Repo

Getting Started

Issue 06

Date 2023-07-25

HUAWEI TECHNOLOGIES CO., LTD.

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 Getting Started

If you are new to Git, go to **2 Getting Started with Git-Based CodeArts Repo** to learn how Git and CodeArts Repo work.

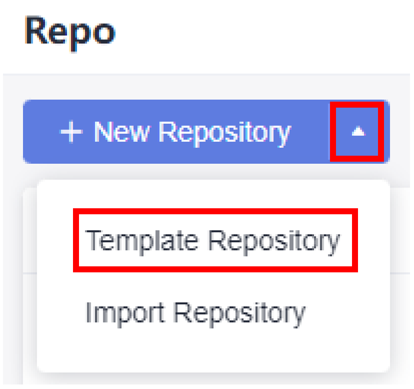
If you have used Git for version management, we will show you a quick overview of CodeArts Repo functions.

Creating a Repository

The following procedure describes how to create a repository using a template.

Step 1 Go to a target project and choose **Repo** from the navigation pane.

Step 2 Click ▾ next to **New Repository** and select **Template Repository** from the drop-down list.



Step 3 On the **Select Template** page, enter a keyword for fuzzy search and select a template as required.

Step 4 Click **Next**. On the **Basic Information** page, enter basic repository information.

Table 1-1 Parameters for creating a repository using a template

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, or period (.). The name can contain a maximum of 200 characters.
Project	Yes	• A repository must be associated with a project. • If there is no project under your account, you can click **Create Project** in the **Project** drop-down list to create a Scrum or an IPD project. (For a basic project, only CodeArts Repo and CodeArts Check can be used. Other services are neither enabled nor displayed. You can change a project to a basic project on the project settings page.) **NOTE** If you create a repository in a project, the project is selected for **Project** by default, and the **Project** parameter is hidden on the repository creation page.
Description	No	Enter the description of your repository.
Permissions	No	• **Make all project developers automatic repository members.** A project manager is automatically set as the repository administrator, and A developer is set as a common repository member. When members of the two roles are added to the project, they are added to the repository member list by automatic synchronization. You can view the list.
Visibility	Yes	The options are as follows: • **Private** Only repository members can access and commit code. • **Public** Read-only for all visitors and hidden from repo list and search result. You can select an open-source license as the remarks.

Step 5 Click **OK**. The repository is created and the repository list is displayed.

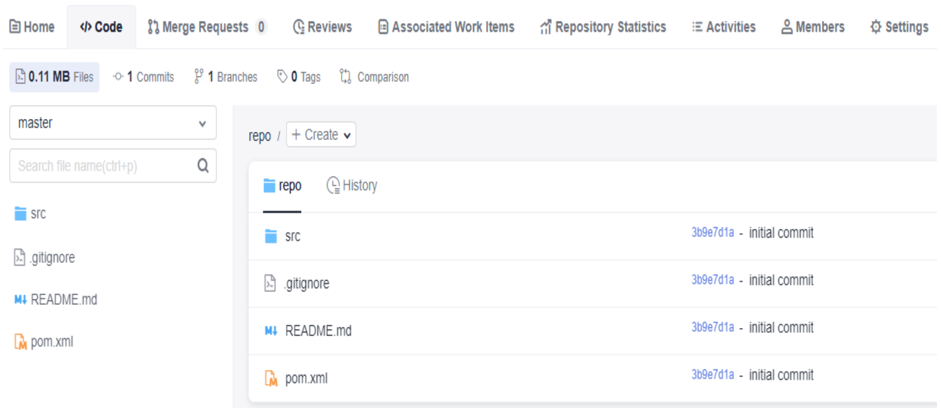
----End

We are done with the repository. Let's create a branch then.

Creating a Branch

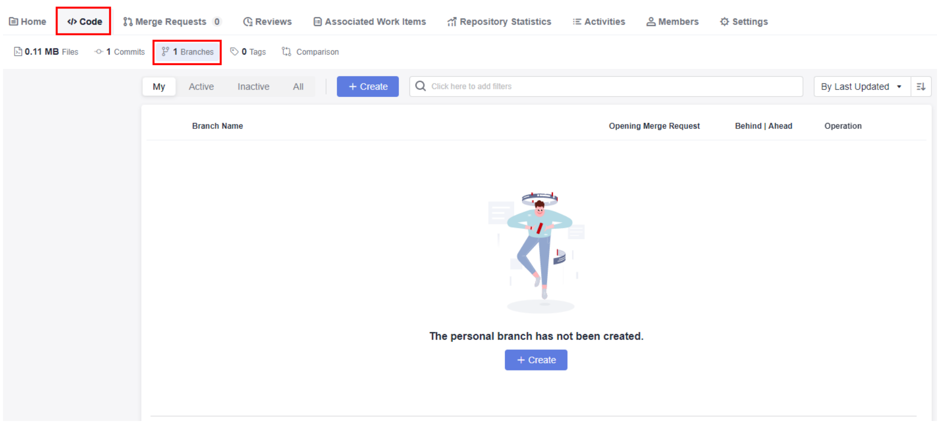
Branch is the most commonly used method in version management. Branches isolate tasks in a project to prevent them from affecting each other, and can be **merged** for version release.

Step 1 Click a repository to go to the details page.



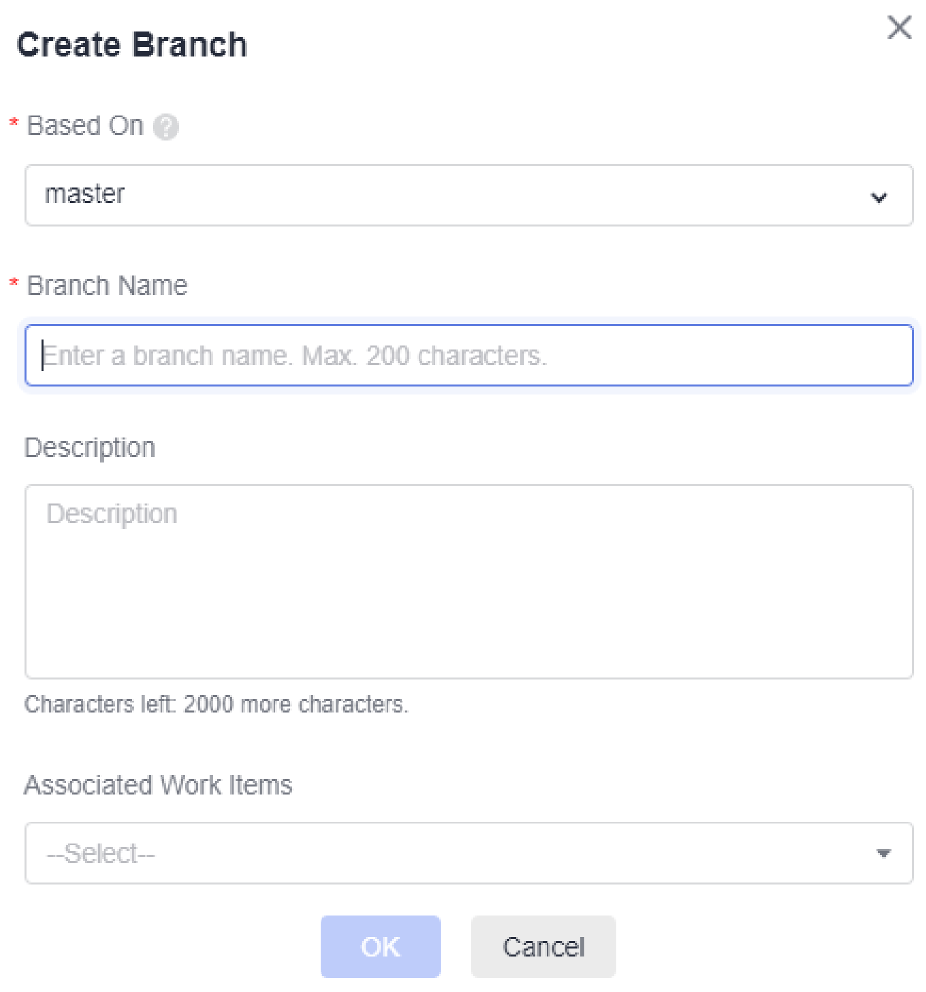
src	3b9e7d1a - initial commit
.gitignore	3b9e7d1a - initial commit
README.md	3b9e7d1a - initial commit
pom.xml	3b9e7d1a - initial commit

Step 2 Switch to the **Branches** tab page under the **Code** tab page. Branches in the remote repository are displayed.



The personal branch has not been created.

Step 3 Click **Create**. In the displayed dialog box, select a version (branch or tag) based on which you want to create a branch and enter the branch name. You can associate the branch with an existing work item.


Create Branch

* Based On

master

* Branch Name

Enter a branch name. Max. 200 characters.

Description

Description

Characters left: 2000 more characters.

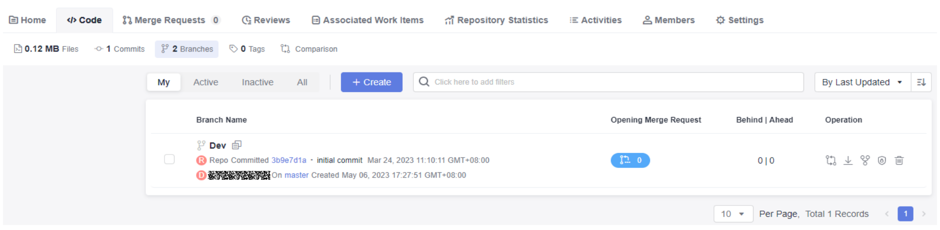
Associated Work Items

--Select--

Table 1-2 Parameter description

Project	Mandatory	Remarks
Based On	Yes	Create a branch based on an existing branch or tag.
Branch Name	Yes	Name of the new branch
Description	No	Description of the new branch
Work Items to Associate	No	Specify the work items to associate with this new branch.

Step 4 Click **OK**. The branch list is displayed.



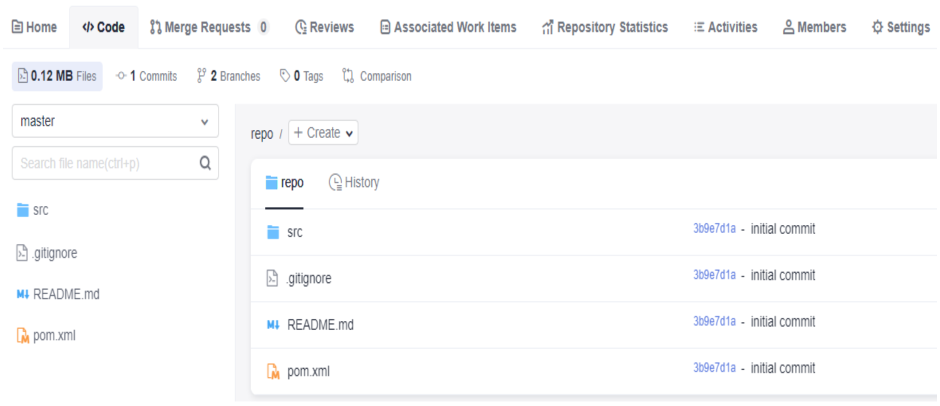
Branch Name	Opening Merge Request	Behind | Ahead	Operation
Dev Repo Committed 3b9e7d1a · initial commit Mar 24, 2023 11:10:11 GMT+08:00 [redacted] On master Created May 06, 2023 17:27:51 GMT+08:00	0	0 | 0	

----End

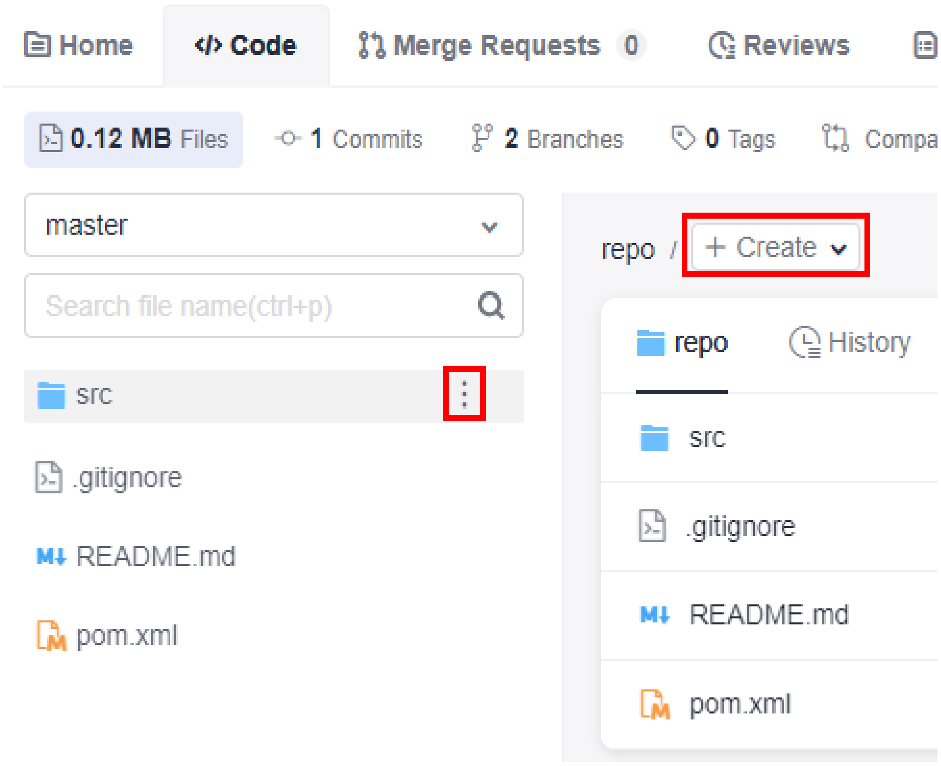
We are done with the branch. Creating a file is our next step.

Creating a File

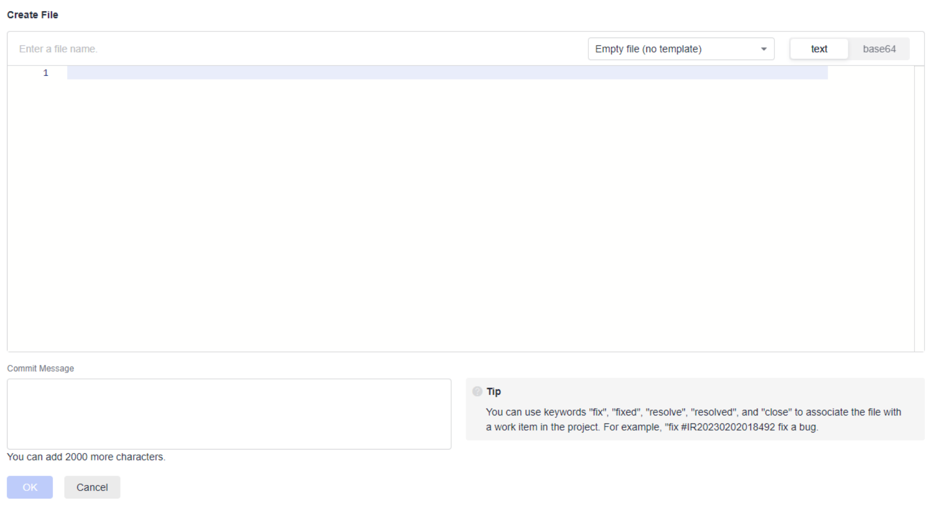
Step 1 Click a repository to go to the details page.



Step 2 Place the cursor on the folder name and click ⋮ or + Create. Then click **Create File**.



Step 3 Set the following parameters as required:



You can use keywords "fix", "fixed", "resolve", "resolved", and "close" to associate the file with a work item in the project. For example, "fix #IR20230202018492 fix a bug.

You can add 2000 more characters.

Table 1-3 Parameter description

Project	Mandatory	Remarks
File name	Yes	Name of the new file
Empty file (no template)	Yes	You can select multiple template types. By default, no template is used.
text/base64	Yes	The value can be **text** (default) or **base64**.
File content	No	Content of the new file
Commit Message	Yes	It automatically synchronizes the file name and can be customized. You can associate work items here.

Step 4 Click **OK**. The file is created and the file list is displayed.

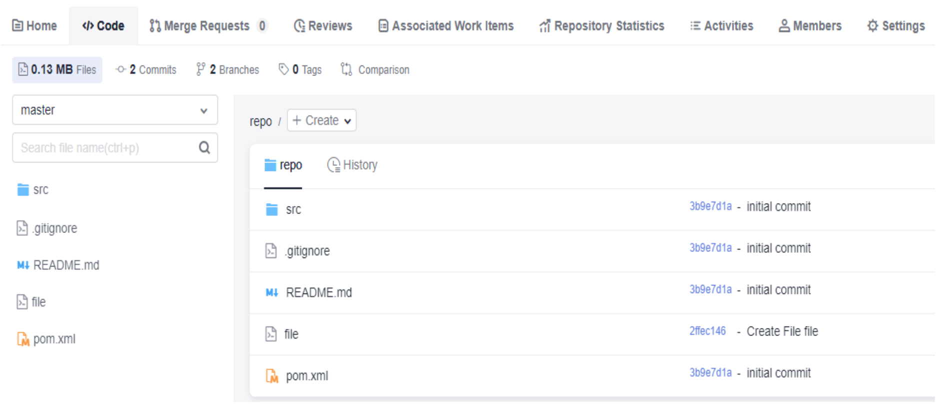
----End

You have created a file. Next, you can create a request for merging the two branches.

Creating a Merge Request

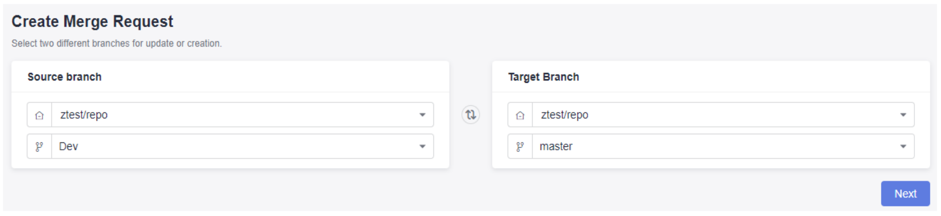
CodeArts Repo supports development of multiple branches and establishes configurable review rules for branch merging. When a developer initiates an MR, some repository members can be selected to participate in code reviews to ensure the correctness of the merged code.

Step 1 Click a repository to go to the details page.



src	3b9e7d1a	initial commit
.gitignore	3b9e7d1a	initial commit
README.md	3b9e7d1a	initial commit
file	2ffec146	Create File file
pom.xml	3b9e7d1a	initial commit

Step 2 Switch to the **Merge Requests** tab page, click **New**, and select the branches to be merged.


Create Merge Request

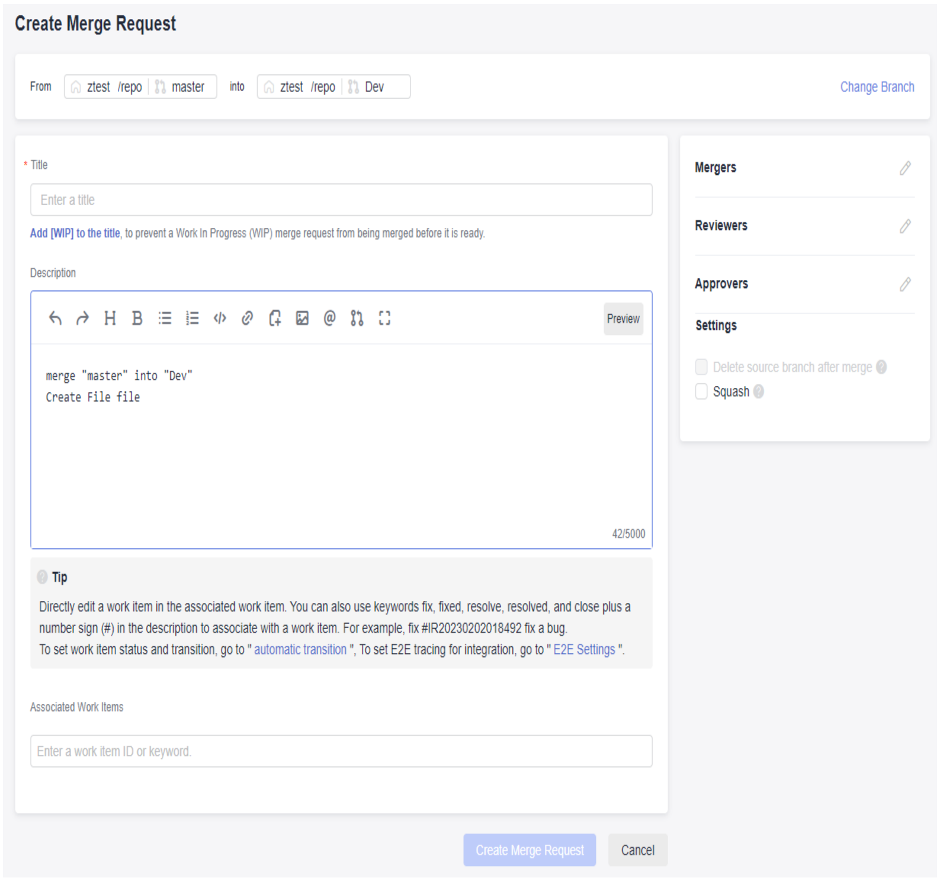
Select two different branches for update or creation.

Source branch: ztest/repo, Dev

Target Branch: ztest/repo, master

Step 3 Click **Next**. The system checks whether the two branches are different.

- If there is no difference between the two, the system displays a message and the request cannot be created.
- If the branches are different, the following **Create Merge Request** page is displayed.



Create Merge Request

From ztest /repo master into ztest /repo Dev Change Branch

* Title

Enter a title

Add [WIP] to the title, to prevent a Work In Progress (WIP) merge request from being merged before it is ready.

Description

```
merge "master" into "Dev"
Create File file
```

42/5000

Tip

Directly edit a work item in the associated work item. You can also use keywords fix, fixed, resolve, resolved, and close plus a number sign (#) in the description to associate with a work item. For example, fix #IR20230202018492 fix a bug.
To set work item status and transition, go to " automatic transition ", To set E2E tracing for integration, go to " E2E Settings ".

Associated Work Items

Enter a work item ID or keyword.

Mergers

Reviewers

Approvers

Settings

- [ ] Delete source branch after merge
- [ ] Squash

Create Merge Request Cancel

The lower part of the **Create Merge Request** page displays differences of the two branches and commit records of the source branch.

Step 4 Set the parameters according to the following table.

Table 1-4 Parameter description

Parameter	Description
Change Branch	Click to return to the previous step and change the branch to be merged.
Title	Enter the MR title.
Description	A default description is generated based on the merge and commit messages of the source branch. You can modify the description as required.
Associated Work Items	You can associate a merge action with a work item to automatically change the status of the work item.
Mergers	Mergers have permissions to merge branches (by clicking the merge button) when all approvers approve MRs and all discussed issues are solved (or you can set the rule to allow merge with issues unsolved). They can also close the MR.
Reviewers	Specified to participate in the merge branch review and can raise questions to the initiator.

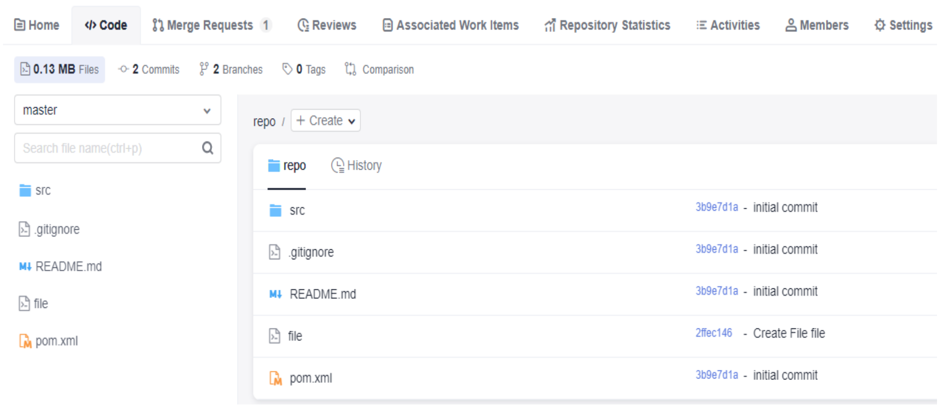
Parameter	Description
Approvers	Appointed to participate in the merge branch review. You can provide review comments (approved or rejected) or raise questions to the initiator.
Delete source branch after merge	You can choose whether to delete the source branch after merge. The preset status in the MR settings is initially used.
Squash	Squash is to merge all change commit information of an MR into one and keep a clean history. When you focus only on the current commit progress but not the commit information, you can use squash merge. Enabling Squash keeps the history of the basic branch clean, with meaningful commit messages, and can be restored more easily if necessary.

Step 5 Click **Create Merge Request** to submit the MR. The **Details** page is displayed.

----End

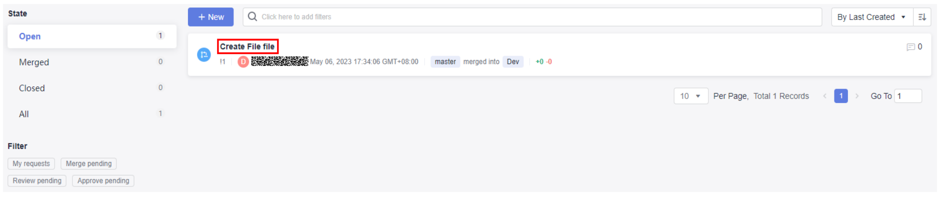
Merging a Request

Step 1 Click a repository to go to the details page.



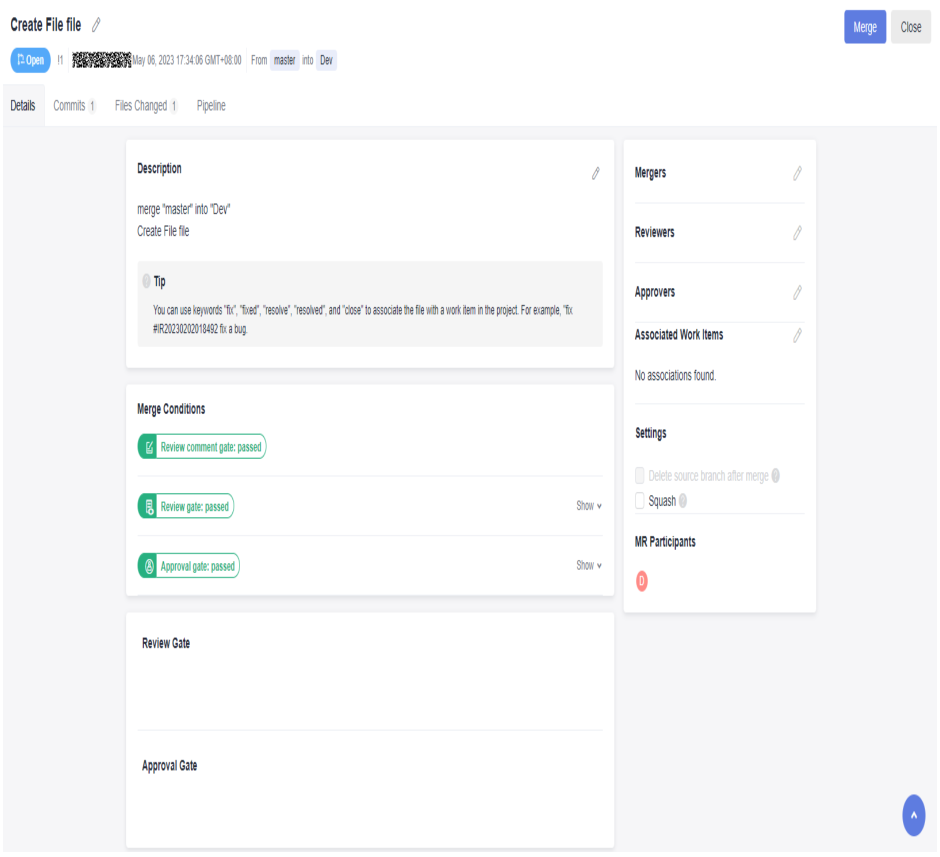
src	3b9e7d1a	initial commit
.gitignore	3b9e7d1a	initial commit
README.md	3b9e7d1a	initial commit
file	2ffec146	Create File file
pom.xml	3b9e7d1a	initial commit

Step 2 Switch to the **Merge Requests** tab page and click the name of the target merge request. The merge request details page is displayed.



Create File file
!1 D [redacted] May 06, 2023 17:34:06 GMT+08:00 master merged into Dev +0 -0 0

Step 3 Ask reviewers and approvers to do their jobs.



Create File file

Open !1 [redacted] May 06, 2023 17:34:06 GMT+08:00 From master into Dev

Merge Close

Details Commits 1 Files Changed 1 Pipeline

Description

merge "master" into "Dev"
Create File file

Tip
You can use keywords "fix", "fixed", "resolve", "resolved", and "close" to associate the file with a work item in the project. For example, "fix #IR20230202018492 fix a bug.

Merge Conditions

Review comment gate: passed

Review gate: passed Show

Approval gate: passed Show

Review Gate

Approval Gate

Mergers

Reviewers

Approvers

Associated Work Items

No associations found.

Settings

Delete source branch after merge

Squash

MR Participants

Table 1-5 Merge conditions

Merge Condition	Description
Code merge conflicts	When a merge conflict occurs between the source and target branch code, resolve the conflict before performing the next operation. For details about how to resolve a code conflict, see **Resolving Code Conflicts in an MR**.
Review comment gate	After an initiator resolves the review comments of all reviewers or approvers, the gate is passed. **NOTE** The gate function takes effect only after you choose **Settings > Policy Settings > Merge Requests** and select **Merge after all reviews are resolved**.
Pipeline gate	When the latest commit or pre-merged commit starts and successfully executes the pipeline, the gate is passed. For details see **Configuring a Pipeline**.
E2E ticket number not associated	After an MR is associated with a work item, the gate is passed. **NOTE** To enable the gate function, choose **Settings > Policy Settings > Merge Requests** and select **Must be associated with CodeArts Req**.
Review gate	When the number of reviewers reaches the minimum number, the gate is passed.

Merge Condition	**Description**
Approval gate	When the number of approvers reaches the minimum number, the gate is passed.

Step 4 Ask the merger to merge the request after an initiator meets the preceding conditions. Otherwise, the merger can close the request.

----End

We are now done with this tutorial. You can explore more functions.

2 Getting Started with Git-Based CodeArts Repo

CodeArts Repo is a Git-based online code hosting service for software developers. It is a cloud code repository with functions such as security management, member and permission management, branch protection and merging, online editing, and statistical analysis. The service aims to address issues such as cross-distance collaboration, multi-branch concurrent development, code version management, and security. This topic aims to help you quickly learn how to use Git and CodeArts Repo. If you are already familiar with Git, you can skip this topic.

In this topic, you will lean how to create a cloud repository, clone the cloud repository to the local Git environment using SSH, edit the code locally, and push the changes to the cloud repository. The Java War demo provided by CodeArts Repo will be used here.

Prerequisites

- A project is available. If no project is available, **create a project**.

  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.

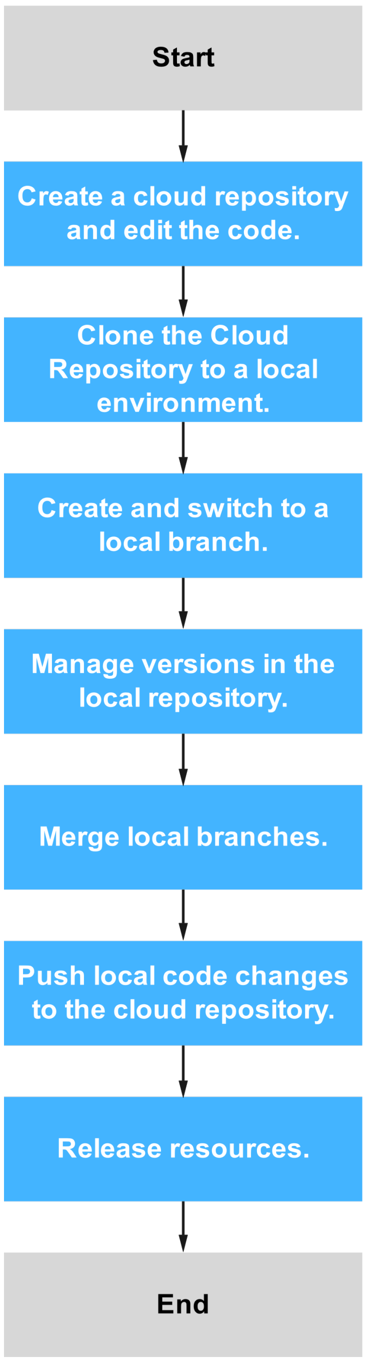
- **Download and install the Git client.**
- **Setting SSH key or HTTPS password for CodeArts Repo repository.**
- Your network can access CodeArts Repo.

  Run the following command on the Git client to check the network connectivity:

  ```
  ssh -vT git@XXXXXXXX.com
  ```

  If the command output contains **connect to host XXXXXXXX.com port 22: Connection timed out**, your network is restricted from accessing CodeArts Repo. Contact your network administrator.

Procedure


Start
Create a cloud repository and edit the code.
Clone the Cloud Repository to a local environment.
Create and switch to a local branch.
Manage versions in the local repository.
Merge local branches.
Push local code changes to the cloud repository.
Release resources.
End

The following operations or knowledge is involved:

1. **Creating a Cloud Repository and Editing the Code**.
2. **Cloning the Cloud Repository to a Local Environment**.
3. **Creating and Switching to a Local Branch**.
4. **Managing Versions in the Local Repository**.
5. **Merging Local Branches**.
6. **Pushing Local Changes to the Cloud Repository**.

7. **Releasing Resources**.

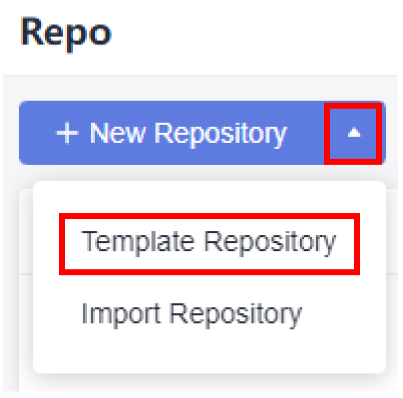
Creating a Cloud Repository and Editing the Code

If you have a cloud repository available, skip this section.

You will use the **Java War Demo** to create a repository. The **Java War Demo** is a template of the **Hello World** applet.

Step 1 Go to a target project and choose **Repo** from the navigation pane.

Step 2 Click next to **New Repository** and select **Template Repository** from the drop-down list.



Step 3 In the **Select Template** step, search for **Java War Demo** in the search box, select the template in the result, and click **Next**.

NOTE

When you search for a template, the region displayed in the filter control indicates the region where the template is stored. If you use the template to create a repository, the repository will be located in the same region as that of the project to which the repository belongs, not as that of the template.

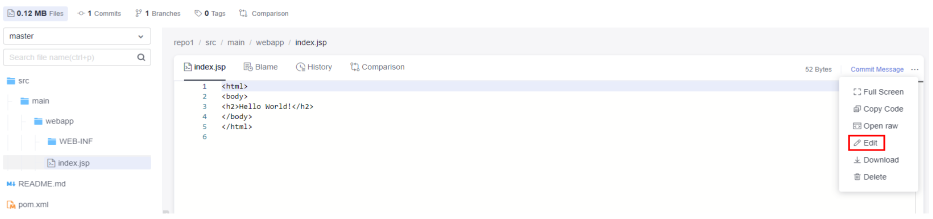
Step 4 In the **Basic Information** page, specify the repository name and other details, and click **OK**.

The created repository is displayed on the CodeArts Repo homepage. You can click the repository name to view the files in the repository.

Step 5 Edit the code.

CodeArts Repo allows you to edit the code in the cloud.

1. On the repository list page, find the repository created in the previous step, and click the repository name.
2. Choose the **Code** tab. In the navigation pane on the left, open the **src/main/webapp/index.jsp** file, and click on the right. Edit the text **Hello World**, enter a commit message, and click **OK**.



```
<html>
<body>
<h2>Hello World!</h2>
</body>
</html>
```

----End

Cloning the Cloud Repository to a Local Environment

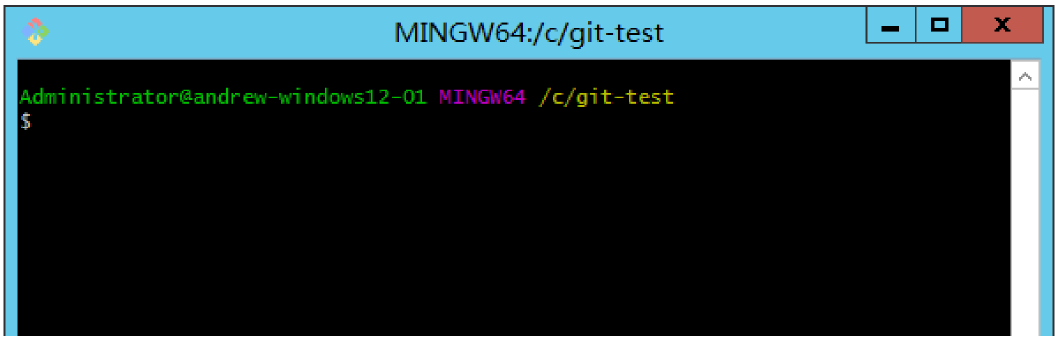
You will clone the cloud repository to your local machine. The following uses the Git Bash client as an example.

Step 1 Obtain the repository address.

Go to the repository details page and click **Clone/Download** to obtain the SSH address.

Step 2 Open the Git Bash client.

Create a directory on your local machine to store the code repository. In this example, the directory is named **git-test**. Go to the directory, right-click on an empty area, and open the Git Bash client.



```
Administrator@andrew-windows12-01 MINGW64 /c/git-test
$
```

NOTE

The repository is automatically initialized during clone. You do not need to run the **init** command.

Step 3 Run the following command to clone the cloud repository:

```
git clone Repository-address
```

Replace *Repository-address* with the SSH address obtained in **Step 1**.

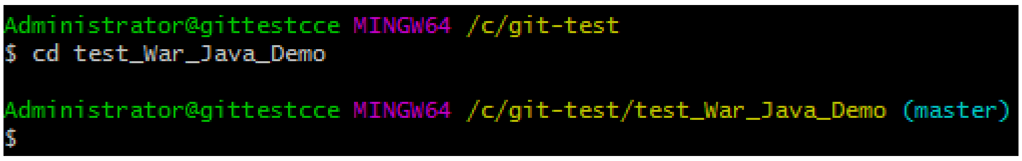
When you communicate with the cloud repository for the first time, you will be asked whether to save the fingerprint. Enter **yes** for the communication to proceed.

After the command is executed, go to the **git-test** directory. If there is a directory with the same name as the cloud repository and the directory contains a hidden **.git** directory, the clone is successful.

Step 4 Run the following command to go to the repository directory:

```
cd Repository-name
```

You will be taken to the **master** branch by default.



```
Administrator@gittestcce MINGW64 /c/git-test
$ cd test_War_Java_Demo

Administrator@gittestcce MINGW64 /c/git-test/test_War_Java_Demo (master)
$
```

----End

Creating and Switching to a Local Branch

The **master** branch is the default main branch after the repository is created. It is recommended that you create a branch off the **master** branch, develop and release code or fix bugs on the derived branch, and commit the changes to the **master** branch instead of making changes on the **master** branch directly. This ensures that the code on the **master** branch is always ready and deployable. In this section, you will create a branch named **dev** in your local machine and switch to the branch.

Step 1 Create a branch.

Open Git Bash, go to the repository directory, and run the following command to create a branch named **dev** in your local machine:

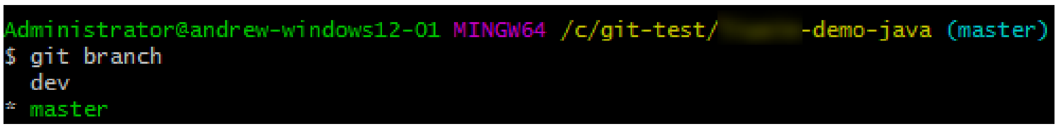
```
git branch dev
```

If no command output is displayed, the branch is created.

Step 2 (Optional) View the branches.

Run the following command to check the local repository branches:

```
git branch
```



```
Administrator@andrew-windows12-01 MINGW64 /c/git-test/[redacted]-demo-java (master)
$ git branch
  dev
* master
```

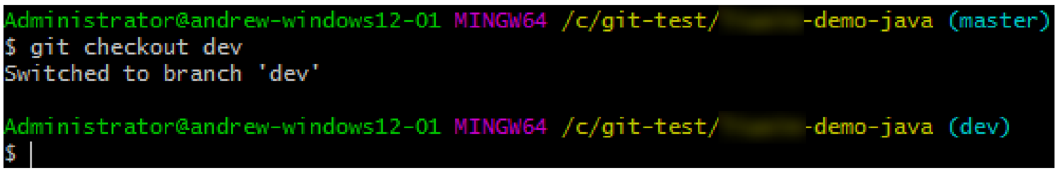
The command output indicates that there are two branches, **master** and **dev**, and you are now on the **master** branch. The code on the **master** branch and **dev** branch is now the same.

Step 3 Check out a branch.

Run the following command to switch to the **dev** branch:

```
git checkout dev
```

If **(dev)** is displayed next to the current path, the checkout is successful. All changes you make in the local repository will be saved to the current branch, namely the **dev** branch.



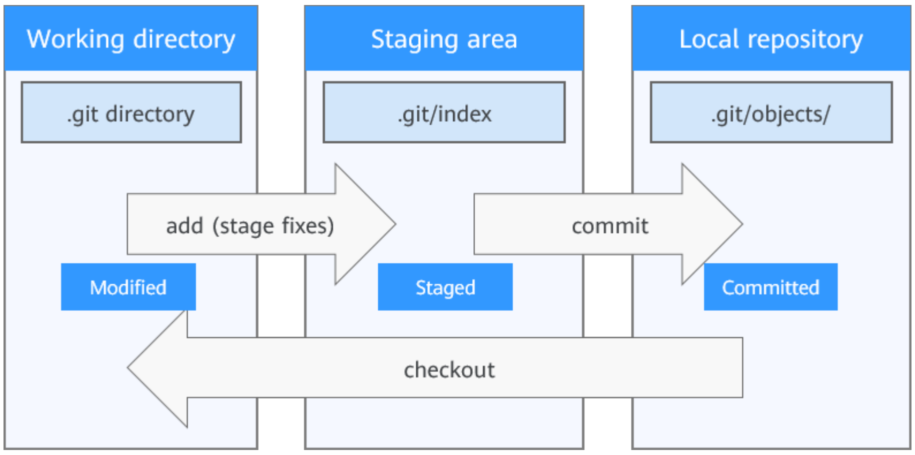
```
Administrator@andrew-windows12-01 MINGW64 /c/git-test/[redacted]-demo-java (master)
$ git checkout dev
Switched to branch 'dev'

Administrator@andrew-windows12-01 MINGW64 /c/git-test/[redacted]-demo-java (dev)
$ |
```

----End

Managing Versions in the Local Repository

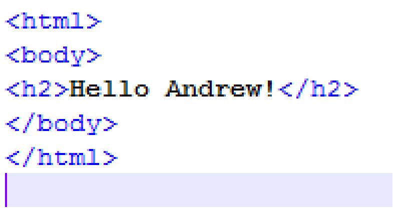
In this section, you will edit the **\src\main\webapp\index.jsp** file in the local repository and run the **add** and **commit** commands to commit the changes to the local repository.

Data in a Git local repository can be in one of the three statuses: modified, staged, and committed. After you edit a file in the repository, the file is in the modified state. You can run the **add** command to add the changes to the staging area, and the file becomes staged. To commit the staged file to the local repository, run the **commit** command. A version is generated for each commit, making it possible for you to switch between versions or roll back versions. The following figure shows the basic working process of a Git local repository. There can be multiple branches in one version. Each branch is a unique set of code and can be seen as a subversion.



Step 1 Edit the code on the **dev** branch.

Ensure that operations in **Cloning the Cloud Repository to a Local Environment** are done. Open the local repository directory, find the **\src\main\webapp \index.jsp** file, and use any text editor to open the file. You can see the content edited in **Creating a Cloud Repository and Editing the Code** because by now, the code on the two local repository branches (**dev** and **master**) is the same as that in the cloud repository.



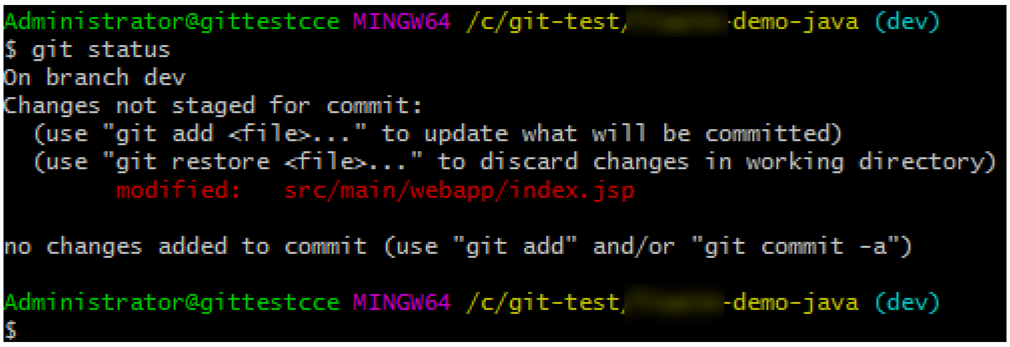
```
<html>
<body>
<h2>Hello Andrew!</h2>
</body>
</html>
```

Change the content to **Hello git!!!**, and save and close the file. The changes are saved to the **dev** branch which is the current branch after the checkout in **Creating and Switching to a Local Branch**.

Step 2 (Optional) View the changes on the current branch.

Run the **status** command to check the status of the files on the current branch.

```
git status
```



```
Administrator@gittestcce MINGW64 /c/git-test/[redacted]-demo-java (dev)
$ git status
On branch dev
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   src/main/webapp/index.jsp

no changes added to commit (use "git add" and/or "git commit -a")

Administrator@gittestcce MINGW64 /c/git-test/[redacted]-demo-java (dev)
$
```

The command output indicates that your changes have not been added to the staging area and have not been committed to the local repository.

Step 3 Stage the changes.

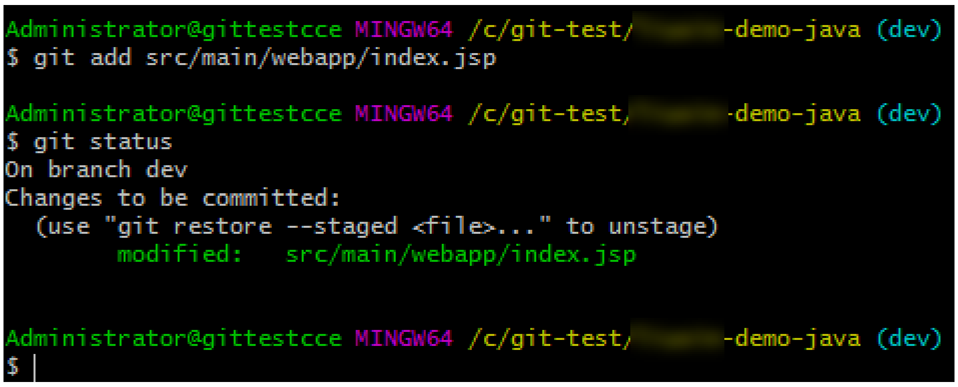
Run the **add** command to add the changes to the staging area:

```
git add .
```

or

```
git add src/main/webapp/index.jsp
```

The **git add.** command will stage all changes. You can also stage a specific file by specifying the file path as shown in the second command. If no command output is displayed, the execution is successful. Run the **status** command again, and you can see the changes have been staged and will go into your next commit.



```
Administrator@gittestcce MINGW64 /c/git-test/[redacted]-demo-java (dev)
$ git add src/main/webapp/index.jsp

Administrator@gittestcce MINGW64 /c/git-test/[redacted]-demo-java (dev)
$ git status
On branch dev
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   src/main/webapp/index.jsp


Administrator@gittestcce MINGW64 /c/git-test/[redacted]-demo-java (dev)
$ |
```

Step 4 Commit the staged changes to the local repository.

Run the **commit** command, where **-m** is followed by the commit tag.

```
git commit -m Tag
```

If **1 file changed** is displayed, the commit is successful. At this point, the code on the local **master** branch and the code on the **dev** branch are different, which means that there are two code versions in the local repository. If you **run the checkout command to switch branches**, you will see the content of the **\src\main\webapp\index.jsp** file is different on the two branches.

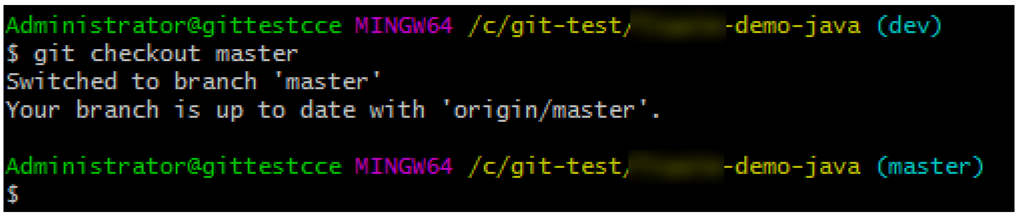
----End

Merging Local Branches

In the previous sections, you have created a **dev** branch and edited a file on the branch. In actual development, there are usually multiple **dev** branches. Before pushing code to the cloud repository, ensure all the edited branches are merged into the **master** branch in the local repository, so the code on the **master** branch is of the most complete and latest version.

Step 1 Run the following command to switch to the **master** branch:

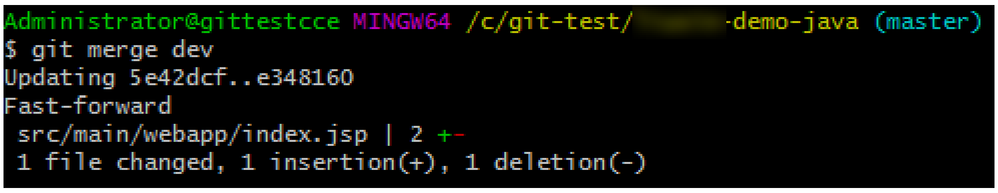
```
git checkout master
```



```
Administrator@gittestcce MINGW64 /c/git-test/[redacted]-demo-java (dev)
$ git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.

Administrator@gittestcce MINGW64 /c/git-test/[redacted]-demo-java (master)
$
```

Step 2 Run the **merge** command to merge the changes on the **dev** branch to the **master** branch.

```
git merge dev
```



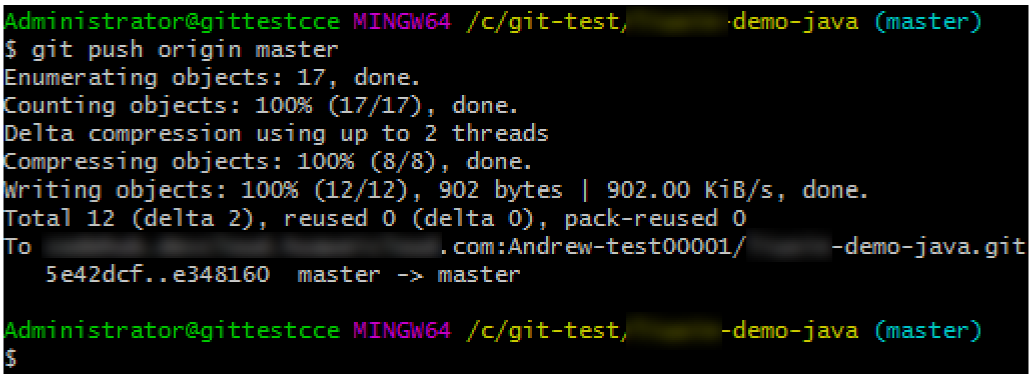
```
Administrator@gittestcce MINGW64 /c/git-test/[redacted]-demo-java (master)
$ git merge dev
Updating 5e42dcf..e348160
Fast-forward
 src/main/webapp/index.jsp | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
```

----End

Pushing Local Changes to the Cloud Repository

Run the **push** command to push the local **master** branch to the remote repository.

```
git push origin master
```



```
Administrator@gittestcce MINGW64 /c/git-test/[redacted]-demo-java (master)
$ git push origin master
Enumerating objects: 17, done.
Counting objects: 100% (17/17), done.
Delta compression using up to 2 threads
Compressing objects: 100% (8/8), done.
Writing objects: 100% (12/12), 902 bytes | 902.00 KiB/s, done.
Total 12 (delta 2), reused 0 (delta 0), pack-reused 0
To [redacted].com:Andrew-test00001/[redacted]-demo-java.git
   5e42dcf..e348160  master -> master

Administrator@gittestcce MINGW64 /c/git-test/[redacted]-demo-java (master)
$
```

The preceding figure indicates that the push is successful. Go to the repository list in CodeArts Repo, click the corresponding repository name, and check the **\src\main\webapp\index.jsp** file. You can see the changes made in the local repository, and the file update time and commit message are also changed.

By now, you have learned how to make changes on the local repository and push them to the cloud repository in CodeArts Repo.

Releasing Resources

In this section, you will delete the CodeArts Repo cloud repository and project that you previously created so fees will not be generated for the repository storage.

CAUTION

Deleted projects and repositories cannot be recovered.

Step 1 Delete a cloud repository.

1. Go to a target project and choose **Repo** from the navigation pane.
2. Click ··· on the row of the target repository and select **Delete Repository**. In the dialog box displayed, enter the repository name as prompted and click **OK**.

Step 2 (Optional) Delete the local repository.

If the local repository is no longer needed, you can delete it by deleting the repository directory to free up storage space.

----End