



Helvetica N...



Step 13 of 13

Hands-on Lab: Getting started with branches using git commands on a local repository.



Effort : 25 mins

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 branch using `git branch`
10. merge changes in your active branch into another branch using `git merge`

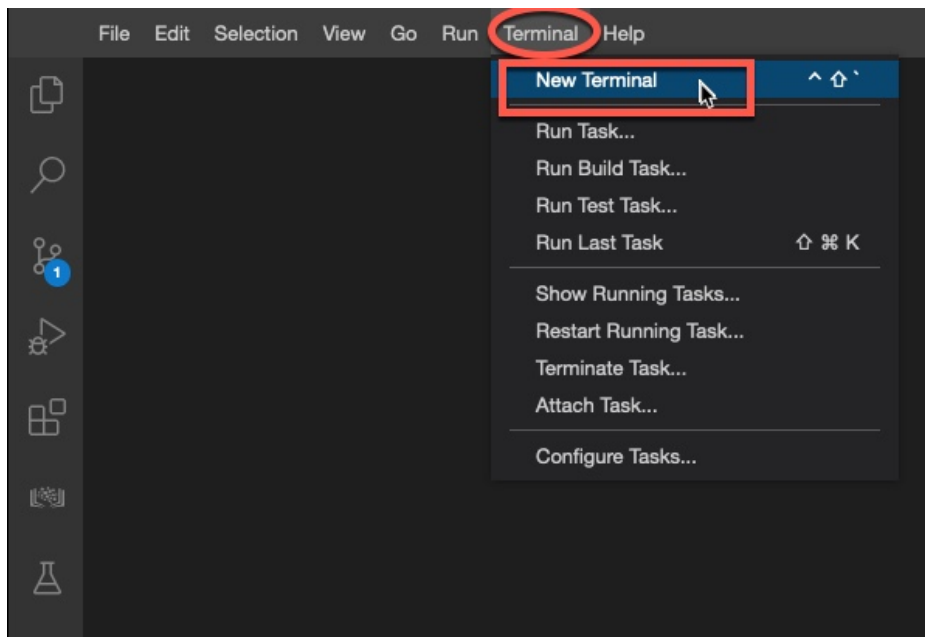
Pre-requisites

This lab is designed to be run on Skills Network - Cloud IDE which is 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 GitBash (on Windows) installed.

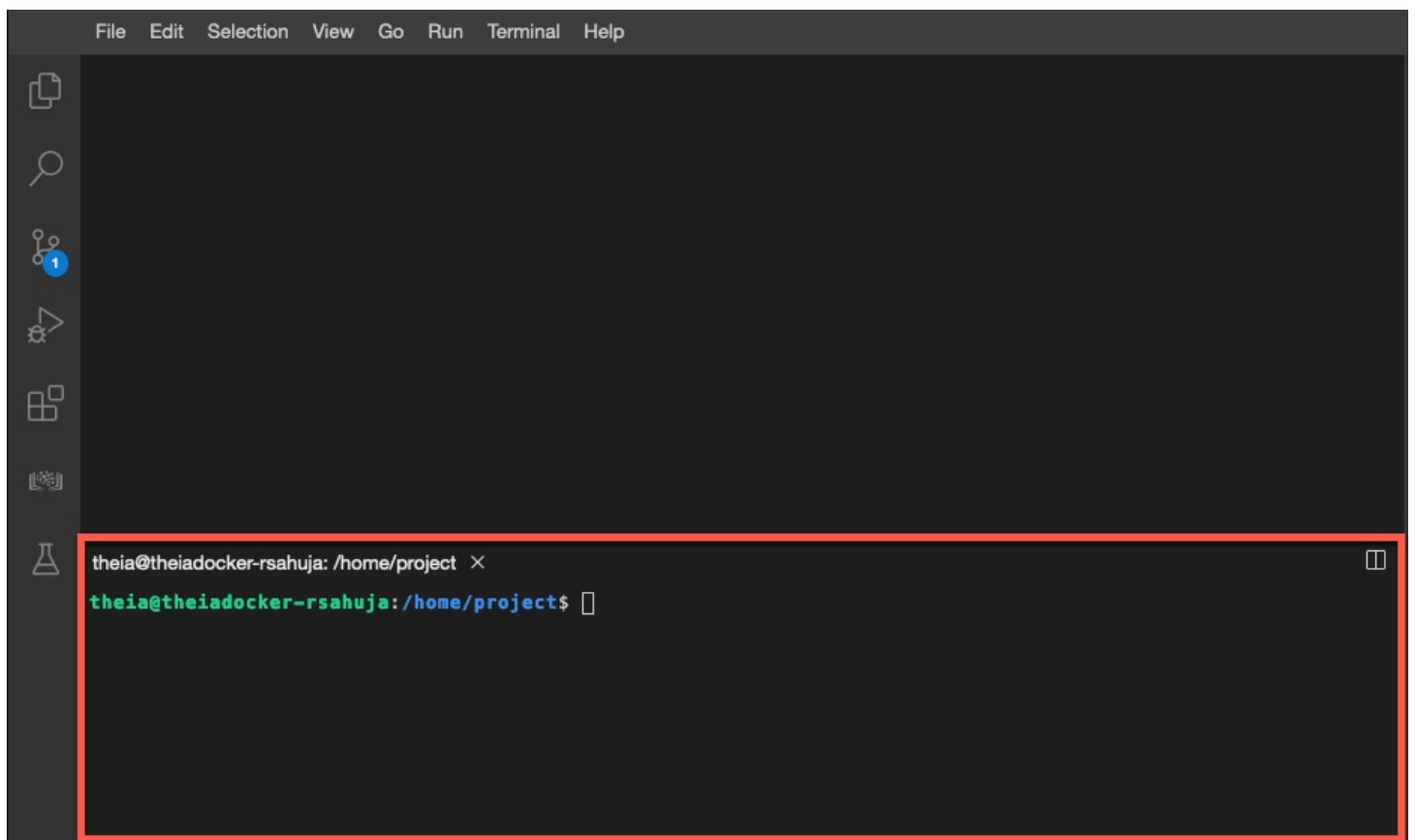
Initialize: Open a new terminal window

Let's first open a terminal window in our IDE where we can start entering our 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.

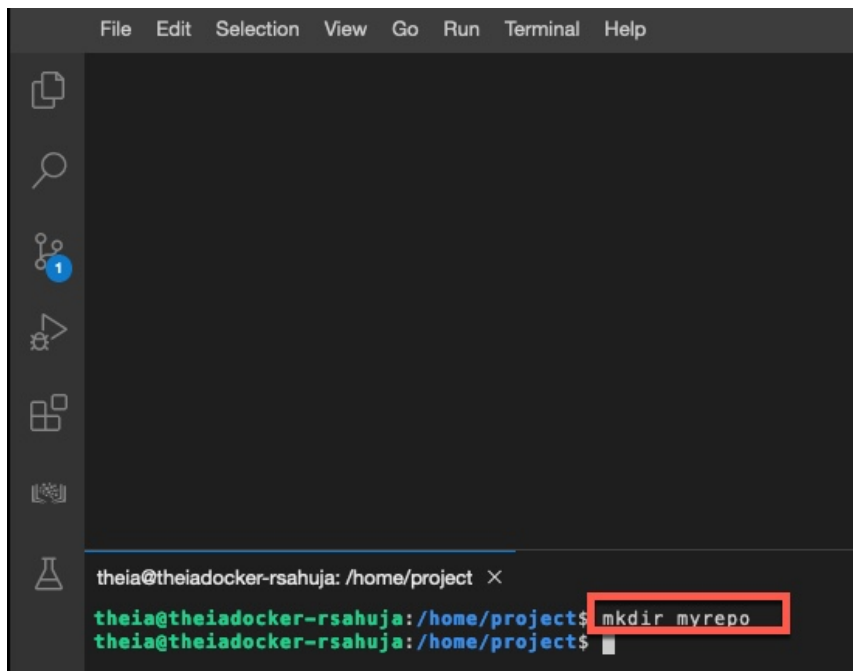


Exercise 1: Create a new local repo

1. Now let us create a new directory for our local repository.

Create a myrepo directory by copying and pasting the `mkdir` command below into the terminal:

```
mkdir myrepo
```

A screenshot of a terminal window with a dark background. The terminal shows the prompt 'theia@theiadocker-rsahuja: /home/project' followed by the command 'mkdir myrepo' which has been executed. The command is highlighted with a red rectangular box. The terminal interface includes a menu bar at the top with options: File, Edit, Selection, View, Go, Run, Terminal, Help. On the left side, there is a vertical toolbar with icons for file operations, search, and development tools.

2. Go into the myrepo directory by copying and pasting the cd command below:

```
cd myrepo
```

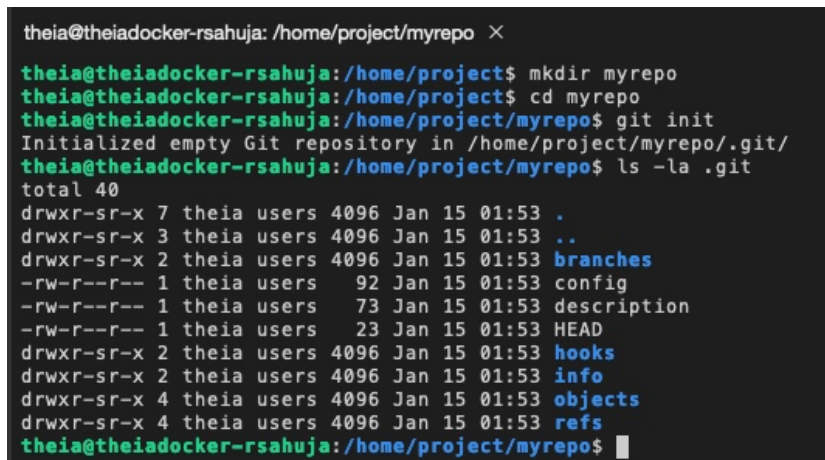
3. In this myrepo directory lets create a new local git repository using the git init command. Copy and paste the command below into the terminal:

```
git init
```

4. A new local repository is now created, which you can verify by doing a directory listing by pasting the following command into the terminal window:

```
ls -la .git
```

The output shows the contents of the .git sub-directory which houses the local repo:

A screenshot of a terminal window showing the output of the 'ls -la .git' command. The terminal shows the prompt 'theia@theiadocker-rsahuja: /home/project/myrepo' followed by the command 'ls -la .git'. The output lists the contents of the .git directory, including files like ., .., branches, config, description, HEAD, hooks, info, objects, and refs. The command and its output are highlighted with a red rectangular box. The terminal interface is the same as the previous screenshot.

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

1. Now lets create an empty file using the following touch command:

```
touch newfile
```

2. Add this file to the repo using the following git add command:

```
git add newfile
```

Exercise 3: Commit changes

1. Before we can commit our changes, we need to tell git who we are. We can do this using the following commands (you can copy these commands as is, no need to enter your actual information):

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

2. Once the repo has the newfile in it let's commit our changes using the the following `git commit` command. Note that the commit requires a message which we include using the `-m` parameter:

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

Exercise 4: Create a branch

0. Our previous commit created a default main branch called `master`.

1. To make subsequent changes in our repo, lets create a new branch in our local repository. Copy and paste the following `git branch` command into the terminal to create a branch called `my1stbranch`:

```
git branch my1stbranch
```

Exercise 5: Get a list of branches and active branch

1. Let's check which branches our repo contains by pasting the following `git branch` command into the terminal:

```
git branch
```

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

Exercise 6: Switch to using a different branch

1. Since we now want to work in the new branch issue the following `git checkout` command to make it the active branch to make your changes in:

```
git checkout my1stbranch
```

2. Let's verify that the new branch is now the active branch by issuing the following `git branch` command:

```
git branch
```

3. Note that the asterix `*` is now next to the `my1stbranch` indicating that it is now active:

As a shortcut to creating a branch using `git branch` and then making it active using `git checkout` you can use the shortcut like follows with the `-b` option that creates the branch and makes it active in one step:

```
git checkout -b my1stbranch
```

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

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

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

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

```
cat newfile
```

3. Now let's create another file called `readme.md` using the following command:

`touch readme.md`

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

```
git add readme.md
```

5. So far in our new branch we have edited the `newfile` and added a file called `readme.md`. We can easily verify the changes in our current branch using the `git status` command:

```
git status
```

6. The output of the `git status` command shows that the files `readme.md` has been added to the branch and is ready to be committed, since we added it to the branch using `git add`. However, even though we modified the file called `newfile` we 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:

```
git add *
```

8. Let's check the status again:

```
git status
```

9. The output now shows both the files can now be committed:

Exercise 8: Commit and review commit history

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

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

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

```
git log
```

3. The log shows 2 recent commits - the last commit to `my1stbranch` as well as the previous commit to `master`:

Exercise 9: Revert committed changes

1. Sometimes you may not fully test your changes before committing them and 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:

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

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:

Exercise 10: merge changes into another branch

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

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

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

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:

```
git checkout master
```

4. Now lets merge the changes from my1stbranch into master

```
git merge my1stbranch  
git log
```

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

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:

```
git branch -d my1stbranch
```

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

Changelog

Date	Version	Changed by	Change Description
2022-01-14	1.0	Rav Ahuja	Initial version created
2022-01-27	1.1	Richard Ye	Added git config instructions

[Previous](#)