Hands-on Lab: Getting Started with Branches using Git Commands

Estimated time needed: 25 minutes

Objectives

After completing this lab, you will be able to use git commands to work with branches on a local repository, including:

1. Create a new local repository using `git init`
2. Create and add a file to the repo using `git add`
3. Commit changes using `git commit`
4. Create a branch using `git branch`
5. Switch to a branch using `git checkout`
6. Check the status of files changed using `git status`
7. Review recent commits using `git log`
8. Revert changes using `git revert`
9. Get a list of branches and active branches using `git branch`
10. Merge changes in your active branch into another branch using `git merge`

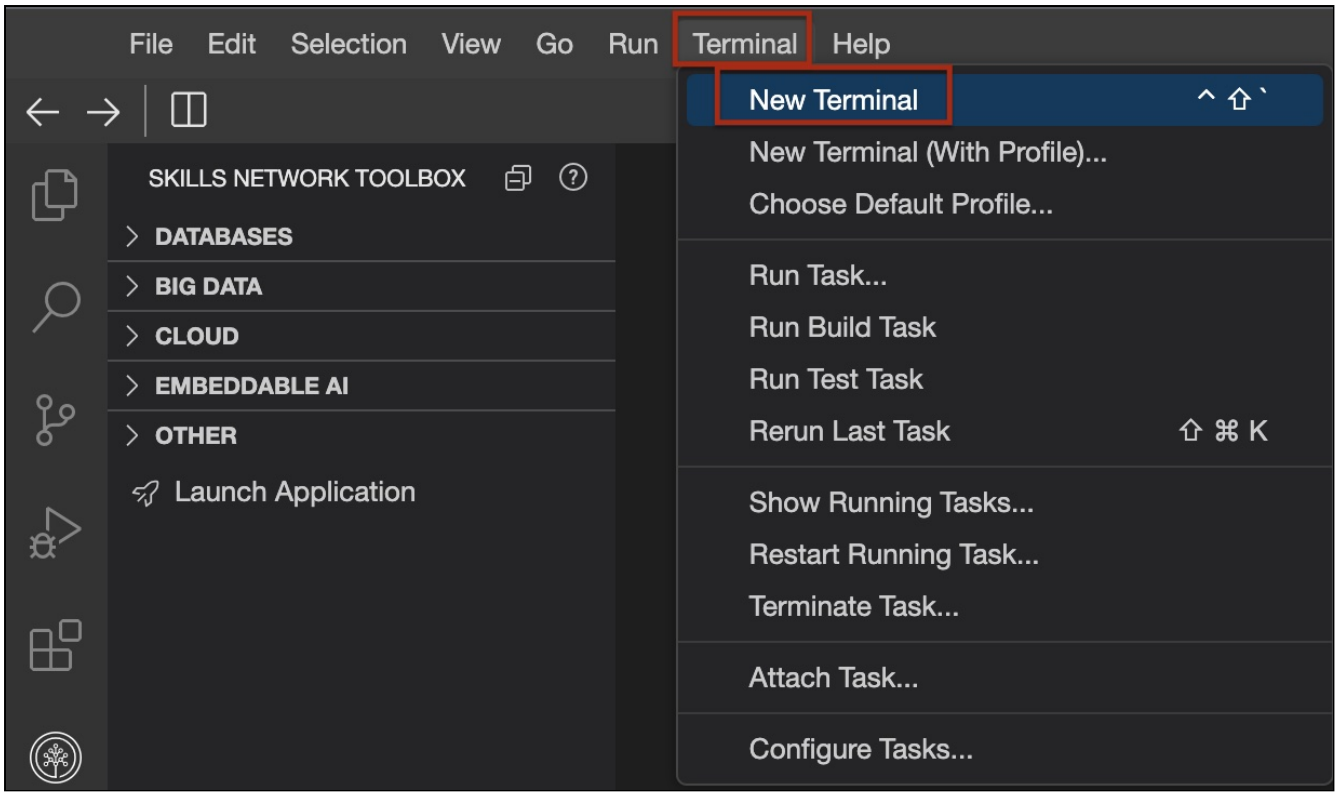
Pre-requisites

This lab is designed to be run on Skills Network, Cloud IDE that runs on a Linux system in the cloud and already has git installed. If you intend to run this lab on your own system, please ensure you have git (on Linux or MacOS) or Git Bash (on Windows) installed. Please check if you are using valid credentials and make sure you have a stable internet connection and a supported broswer. In case of any issues, please consider clearing cache/cookies and retry.

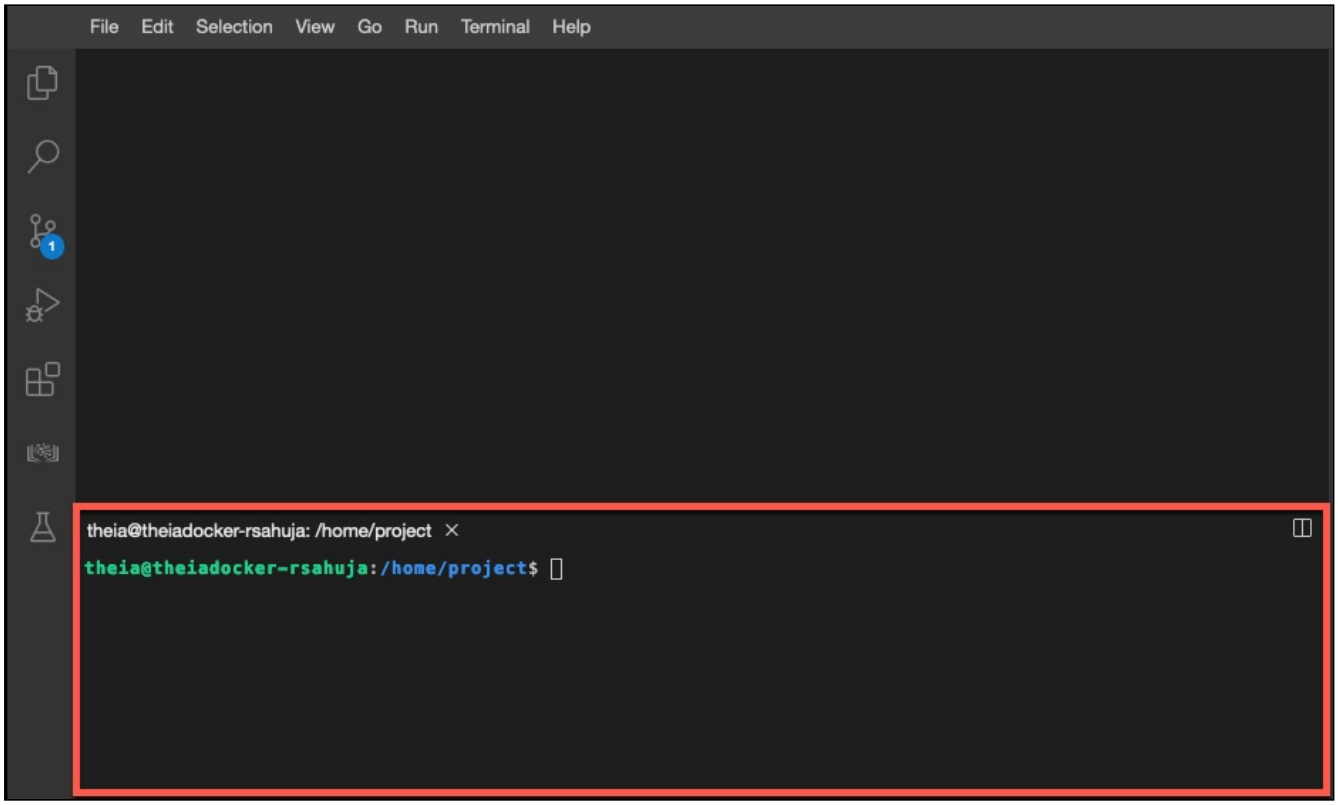
Initialize: Open a new terminal window

Open a terminal window in your IDE where you can start entering your shell and Git commands.

1. Click on the `Terminal` menu to the right of this instructions pane and then click on `New Terminal`.



2. This will add a new terminal window at the bottom where you can start entering commands.

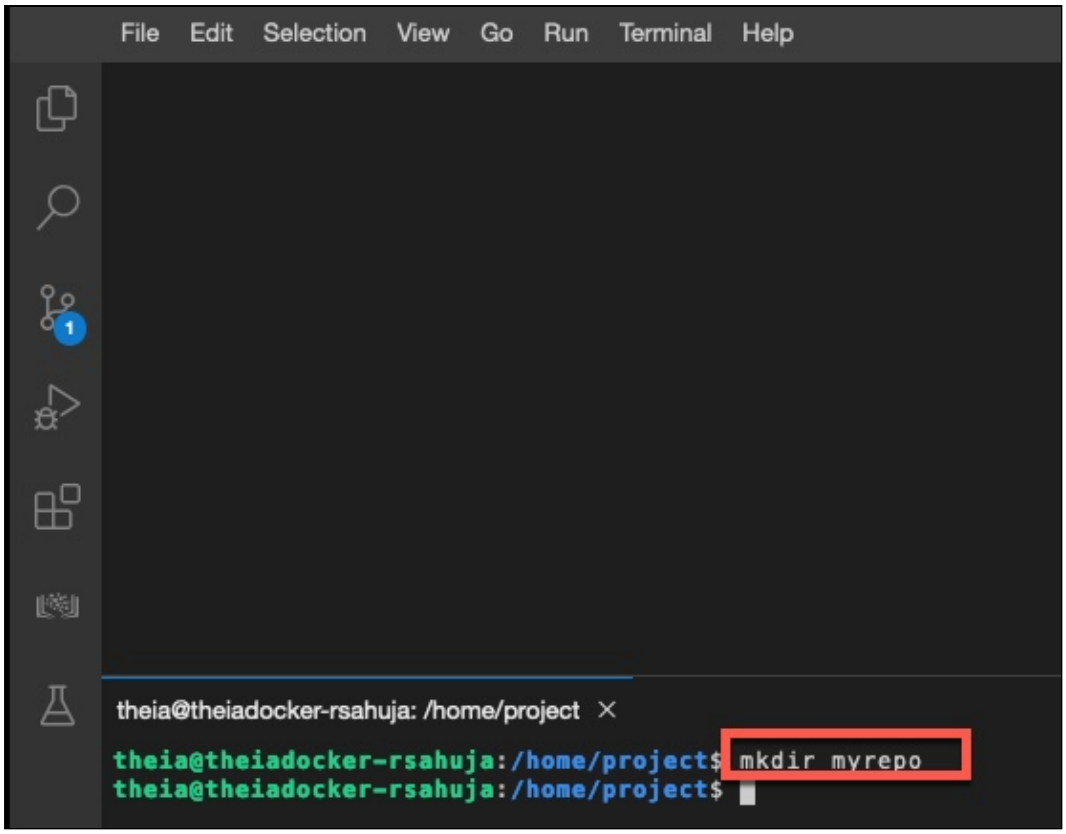


Exercise 1: Create a new local repo

1. Create a `myrepo` directory by running the `mkdir` command given below in the terminal.

```
1. 1
1. mkdir myrepo
```

Copied! Executed!



2. Go into the `myrepo` directory by running the following command.

```
1
cd myrepo
```

Copied! Executed!

3. Initiate the `myrepo` directory as a git repository by using the `git init` command.

```
1
git init
```

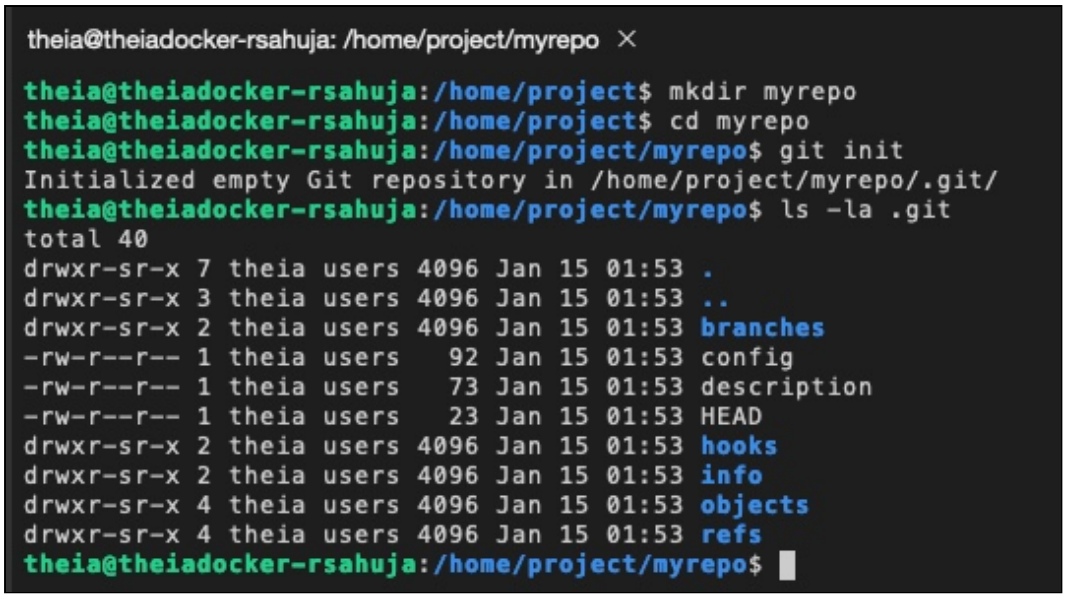
Copied! Executed!

4. A local git repository is now initiated with a `.git` folder containing all the git files, which you can verify by doing a directory listing by running the following command into the terminal window. The `.` prefix will make the git directory hidden. The `-la` option renders a long list, including the access permission, time of creation and other details for all the files in the hidden git directory.

```
1
ls -la .git
```

Copied! Executed!

The output shows the contents of the `.git` sub-directory which contains all the information required by git server.



Exercise 2: Create and add a file to the local repo

1. Now create an empty file named `newfile` using the following `touch` command.

```
1
```

```
touch newfile
```

Copied! Executed!

2. Add this file to the repo use the following `git add` command.

```
1
```

```
git add newfile
```

Copied! Executed!

Exercise 3: Commit changes

1. Before you commit the changes, you need to tell Git who you are. You can do this using the following commands. Replace "you@example.com" with the email address you use to login to GitHub. Replace "Your Name" with your name.

   Please note the email and name have to be within quotes.

```
1
```

```
git config --global user.email "you@example.com"
```

Copied!

```
1
```

```
git config --global user.name "Your Name"
```

Copied!

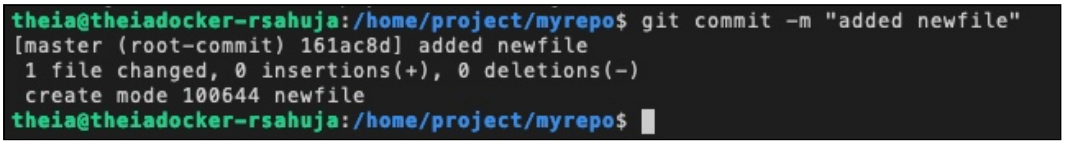
2. You can now commit your changes using the following `git commit` command.

   Note that the commit requires a message, which you can include using the `-m` parameter.

```
1
```

```
git commit -m "added newfile"
```

Copied! Executed!



Exercise 4: Create a branch

1. Your previous commit created a default main branch called `master`.
2. To make subsequent changes in your repository, create a new branch in your local repostitory. Run the following `git branch` command into the terminal to create a branch called `my1stbranch`.

```
1
```

```
git branch my1stbranch
```

Copied! Executed!

Exercise 5: Get a list of branches and active branch

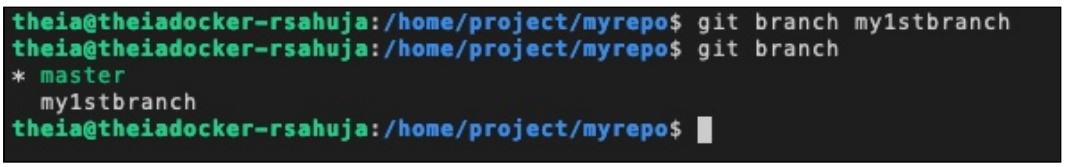
1. Check the list of branches your repository contains by running the following `git branch` command.

```
1
```

```
git branch
```

Copied! Executed!

2. Note that the output lists two branches: The default `master` branch with an asterisk `*` next to it indicating that it is the currently active branch and the newly created `mys1stbranch`.



Exercise 6: Switch to a different branch

1. Since you now want to work in the new branch to make your changes, run the following `git checkout` command to make it the active branch.

```
1
```

```
git checkout my1stbranch
```

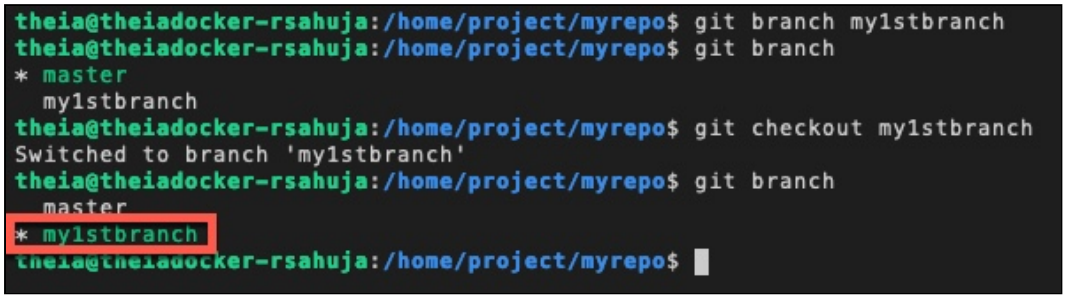
Copied! Executed!

2. Verify that the new branch is now the active branch by running the following `git branch` command.

```
1. 1
1. git branch
```

Copied! Executed!

3. Note that the asterisk `*` is now next to the `my1stbranch`, indicating that it is now active.



As a shortcut, instead of creating a branch using `git branch` and then making it active using `git checkout`, you can use the git checkout command followed by the `-b` option, that creates the branch and makes it active in one step.

```
1. 1
1. git checkout -b my1stbranch
```

Copied! Executed!

Exercise 7: Make changes in your branch and check the status of files added or changed

1. Make some changes to your new branch, called `my1stbranch`. Start by adding some text to `newfile` by running the following command into the terminal that will append the string "Here is some text in my newfile." into the file.

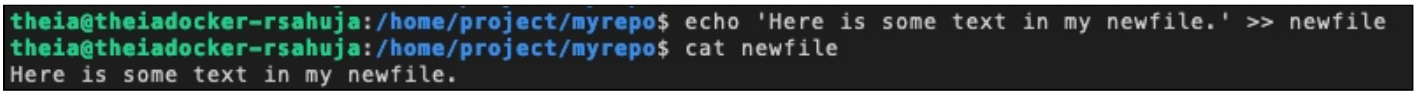
```
1. 1
1. echo 'Here is some text in my newfile.' >> newfile
```

Copied! Executed!

2. Verify the text has been added by running the following `cat` command.

```
1. 1
1. cat newfile
```

Copied! Executed!



3. Now, create another file called `readme.md` using the following command.

```
1. 1
```

```
1. touch readme.md
```

Copied! Executed!

4. And now, add it to the repo with the following `git add` command.

```
1. 1
```

```
1. git add readme.md
```

Copied! Executed!

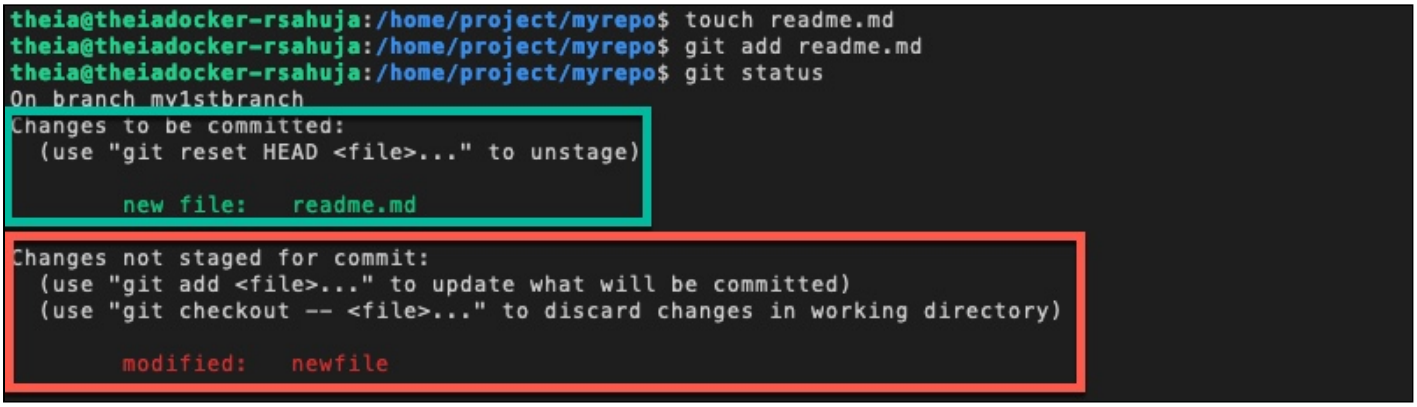
5. So far, in your new branch, you have edited the `newfile` and added a file called `readme.md`. You can easily verify the changes in your current branch using the `git status` command.

```
1. 1
```

```
1. git status
```

Copied! Executed!

6. The output of the `git status` command shows that the file `readme.md` has been added to the branch and is ready to be committed since you added it to the branch using `git add` . However, even though you modified the file called `newfile` you did not explicitly add it using `git add`, and hence it is not ready to be committed.



7. A shortcut to adding all modifications and additions is to use the following `git add` command with an asterisk `*` . This will also add the modified file `newfile` to the branch and make it ready to be committed.

```
1. 1
```

```
1. git add *
```

Copied! Executed!

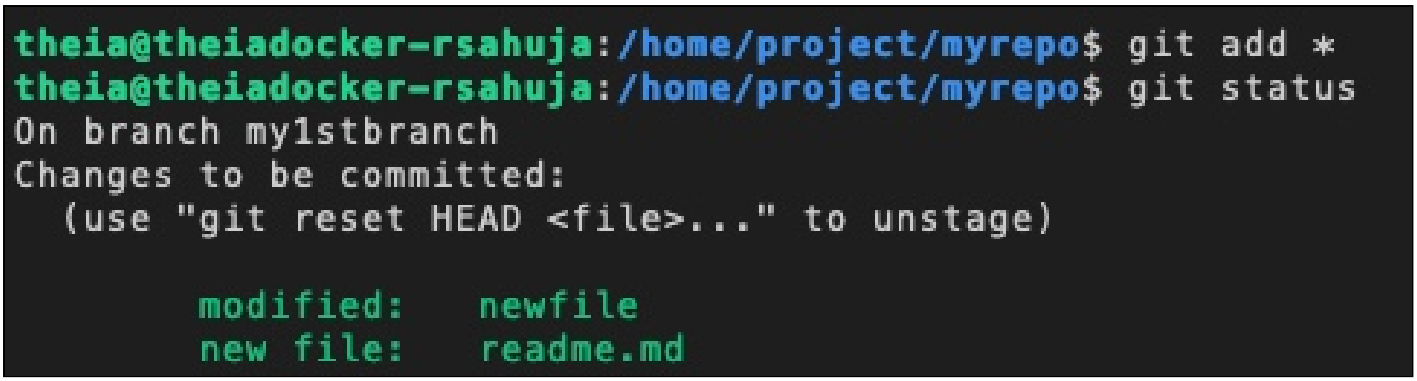
8. Let's check the status again.

```
1. 1
```

```
1. git status
```

Copied! Executed!

9. The output now shows that both the files can now be comitted.



Exercise 8: Commit and review commit history

1. Now that your changes are ready, you can save them to the branch using the following commit command with a message indicating the changes.

```
1
```

```
git commit -m "added readme.md modified newfile"
```

Copied! Executed!

2. We can run the following `git log` command to get a history of recent commits:

```
1
```

```
git log
```

Copied! Executed!

3. The log shows two recent commits: The last commit to `my1stbranch` as well as the previous commit to `master`.

Note: To exit the `git log` command, simply press the "Q" key. This action will close the log view and bring you back to the command prompt.

Exercise 9: Revert committed changes

1. Sometimes, you may not fully test your changes before committing them, which may have undesirable consequences. You can back out your changes by using a `git revert` command like the following.

You can either specify the ID of your commit that you can see from the previous log output or use the shortcut `HEAD` to rollback the last commit:

```
1
```

```
git revert HEAD --no-edit
```

Copied!

NOTE: If you don't specify the `--no-edit` flag, you may be presented with an editor screen showing the message with changes to be reverted. In that case, press the `Control` (or Ctrl) key simultaneously with `X`.

2. The output shows the most recent commit with the specified id has been reverted.

```
theia@theiadocker-rsahuja:/home/project/myrepo$ git revert HEAD --no-edit
[my1stbranch f4f5600] Revert "modified newfile added readme.md"
 Date: Sat Jan 15 04:39:38 2022 +0000
 2 files changed, 1 deletion(-)
 delete mode 100644 readme.md
theia@theiadocker-rsahuja:/home/project/myrepo$
```

Exercise 10: Merge changes into another branch

1. Let's make one more change in your currently active `my1stbranch` using the following commands.

```
1
2
3
4
```

```
touch goodfile
git add goodfile
git commit -m "added goodfile"
git log
```

Copied!

2. The output of the log shows the newly added `goodfile` has been comitted to the `my1stbranch` branch:

```
theia@theiadocker-rsahuja:/home/project/myrepo$ git status
On branch my1stbranch
nothing to commit, working tree clean
theia@theiadocker-rsahuja:/home/project/myrepo$ touch goodfile
theia@theiadocker-rsahuja:/home/project/myrepo$ git add goodfile
theia@theiadocker-rsahuja:/home/project/myrepo$ git commit -m "added goodfile"
[my1stbranch d8680b4] added goodfile
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 goodfile
theia@theiadocker-rsahuja:/home/project/myrepo$ git log
commit d8680b4cf63ff3e97b2970704806c14dc6134dd1 (HEAD -> my1stbranch)
Author: Your Name <you@example.com>
Date:   Sat Jan 15 04:53:27 2022 +0000

    added goodfile
```

Note: To exit the `git log` command, simply press the "Q" key. This action will close the log view and bring you back to the command prompt.

3. Now, let's merge the contents of the `my1stbranch` into the `master` branch. We will first need to make the `master` branch active using the following `git checkout` command.

```
1
```

```
git checkout master
```

Copied!

4. Now, let's merge the changes from `my1stbranch` into master.

```
1
2
```

```
git merge my1stbranch
git log
```

Copied!

5. Output and log shows the successful merging of the branch.

```
theia@theiadocker-rsahuja:/home/project/myrepo$ git checkout master
Switched to branch 'master'
theia@theiadocker-rsahuja:/home/project/myrepo$ git merge my1stbranch
Updating 8954f54..d8680b4
Fast-forward
 goodfile | 0
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 goodfile
theia@theiadocker-rsahuja:/home/project/myrepo$ git log
commit d8680b4cf63ff3e97b2970704806c14dc6134dd1 (HEAD -> master, my1stbranch)
Author: Your Name <you@example.com>
Date:   Sat Jan 15 04:53:27 2022 +0000

    added goodfile
```

6. Now that changes have been merged into `master` branch, the `my1stbranch` can be deleted using the following `git branch` command with the `-d` option:

```
1
```

```
git branch -d my1stbranch
```

Copied!

```
theia@theiadocker-rsahuja:/home/project/myrepo$ git branch -d my1stbranch
Deleted branch my1stbranch (was d8680b4).
theia@theiadocker-rsahuja:/home/project/myrepo$
```

Exercise 11: Practice on your own

1. Create a new directory and branch called `newbranch`
2. Make `newbranch` the active branch
3. Create an empty file called `newbranchfile`
4. Add the newly created file to your branch
5. Commit the changes in your `newbranch`
6. Revert the last committed changes
7. Create a new file called `newgoodfile`
8. Add the latest file to `newbranch`

9. Commit the changes
10. Merge the changes in `newbranch` into `master`

Summary

In this lab, you have learned how to create and work with branches using Git commands in a local repository. In a subsequent lab, you will learn how to synchronize changes in your local repository with remote GitHub repositories.

Author(s)

Rav Ahuja

Other Contributor(s)

Richard Ye