**git branch**

Up to this point, you’ve worked in a single Git branch called master. Git allows us to create *branches* to experiment with versions of a project. Imagine you want to create version of a story with a happy ending. You can create a new branch and make the happy ending changes to that branch only. It will have no effect on the master branch until you’re ready to merge the happy ending to the master branch.

In this lesson, we’ll be using Git branching to develop multiple versions of a resumé.

You can use the command below to answer the question: “which branch am I on?”

git branch

**Instructions**

**1.**

Check what branch you are currently on.

In the output, the \* (asterisk) is showing you what branch you’re on. The project only has one branch at this time.

# branching overview

The diagram to the right illustrates branching.

* The circles are commits, and together form the Git project’s commit history.
* New Branch is a different version of the Git project. It contains commits from Master but also has commits that Master does not have.

Click Next to make your first new branch.



# git branch 2

Right now, the Git project has only one branch: master.

To create a new branch, use:

git branch new\_branch

Here new\_branch would be the name of the new branch you create, like photos or blurb. Be sure to name your branch something that describes the purpose of the branch. Also, branch names can’t contain whitespaces: new-branch and new\_branch are valid branch names, but new branch is not.

**Instructions**

**1.**

Let’s create a new version of a resumé to apply for a fencing instructor role.

Create a new branch called fencing.

Remember to spell the word “fencing” correctly.

Next, view your branches as you did in the previous exercise.

Notice in the output there now appear two branches: master and fencing.

# git checkout

Great! You just created a new branch.

The master and fencing branches are identical: they share the same exact commit history. You can switch to the new branch with

git checkout branch\_name

Here, branch\_name is the name of the branch. If the branch’s name is skill

git checkout skill

Once you switch branch, be now able to make commits on the branch that have no impact on master.

You can continue your workflow, while master stays intact!

**Instructions**

**1.**

Switch to the fencing branch from the master branch.

**2.**

Use git branch to verify that you have switched branches.

In the output, notice the \* is now over the fencing branch.

**commit on a new branch**

Congratulations! You have switched to a new branch. All the commands you do on master, you can also do on this branch.

For example, to add files to the staging area, use:

git add filename

And to commit, use:

git commit -m "Commit message"

In a moment, you will make a commit on the fencing branch. On the far right, the diagram shows what will happen to the Git project.

**Instructions**

**1.**

Print the Git commit log.

Notice the output:

* The commits you see were all made in the master branch. fencing inherited them.
* This means that every commit master has, fencing also has.

*Note*: if you find that your cursor is stuck in Git log, press q to escape.

**2.**

In **resume.txt**, replace your skill at scheming against Hook with your experience in sword-fights.

Delete this line:

-Scheme against Captain Hook

and type this line in its place:

-Engage in swordfights with pirates

**3.**

Add **resume.txt** into the staging area.

**4.**

Commit the changes to the repository with a commit message.

**git merge**

What if you wanted include all the changes made to the fencing branch on the master branch? We can easily accomplish this by *merging* the branch into master with:

git merge branch\_name

For example, if I wanted to merge the skills branch to master, I would enter

git merge skills

In a moment, you’ll merge branches. Keep in mind:

* Your goal is to update master with changes you made to fencing.
* fencing is the giver branch, since it provides the changes.
* master is the receiver branch, since it accepts those changes.

**Instructions**

**1.**

You are currently on the fencing branch. Switch over to the master branch.

**2.**

Your sword-fighting experience is so impressive that it belongs on the master version of your resumé.

From the terminal, merge the fencing branch into the master branch.

Notice the output: The merge is a “fast forward” because Git recognizes that fencing contains the most recent commit. Git *fast forwards* master to be up to date with fencing.

# merge conflict I

The merge was successful because master had not changed since we made a commit on fencing. Git knew to simply update master with changes on fencing.

What would happen if you made a commit on master before you merged the two branches? Furthermore, what if the commit you made on master altered the same exact text you worked on in fencing? When you switch back to master and ask Git to merge the two branches, Git doesn’t know which changes you want to keep. This is called a merge conflict.

**Instructions**

**1.**

You are on the master branch. In the code editor, where you have written:

-Engage in swordfights with pirates

Add the word “professional”, so the text reads:

-Engage in swordfights with professional pirates

Click Run.

**2.**

Add **resume.txt** to the staging area.

**3.**

Commit the changes to the repository with a commit message.

**4.**

Imagine a few weeks have passed, and you’d like to develop your fencing resumé some more.

Switch back to the fencing branch.

**5.**

From fencing, change the line so it reads:

-Engage in swordfights with professional pirates such as Smee.

Click Run.

**6.**

Once again, add **resume.txt** to the staging area.

**7.**

Commit the changes to the repository with a commit message.

# merge conflict II

Let’s say you decide you’d like to merge the changes from fencing into master.

Here’s where the trouble begins!

You’ve made commits on separate branches that alter the same line in conflicting ways. Now, when you try to merge fencing into master, Git will not know which version of the file to keep.

**Instructions**

**1.**

Switch to the master branch.

**2.**

From the terminal, enter the command below:

git merge fencing

This will try to merge fencing into master.

In the output, notice the lines:

CONFLICT (content): Merge conflict in resumé.txt

Automatic merge failed; fix conflicts and then commit the result.

**3.**

We must fix the merge conflict.

In the code editor, look at **resume.txt**. Git uses markings to indicate the HEAD (master) version of the file and the fencing version of the file, like this:

<<<<<<< HEAD

master version of line

=======

fencing version of line

>>>>>>> fencing

**Note**: If the markings are not showing in resume.txt, please close resume.txt and re-open via the folder icon at the top left corner of the editor.

Git asks us which version of the file to keep: the version on master or the version on fencing. You decide you want the fencing version.

From the code editor:

Delete the content of the line as it appears in the master branch

Delete **all of Git’s special markings** including the words HEAD and fencing. If any of Git’s markings remain, for example, >>>>>>> and =======, the conflict remains.

**4.**

Add **resume.txt** to the staging area.

**5.**

Now, make a commit. For your commit message, type “Resolve merge conflict” to indicate the purpose of the commit.

[Here’s a hint](https://www.codecademy.com/en/courses/learn-git/lessons/git-workflow/exercises/git-commit) for the commit command.

# delete branch

In Git, branches are usually a means to an end. You create them to work on a new project feature, but the end goal is to merge that feature into the master branch. After the branch has been integrated into master, it has served its purpose and can be deleted.

The command

git branch -d branch\_name

will delete the specified branch from your Git project.

Now that master contains all the file changes that were in fencing, let’s delete fencing.

**Instructions**

**1.**

Delete the fencing branch.

Now, verify that you have indeed deleted fencing by listing all your project’s branches on the terminal.

Notice in the output that only one branch, master, is shown.

Click Next to continue!

# generalizations

Let’s take a moment to review the main concepts and commands from the lesson before moving on.

* Git branching allows users to experiment with different versions of a project by checking out separate branches to work on.

The following commands are useful in the Git branch workflow.

* git branch: Lists all a Git project’s branches.
* git branch branch\_name: Creates a new branch.
* git checkout branch\_name: Used to switch from one branch to another.
* git merge branch\_name: Used to join file changes from one branch to another.
* git branch -d branch\_name: Deletes the branch specified.