**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

# 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.

# 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 masterstays intact!

# 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.

**git merge**

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

git merge branch\_name

For example, if I wanted to merge the skillsbranch 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.

# merge conflict I

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

# 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.

# 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.

**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.