File and Directories

Where I am?

The part of the operating system responsible for managing files and directories is called the filesystem. It organizes data into files, which hold information, and directories (also called "folders"), which hold files or other directories.

Every file or directory is identified by an absolute path that specifies how to get to it from the top (or root) of the filesystem. For example, the path /home/repl is the path to a directory called repl inside a directory called home, while the path /home/repl/course.txt identifies a file called course.txt inside that directory.

To find out where you are in the filesystem, type the command pwd (which stands for "print working directory"). This tells you the absolute path of the directory where the shell will run commands and look for files unless and until you tell it to do so elsewhere. You can also use the command whoami to find out who the computer thinks you are.

**Relative paths**

100xp

An absolute path is like the latitude and longitude of a point on a map: it specifies the same thing no matter where you are. A *relative path*, on the other hand, specifies a location starting from where you are: it's like saying "20 kilometers north".

For example, if you are in the directory /home/repl, the relative path seasonal specifies the same directory as /home/repl/seasonal, and the relative path seasonal/winter.csv specifies the same file as /home/repl/seasonal/winter.csv. The shell can tell whether a path is absolute or relative by looking at its first character. If the path begins with /, it is absolute; if it doesn't, it is relative.

**Changing directories**

100xp

Just as you can move around in a file browser by double-clicking on folders, you can move around in the filesystem using the command cd (which stands for "change directory").

If you type cd seasonal and then type pwd, the shell will tell you that you are now in /home/repl/seasonal. If you then run ls on its own, it shows you the contents of /home/repl/seasonal, because that's where you are. If you want to get back to your home directory /home/repl, you can use the command cd /home/repl.

**GIT HUB TUTORIAL**

**Switching branches**

can use:

git checkout -b new\_branch

to checkout and create a branch at the same time. This is the same thing as doing:

git branch new\_branch  
git checkout new\_branch

# 1.20 Removing All The Things

Ok, so you're in the clean\_up branch. You can finally remove all those pesky octocats by using the git rm command which will not only remove the actual files from disk, but will also stage the removal of the files for us.

You're going to want to use a wildcard again to get all the octocats in one sweep, go ahead and run:

git rm '\*.txt'

Advice

**Remove all the things!**

Removing one file is great and all, but what if you want to remove an entire folder? You can use the recursive option on git rm:

git rm -r folder\_of\_cats

This will recursively remove all folders and files from the given directory.

# 1.21 Commiting Branch Changes

Now that you've removed all the cats you'll need to commit your changes.

Feel free to run git status to check the changes you're about to commit.

git commIf you happen to delete a file without using 'git rm' you'll find that you still have to 'git rm' the deleted files from the working tree. You can save this step by using the '-a' option on 'git commit', which auto removes deleted files with the commit.

git commit -am "Delete stuff"

it -m "Remove all the cats"

# 1.22 Switching Back to master

Great, you're almost finished with the cat... er the bug fix, you just need to switch back to the master branch so you can copy (or merge) your changes from the clean\_up branch back into the master branch.

Go ahead and checkout the master branch:

git checkout master

**Pull Requests**

If you're hosting your repo on GitHub, you can do something called a pull request.

A pull request allows the boss of the project to look through your changes and make comments before deciding to merge in the change. It's a really great feature that is used all the time for remote workers and open-source projects.

Check out the [pull request help page](https://help.github.com/articles/using-pull-requests) for more information.

# 1.23 Preparing to Merge

Alrighty, the moment has come when you have to merge your changes from the clean\_up branch into the master branch. Take a deep breath, it's not that scary.

We're already on the master branch, so we just need to tell Git to merge the clean\_up branch into it:

git merge clean\_up

**Merge Conflicts**

Merge Conflicts can occur when changes are made to a file at the same time. A lot of people get really scared when a conflict happens, but fear not! They aren't that scary, you just need to decide which code to keep.

Merge conflicts are beyond the scope of this course, but if you're interested in reading more, take a look the section of the [Pro Git book](http://git-scm.com/book) on [how conflicts are presented](http://git-scm.com/docs/git-merge#_how_conflicts_are_presented).

# 1.24 Keeping Things Clean

Congratulations! You just accomplished your first successful bugfix and merge. All that's left to do is clean up after yourself. Since you're done with the clean\_up branch you don't need it anymore.

You can use git branch -d <branch name> to delete a branch. Go ahead and delete the clean\_up branch now:

git branch -d clean\_up

What if you have been working on a feature branch and you decide you really don't want this feature anymore? You might decide to delete the branch since you're scrapping the idea. You'll notice that git branch -d bad\_feature doesn't work. This is because -d won't let you delete something that hasn't been merged.

You can either add the --force (-f) option or use -D which combines -d -f together into one command.

# 1.25 The Final Push

Here we are, at the last step. I'm proud that you've made it this far, and it's been great learning Git with you. All that's left for you to do now is to push everything you've been working on to your remote repository, and you're done!

git push

We only scratched the surface of Git in this course. There is so much more you can do with it. Check out the [Git documentation](http://git-scm.com/docs) for a full list of functions.

The [Pro Git book](http://git-scm.com/book), by Scott Chacon, is an excellent resource to teach you the inner workings of Git.

[help.github](https://help.github.com/) and GitHub Training are also great for anything related to Git in general and using Git with GitHub.