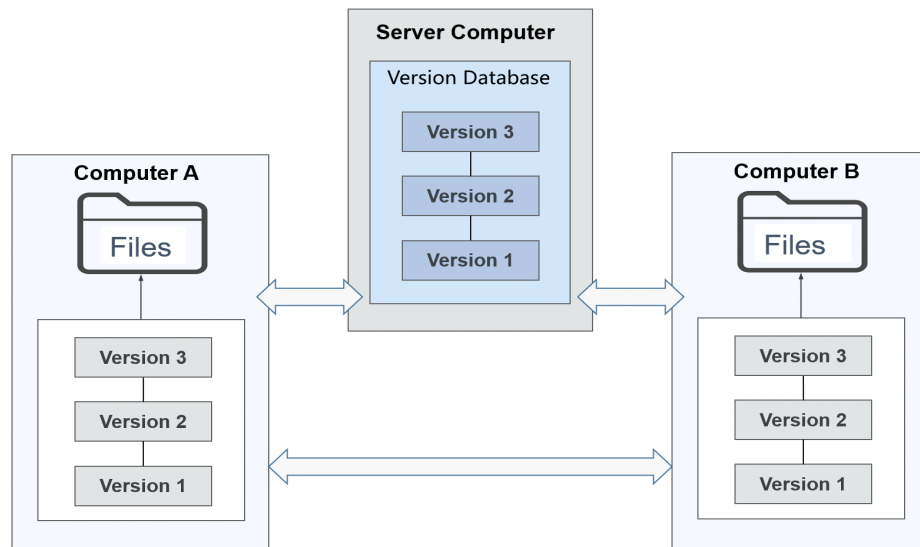
The advantages and disadvantages of CVCSs are listed below.

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 files.	<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 created and deleted at a low cost, and are fast to be checked 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 Cloud Repository Operations

Preparations

- You have registered an account for CodeArts Repo.
- You have [Installed and configured Git](#)
- A has been created.

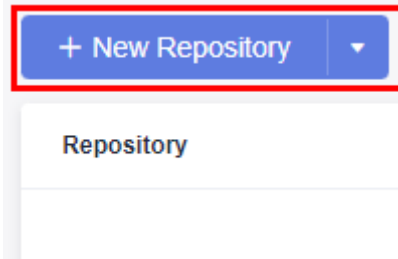
Cloud Repositories

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

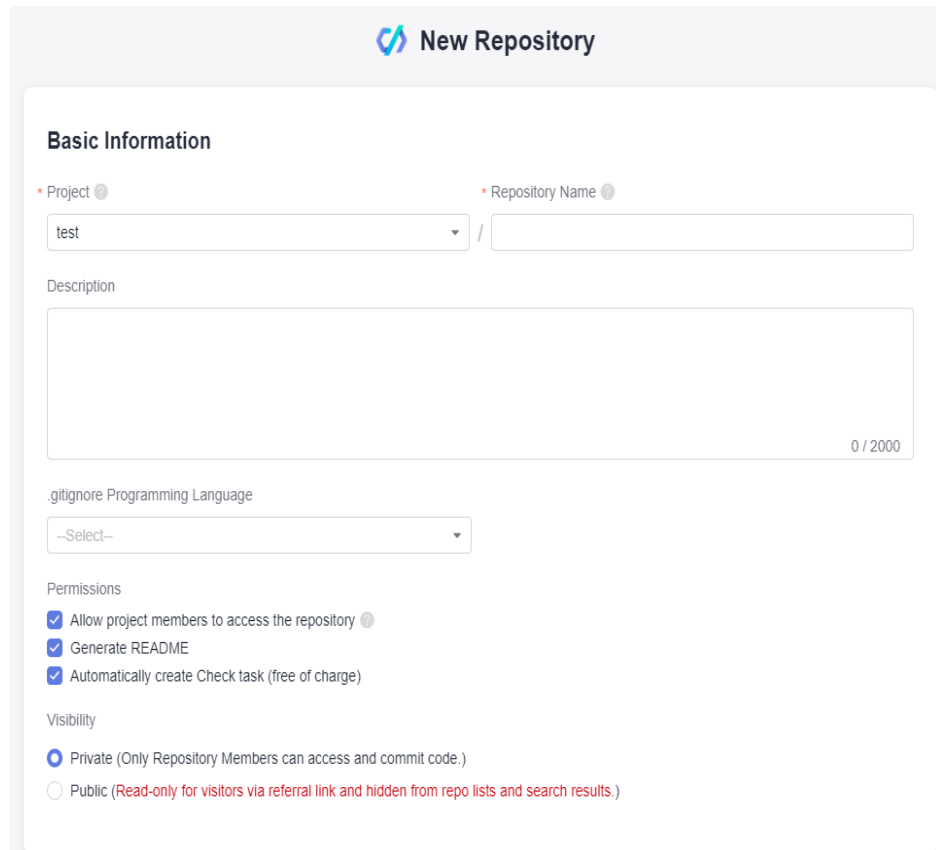
operations on code, including committing, pulling, pushing, viewing, and online editing. For more details about cloud repositories, see [Product Overview](#).

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 several sections: 'Basic Information' with 'Project' (a dropdown menu showing 'test') and 'Repository Name' (a text input field); 'Description' (a large text area with a '0 / 2000' character count); '.gitignore Programming Language' (a dropdown menu showing '--Select--'); 'Permissions' with three checked checkboxes: 'Allow project members to access the repository', 'Generate README', and 'Automatically create Check task (free of charge)'; and 'Visibility' with two radio buttons: 'Private (Only Repository Members can access and commit code.)' (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 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 "<your_email_address>"
```

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

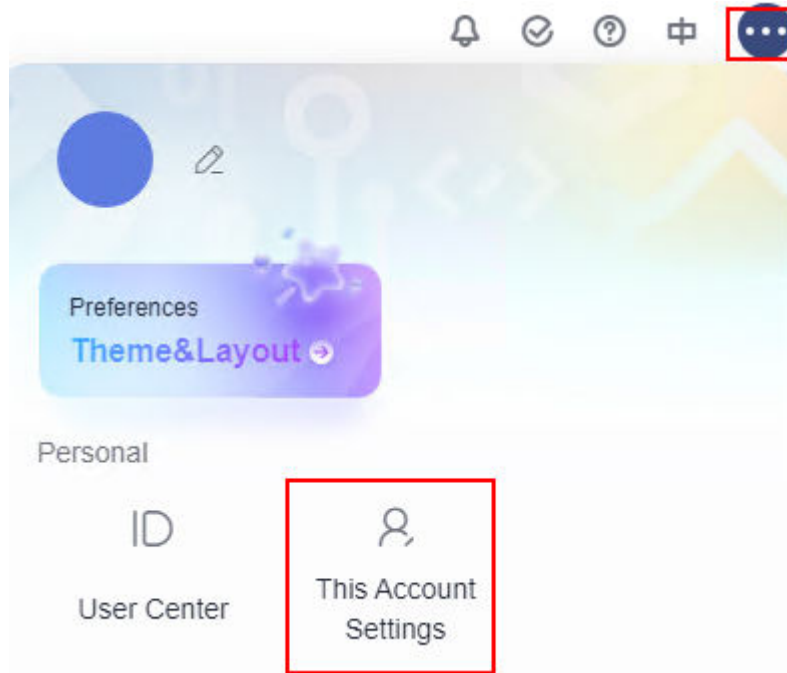


```
MINGW32 /
$ ssh-keygen -t rsa -C "devcloud@huaweicloud.com"
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/.../.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /c/Users/.../.ssh/id_rsa.
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]-----+
MINGW32 /
```

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

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

- Step 3** Copy the preceding SSH key, log in to your CodeArts Repo, click the alias 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.

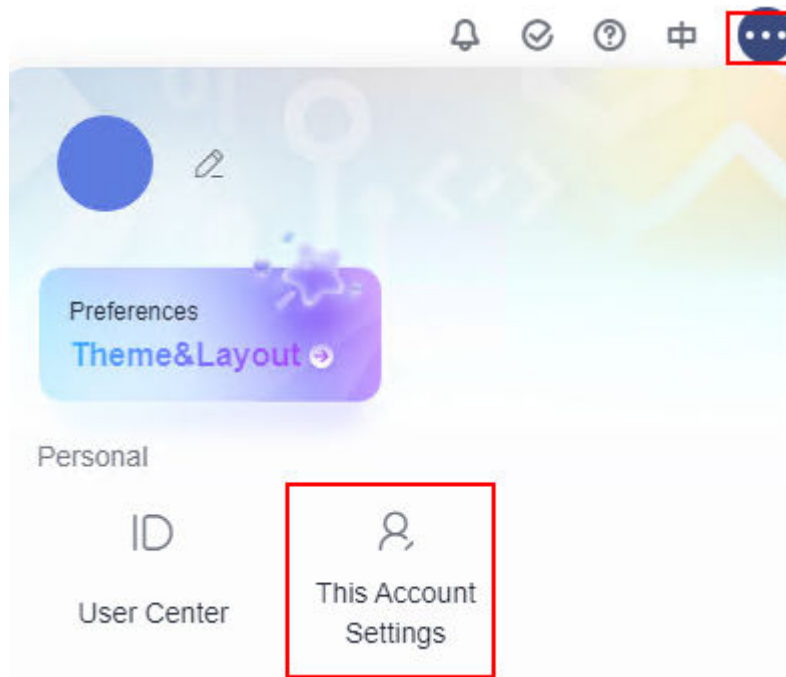
The SSH key has been added. You can proceed to set an HTTPS password.

----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, click the alias in the upper right corner, and choose **This Account Settings > HTTP Password**.



Step 2 Click **Set new password**, and then click **Change** to change the password. (If you have set an HTTPS password and are using it, click **Change**.)

HTTPS Password

Change the HTTPS password below if needed.

Username [REDACTED]

* Verification Code: [Send Email](#)

* New Password: ⓘ

* Confirm Password:

I have read and agree to the [Privacy Statement](#) and [CodeArts Service Statement](#).

OK Cancel

Step 3 Enter the new password and email verification code, select **I have read and agree to the Privacy Statement and CodeArts Service Statement**, and click **OK**. A message is displayed, indicating that the operation is successful.

----End

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 is the same as that of the code repository created in the cloud. Right-click on the root directory and choose **Git Bash Here**.
2. Push local code to the cloud.

Run commands on **Git Bash** as instructed below.

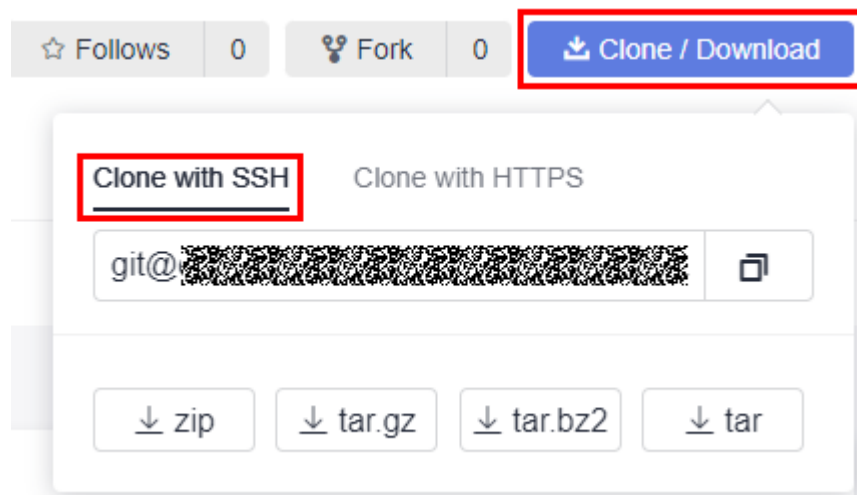
- a. Initialize a local code repository. After this command is executed, a **.git** directory is generated in **D:/code/repo1/**.

```
$ 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.

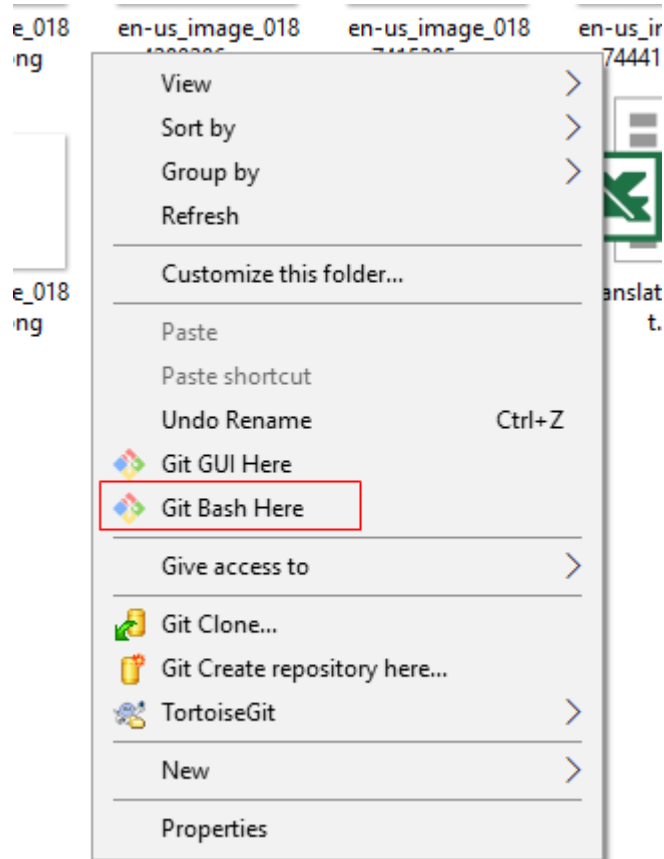
```
$ git add .  
$ git commit -m "init project"
```

```
$ git branch --set-upstream-to=origin/master master  
$ git pull --rebase  
$ git push
```

Cloning Code

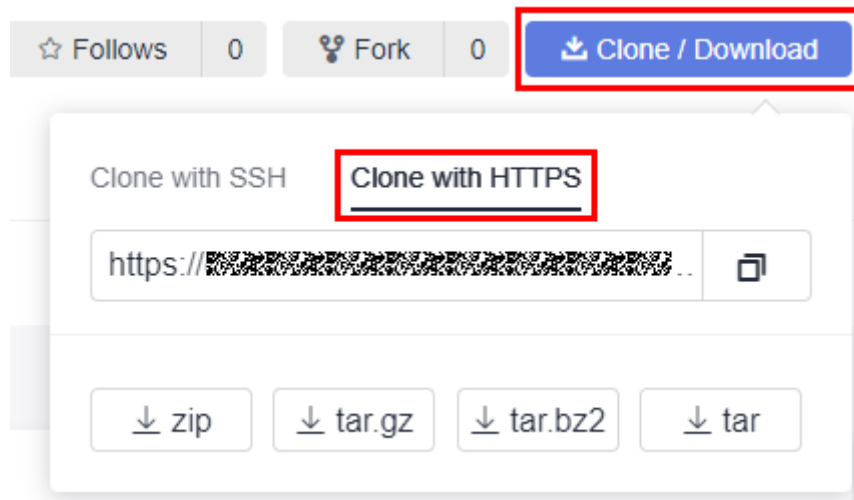
Clone the architecture code from the cloud to the local computer.

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

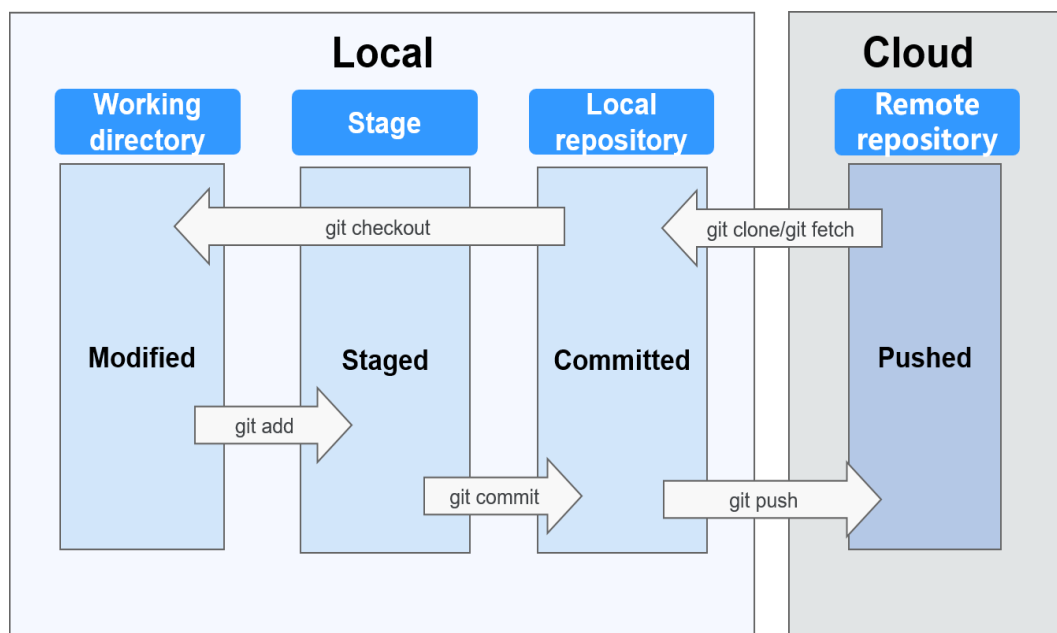
```
$ git clone repoUrl//Clone the code from the remote repository to the local computer.
```



Committing Code

A change travels from the working directory, stage, local repository 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

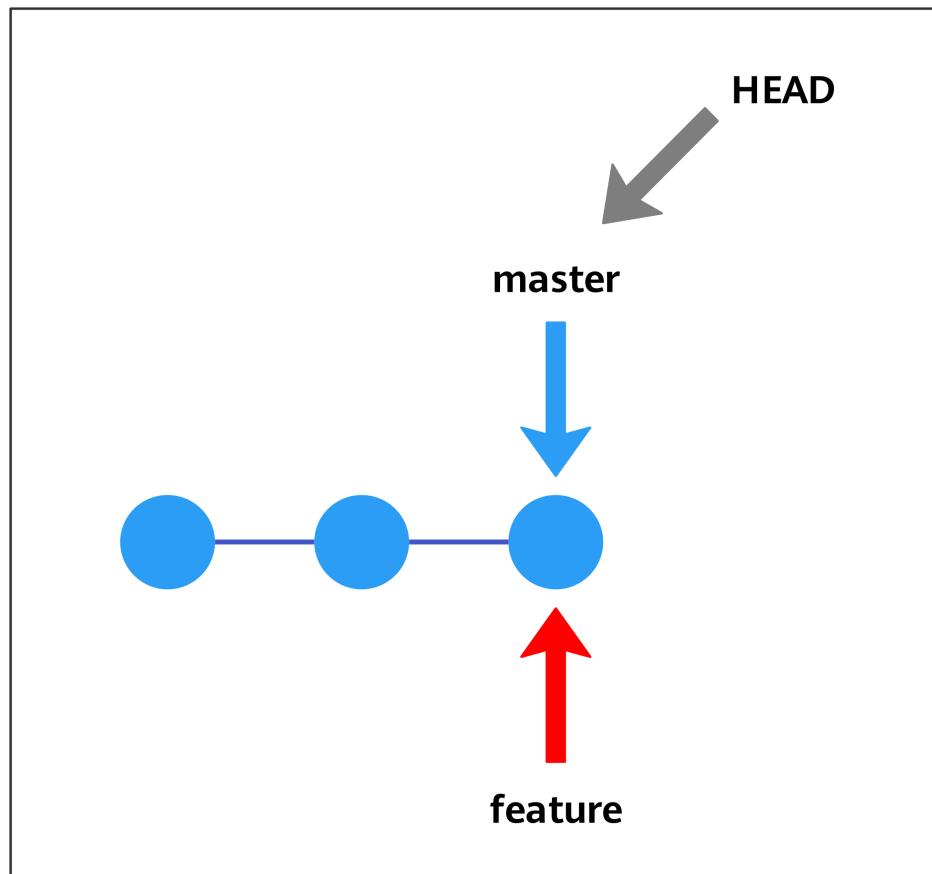
- Create a branch.

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**.

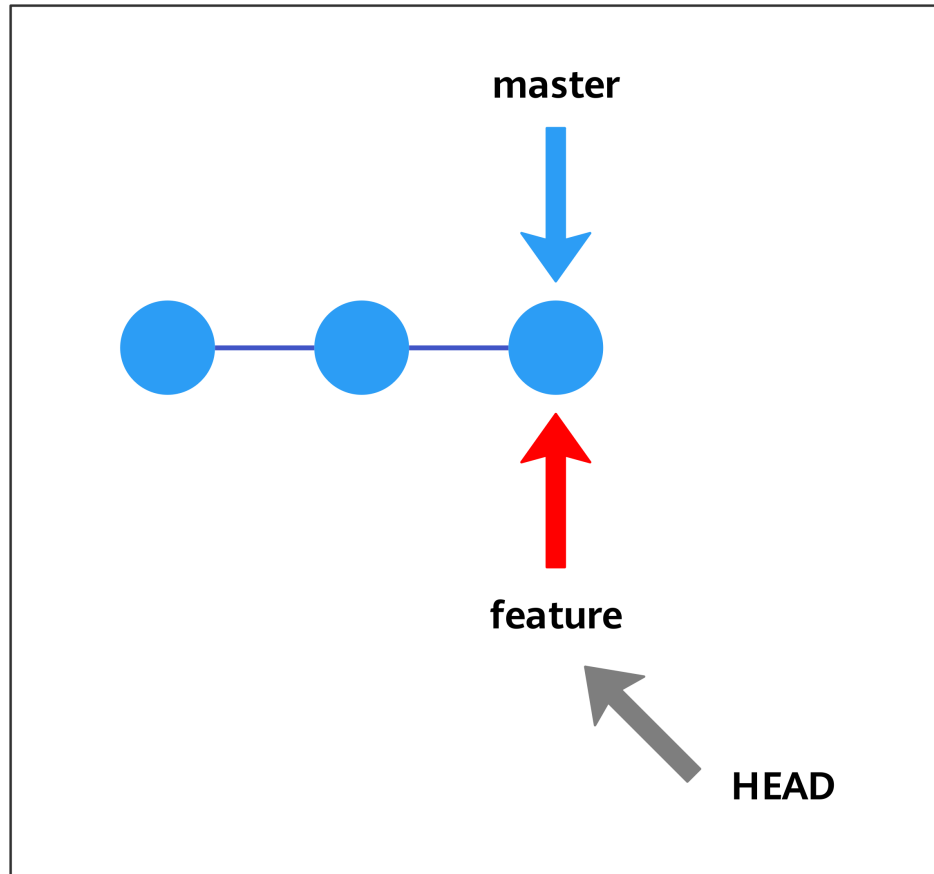


- Check out a branch.

Run the following command:

```
#git checkout branchName
```

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



- Integrate branches.

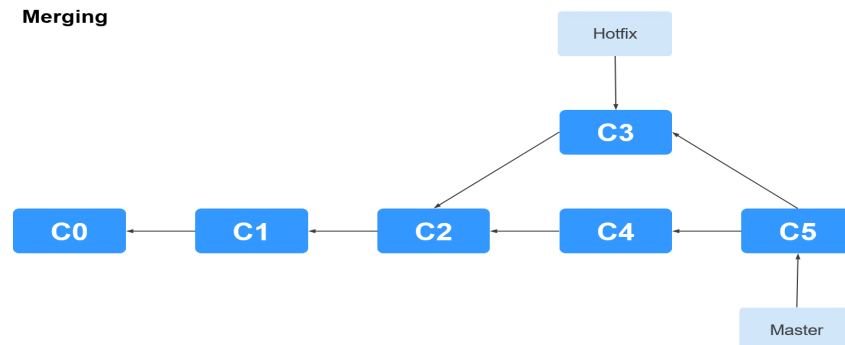
There are two ways to integrate changes from one branch to another: **git merge** and **git rebase**. The following describes the differences between them.

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.

- 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.

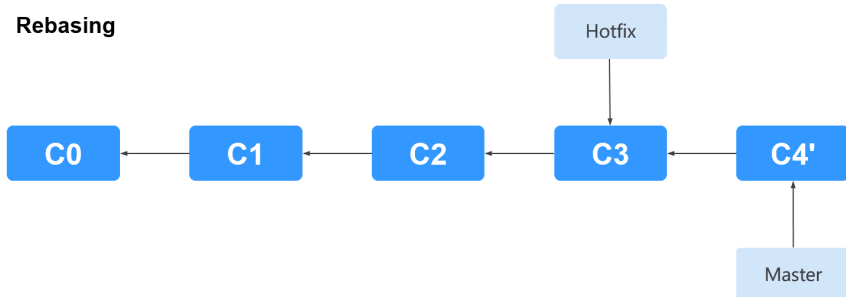
```
#git checkout master
#git merge hotfix
```

Merging



- 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
```



- 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.

2 Migrating the Repository to CodeArts Repo

This practice shows how to migrate your **local** or **cloud repository** to CodeArts Repo.

Application Scenario

With the development of **code migration to the cloud**, self-migration of repositories tends to be normal. CodeArts Repo provides a complete operation guide for you to migrate repositories to CodeArts Repo.

Principle

CodeArts Repo provides the following migration solutions based on the repository storage mode:



- **HTTP online import**

You can directly import your remote repository to CodeArts Repo through HTTP. However, the import duration is affected by network conditions and repository capacity.

NOTE

You are advised to use the **Git client push** mode to migrate cloud repositories with a large capacity.

- **Git client push**

Use the Git client to push code files in the local repository to CodeArts Repo.

- For users who store project files on the local computer, you are advised to initialize the local project files to Git repository and then use the Git client for migration.
- When you **create a repository** for a cloud repository with a large capacity, you are advised to clone or download the cloud repository to the local host, and then use the Git client to migrate the repository.

Prerequisites

- A project is available. If no project is available, **create one** first.
- A repository is available. If no repository is available, **create one** first.
- During repository migration, ensure that the network is stable and smooth.

- The capacity of the repository to be migrated cannot exceed 2 GB. Otherwise, the created repository will be frozen and cannot be used.

HTTP Online Import


- Step 1** Go to the CodeArts homepage and click the target project name to access the project.
- Step 2** Choose **Code > Repo**.
- Step 3** On the CodeArts Repo page, click the  icon next to **New Repository** and select **Import Repository** from the drop-down list box.
- Step 4** On the **Set Basic Information** page, set the following parameters based on the site requirements:

Table 2-1 External repository parameters

Parameter	Mandatory	Description
Source Repository URL	Yes	Enter a URL starting with http:// or https:// and ending with .git.
Source Repository Access	Yes	<ul style="list-style-type: none">• Username and password not required: Select this option if the source repository is open-source (public).• Username and password required: Select this option if the source repository is private, and enter the username and password for cloning HTTPS code.

- Step 5** Select **I have read and agree to the Privacy Statement and CodeArts Service Statement** and click **Next**.
- Step 6** On the **Create Repository** page, set the parameters in the following table.

Table 2-2 Parameter description

Parameter	Mandatory	Remarks
Repository Name	Yes	The name must start with a letter, digit, or underscore (_) and can contain periods (.) and hyphens (-), but cannot end with .git, .atom. The name can contain a maximum of 200 characters.
Description	No	Enter a description for your repository. The description can contain a maximum of 2,000 characters.

Parameter	Mandatory	Remarks
Permissions	No	<ul style="list-style-type: none"> • Make all project developers automatic repository members If you select this option, the project developer is automatically added as a repository member. By default, the project manager is a repository member. • Create a code check task automatically (for free). After the repository is created, you can view the code check task of the repository in the check task list
Visibility	Yes	<p>The options are as follows:</p> <ul style="list-style-type: none"> • Private The repository is visible only to repository members. Repository members can access the repository or commit code. • Public read-only The repository is open and read-only to all visitors. You can select an open-source license as the remarks.
Branch	Yes	You can choose to synchronize the default branch or all branches of the source repository.
Schedule	No	<p>Select Schedule sync into repo.</p> <ul style="list-style-type: none"> • The default branch of the source repository is automatically imported to the default branch of the new repository every day. • The repository becomes a read-only image repository and cannot be written. In addition, only the branches of the third-party repository corresponding to the default branch of the current repository are synchronized.

Step 7 Click **OK**. The repository list page is displayed.

----End

Git Client Push (Git Bash Is Used as an Example)

NOTE

Before pushing, ensure that [SSH key or HTTPS password](#) has been configured in the CodeArts Repo service.

Step 1 Access the target CodeArts Repo.

Step 2 Initialize the local repository to Git repository for association with CodeArts Repo.

Open the Git Bash client in your repository and run the following command:

```
git init
```

The following figure shows that the initialization is successful. The current folder is the local Git repository.

```
Administrator@ecstest-paas-1w MINGW64 ~/Desktop/liu'Code/java
$ git init
Initialized empty Git repository in C:/Users/Administrator/Desktop/liu'Code/java/.git/
```

Step 3 Bind the local repository to CodeArts Repo.

1. Go to the CodeArts Repo and obtain the repository address.
2. Run the remote command to bind the local repository to the cloud repository.
`git remote add <repository_alias> <repository_address>`

Example:

```
git remote add origin git@*****/java-remote.git # Change the address to that of your repository.
```

NOTE

- By default, **origin** is used as the repository alias when you clone a remote repository to the local computer. You can change the alias.
- If the system displays a message indicating that the repository alias already exists, use another one.
- If no command output is displayed, the binding is successful.

Step 4 Pull the master branch of the CodeArts Repo to the local repository.

This step is performed to avoid conflicts.

```
git fetch origin master # Change origin to your repository alias.
```

Step 5 Commit local code files to the master branch.

Run the following commands:

```
git add .
git commit -m "<your_commit_message>"
```

The following figure shows a successful execution.

```
Administrator@ecstest-paas-1wx6 MINGW64 ~/Desktop/liu'Code/java (master)
$ git add .

Administrator@ecstest-paas-1wx6 MINGW64 ~/Desktop/liu'Code/java (master)
$ git commit -m "init commit"
[master (root-commit) 95e7374] init commit
3 files changed, 130 insertions(+)
create mode 100644 file001.txt
create mode 100644 file002.txt
create mode 100644 file003.txt
```

Step 6 Bind the local master branch to the CodeArts Repo master branch.

```
git branch --set-upstream-to=origin/master master # Change origin to your repository alias.
```

The following figure is displayed, indicating that the merged repository has been placed in the working directory and repository.

```
Administrator@ecstest-paas-1 MINGW64 ~/Desktop/liu'Code/java (master)
$ git branch --set-upstream-to=origin/master master
Branch 'master' set up to track remote branch 'master' from 'origin'.
```

Step 7 Merge the files in the CodeArts Repo repository and local repository and store them locally.

```
git pull --rebase origin master # Change origin to your repository alias.
```

The following figure is displayed, indicating that the merged repository has been placed in the working directory and repository.

```
Administrator@ecstest-paas-lwx MINGW64 ~/Desktop/liu'Code/java (master)
$ git pull --rebase origin master
From [redacted]ecstest00001/java-remote
 * branch          master       -> FETCH_HEAD
Successfully rebased and updated refs/heads/master.
```

Step 8 Push the local repository to overwrite the CodeArts Repo repository.

Run the **push** command because the repositories have been bound:

```
git push
```

After the operation is successful, pull the repository to verify that the version of the CodeArts Repo repository is the same as that of the local repository.

```
Administrator@ecstest-paas-lwx MINGW64 ~/Desktop/liu'Code/java (master)
$ git push
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 2 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 427 bytes | 427.00 KiB/s, done.
Total 5 (delta 2), reused 0 (delta 0), pack-reused 0
To [redacted].git
   Oca3cd3..bafb729  master -> master

Administrator@ecstest-paas-lwx MINGW64 ~/Desktop/liu'Code/java (master)
$ git pull
Already up to date.
```

----End

3 Code Review Practice

NOTE

Reviews and **Review Template** support only users of the professional edition or higher.

Background

Code review refers to the process in which developers assign other personnel to read and review the code after designing, compiling, and debugging the code. The purpose of code review is to find standardization and correctness problems about the format, logic, and syntax in the code, thus ensuring the quality of the code. The cost of finding code problems in the code review phase is the lowest. Therefore, strict and careful code review is necessary to improve code quality. To help you review code more efficiently and quickly, you can add a review template in CodeArts Repo as required.

Prerequisites

- A project is available. If no project is available, [create one](#) first.

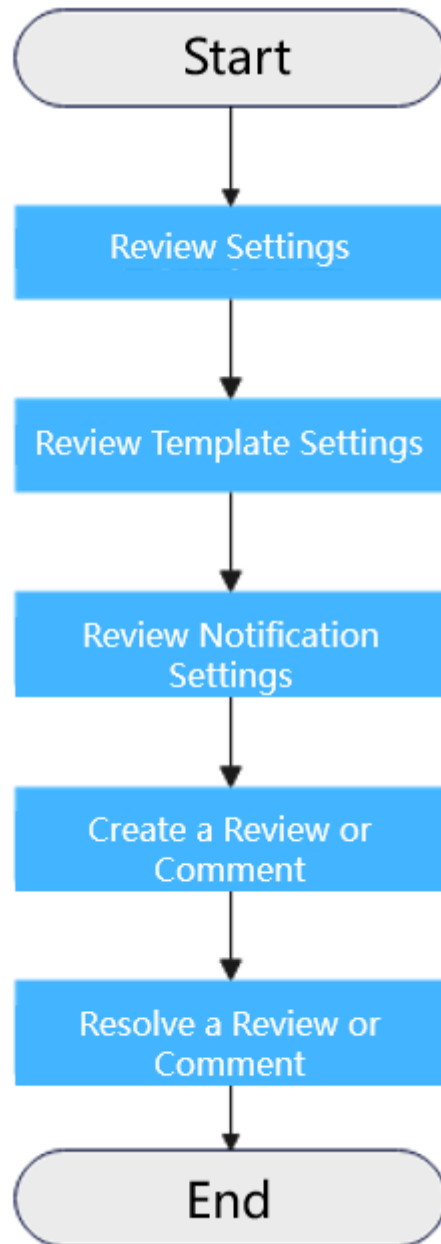
NOTE

- If you purchase a CodeArts service package, you need to [create a project](#) in CodeArts Req.
- If you purchase a CodeArts Repo service package, you need to create a Scrum or an IPD project when creating a repository.
- A repository is available. If no repository is available, [create one](#) first.

Procedure

Before code review, the administrator needs to perform the following management settings:

Figure 3-1 Code review procedure



This document describes how to set a review, review template, and review notification, and create and resolve a review or comment. The procedure is as follows:

- **Review Settings**
- **Review Template Settings**
- **Review Notification Settings**
- **Create a Review or Comment**
- **Resolve a Review or Comment**

Review Settings

The review setting is used to standardize the review and configure the [review template](#). The setting takes effect only for the configured repository. Only the repository administrator and owner can view the page and have the setting permission.

Step 1 Go to the CodeArts homepage and click the target project name to access the project.

Step 2 Choose **Code > Repo**.

Step 3 Go to the repository details page and choose **Settings > Policy Settings > Reviews**. The **Reviews** page is displayed.

Step 4 Select **Enable comment types and modules** as required.

Step 5 Configure review types.

- Enable preset comment types

If you select **Enable preset comment types**, you can directly use the preset review comment categories.

- Customize category

You can customize the review comment category. Enter a type name, for example, **code specification problem**, and press **Enter** to save the settings.

NOTE

The name can contain a maximum of 200 characters. A maximum of 20 names can be created.

Step 6 Enter a category name in the text box under **Comment Modules**.

NOTE

The name can contain a maximum of 200 characters. A maximum of 20 names can be created.

Step 7 Select **Mandatory fields to Verify for Comment Creation/Editing** as required.

Step 8 Click **Submits**.

----End

Review Template Settings

To configure comment templates, you can choose **Settings > Template Management > MR Comment Templates** on the repository details page. You can create, edit, and delete a review template, and customize template information such as **Severity**, **Assigned to**, **Comment category**, **Comment module**, and **Description**. When adding a review, you can select a review template. The template content will be automatically applied to the merge request or the code file to be reviewed. The settings take effect only for the repository configured. Only the repository administrator and owner can view the page and have the setting permission.

You can create a review template by referring to [Table 3-1](#).

Table 3-1 Parameter description

Parameter	Description
Template name	Mandatory. Name of the template to be created. For example, the code review template.
Set as default	Optional. If this parameter is selected, this template is used by default during reviewing.
Severity	Optional. Classified into the following types based on problem severity: Fatal , Major , Minor , and Suggestion . For example, set the value of Severity to Minor .
Assigned to	Optional. <ul style="list-style-type: none">• If this parameter is set to empty:<ul style="list-style-type: none">– When a review is added to an MR, the review is assigned to the MR creator by default.– When a review is added to a file or commit, the review is not assigned by default.• If this parameter is set to the MR Creator or Committer.<ul style="list-style-type: none">– When a review is added to an MR, the review is assigned to the MR creator by default.– When a review is added to a file or commit, the review is assigned to the committer by default.• Assign to a specific person.<ul style="list-style-type: none">– When a review is added to an MR, the review is assigned to a specific person by default.– When a review is added to a file or commit, the review is assigned to a specific person by default. For example, assign the review to the MR creator.
Comment category	This parameter is optional and is disabled by default. You need to select Enable comment types and modules first and configure the review types. For details, see Review Settings .
Comment Modules	This parameter is optional and is disabled by default. You need to select Enable review comment categories and modules first and configure the review modules. For details, see Review Settings .

Parameter	Description
Description	Optional. Enter the description of the template. The description can be previewed. For example, the code format is incorrect.

Set Review Comment Notification

To set notifications, choose **Settings > General Settings > Notifications** on the repository details page.

Figure 3-2 Notification settings page

Merge Request

Open	<input type="checkbox"/> Scorer	<input type="checkbox"/> Approver	<input type="checkbox"/> Reviewer	<input type="checkbox"/> Merger
Update	<input type="checkbox"/> Scorer	<input type="checkbox"/> Approver	<input type="checkbox"/> Reviewer	
Merge	<input checked="" type="checkbox"/> MR Creator	<input type="checkbox"/> Merger		
Review	<input checked="" type="checkbox"/> MR Creator			
Approve	<input checked="" type="checkbox"/> MR Creator			
Comment	<input checked="" type="checkbox"/> MR Creator			
Resolve Comment	<input checked="" type="checkbox"/> MR Creator			

- **Comment:** You can manually set to send an email notification to the MR creator.
- **Resolve Comment:** You can manually set to send an email notification to the MR creator.

Create a review or comment


You can add a review for a file on the **Files** and **Commits** submenus of the **Code** tab page, or on the **Files Changed** submenu of the **Merge Requests** tab page.

You can add comments for a merge request on **Comments** of the **Merge Requests** details page, or add comments for a commit on **Commits** submenu of the **Code** tab page.

The reviews added on the **Files** and **Commits** submenus of the **Code** tab page can be viewed on the **Reviews for commit** of the **Reviews** tab page.

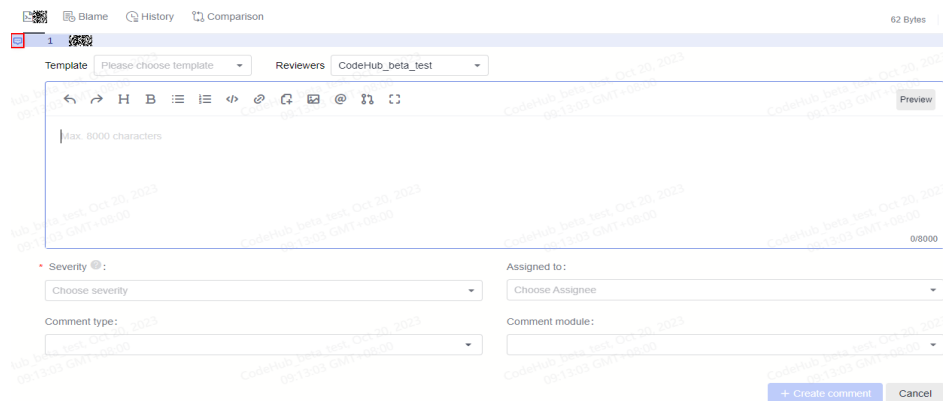
The reviews added on the the **Files Changed** submenu of the **Merge Requests** tab page and the comments added on **Comments** of the **Merge Requests** details page can be viewed on the **Reviews for MR** of the **Reviews** tab page.

- Click **Code > Files** to create a review.

Click the target file on the **Files** tab page, click the  icon in the code line, enter a review in the text box, and set a value for **Severity** and **Assigned to**.

For example, set **Severity** to **Minor** and **Assigned to** to **MR Creator**, and select values for **Comment type** and **Comment module** from the drop-down list box, and click **OK**.

Figure 3-3 Create a review



- Click **Code > Commits** to create a review.


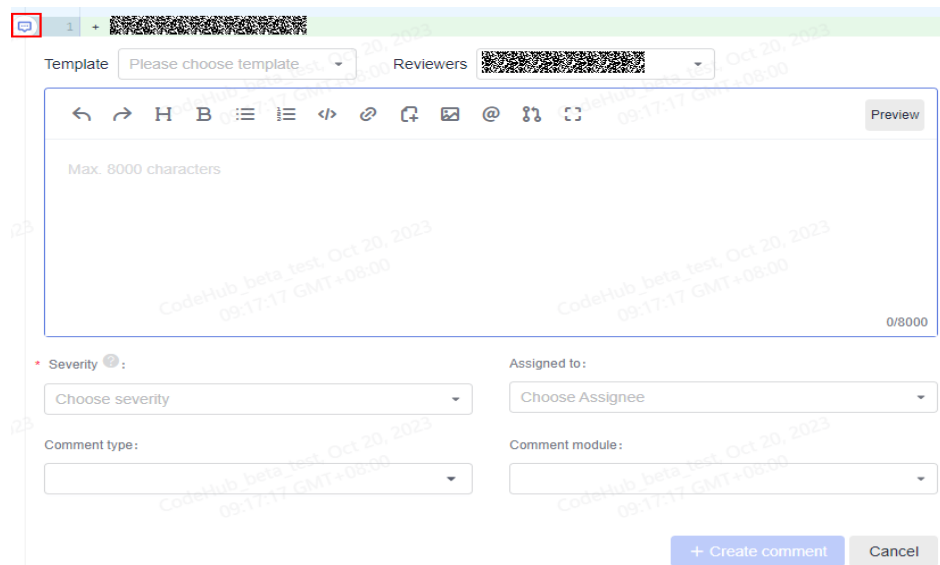
Click the target file on the **Commits** tab page, click the  icon in the code line, enter a review in the text box, and set a value for **Severity** and **Assigned to**. For example, set **Severity** to **Minor** and **Assigned to** to **MR Creator**, and select values for **Comment type** and **Comment module** from the drop-down list box, and click **OK**.

Figure 3-4 Create a review



- Click **Merge Requests > Files Changed** to create a review.


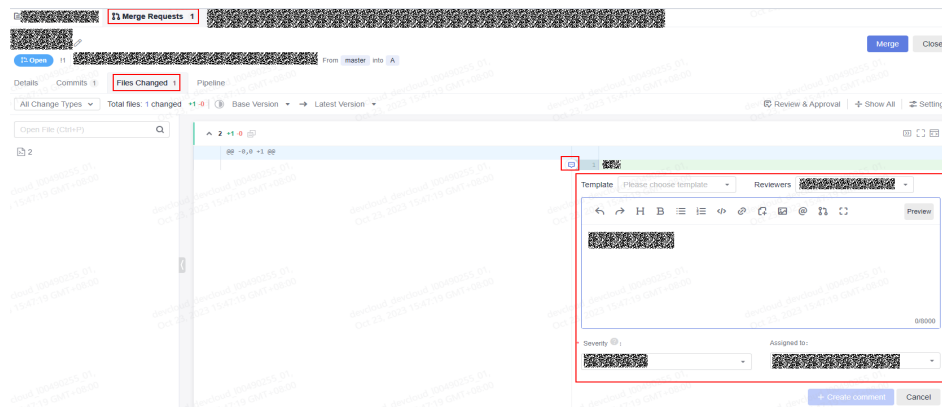
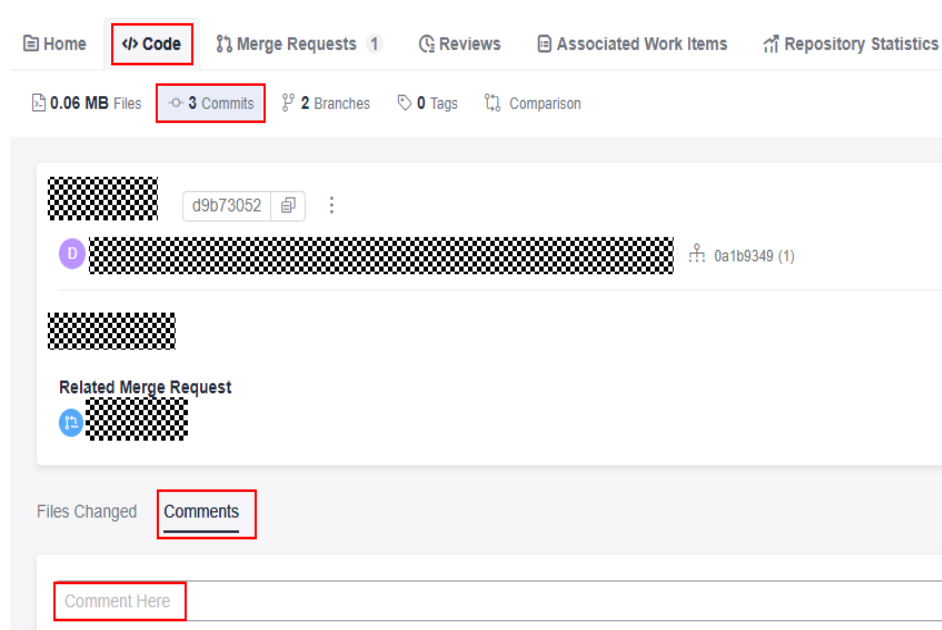
Go to the **Files Changed** submenu, click the  icon in the code line, enter a review in the text box, and set a value for **Severity** and **Assigned to**. For example, set **Severity** to **Minor** and **Assigned to** to **MR Creator**, and select values for **Comment type** and **Comment module** from the drop-down list box, and click **OK**.

Figure 3-5 Create a review



- Click **Merge Requests > details** to create a comment. Click target MR on the **Merge Requests** page. The merge request details page is displayed. Click **Comments**, enter a comment, and click **OK**.
- On the **Commits** page, click a commit to switch to the **Comments** page. Then you can create a comment.

Figure 3-6 Creating a comment



Resolve a review or comment

On the **Reviews** tab page, you can view **Reviews for MR** and **Reviews for commit**.

- After modifying the code file based on the reviews on the **Reviews for commit**, contact the committer for review. After the code file is approved, change the status to  **Resolved**.
- After modifying the code file in the merge request based on the reviews or comments on the **Reviews for MR**, contact the committer to review the code file. After the code file is approved, change the status to  **Resolved**.

