**GIT FAST TUTORIAL**

**1.1 Got 15 minutes and want to learn Git?**

Git allows groups of people to work on the same documents (often code) at the same time, and without stepping on each other's toes. It's a distributed version control system.

Our terminal prompt below is currently in a directory we decided to name "octobox". To initialize a Git repository here, type the following command:

git init

**{**

**Directory:** A folder used for storing multiple files.

**Repository:** A directory where Git has been initialized to start version controlling your files.

**Clicky Click:** Click on the instructions preceded by an arrow. They will be copied into the terminal prompt.

**}**

# 1.2 Checking the Status

Good job! As Git just told us, our "octobox" directory now has an empty repository in /.git/. The repository is a hidden directory where Git operates.

To save your progress as you go through this tutorial -- and earn a badge when you successfully complete it -- head over to [create a free Code School account](https://www.codeschool.com/account/courses/try-git/add). We'll wait for you here.

Next up, let's type the git status command to see what the current state of our project is:

git status

**{**

**The .git directory** on the left you'll notice a .git directory. It's usually hidden but we're showing it to you for convenience.

If you click it you'll notice it has all sorts of directories and files inside it. You'll rarely ever need to do anything inside here but it's the guts of Git, where all the magic happens.

It's healthy to run git status often. Sometimes things change and you don't notice it.

}

**1.3 Adding & Committing**

I created a file called octocat.txt in the octobox repository for you (as you can see in the browser below).

You should run the git status command again to see how the repository status has changed:

git status

**{**

**staged:** Files are ready to be committed.

**unstaged:** Files with changes that have not been prepared to be committed.

**untracked:** Files aren't tracked by Git yet. This usually indicates a newly created file.

**deleted:** File has been deleted and is waiting to be removed from Git

}

# 1.4 Adding Changes

Good, it looks like our Git repository is working properly. Notice how Git says octocat.txt is "untracked"? That means Git sees that octocat.txt is a new file.

To tell Git to start tracking changes made to octocat.txt, we first need to add it to the staging area by using git add.

git add octocat.txt

{

**add all:** You can also type git add -A . where the dot stands for the current directory, so everything in and beneath it is added. The -A ensures even file deletions are included.

**git reset:** You can use git reset <filename> to remove a file or files from the staging area.

**}**

# 1.5 Checking for Changes

Good job! Git is now tracking our octocat.txt file. Let's run git status again to see where we stand:

git status

# 1.6 Committing

Notice how Git says changes to be committed? The files listed here are in the Staging Area, and they are not in our repository yet. We could add or remove files from the stage before we store them in the repository.

To store our staged changes, we run the commit command with a message describing what we've changed. Let's do that now by typing:

git commit -m "Add cute octocat story"

{

**Staging Area:** A place where we can group files together before we "commit" them to Git.

**Commit** A "commit" is a snapshot of our repository. This way if we ever need to look back at the changes we've made (or if someone else does), we will see a nice timeline of all changes.

}

# 1.7 Adding All Changes

Great! You also can use wildcards if you want to add many files of the same type. Notice that I've added a bunch of .txt files into your directory below.

I put some in a directory named "octofamily" and some others ended up in the root of our "octobox" directory. Luckily, we can add all the new files using a wildcard with git add. Don't forget the quotes!

git add '\*.txt'

{

**Wildcards:** We need quotes so that Git will receive the wildcard before our shell can interfere with it. Without quotes our shell will only execute the wildcard search within the current directory. Git will receive the list of files the shell found instead of the wildcard and it will not be able to add the files inside of the octofamily directory.

}

# 1.8 Committing All Changes

Okay, you've added all the text files to the staging area. Feel free to run git status to see what you're about to commit.

If it looks good, go ahead and run:

git commit -m 'Add all the octocat txt files'

{

**Check all the things!**

When using wildcards you want to be extra careful when doing commits. Make sure to check what files and folders are staged by using git status before you do the actual commit. This way you can be sure you're committing only the things you want.

}

# 1.9 History

So we've made a few commits. Now let's browse them to see what we changed.

Fortunately for us, there's git log. Think of Git's log as a journal that remembers all the changes we've committed so far, in the order we committed them. Try running it now:

git log

{

**More useful logs:** Use git log --summary to see more information for each commit. You can see where new files were added for the first time or where files were deleted. It's a good overview of what's going on in the project.

}

# 1.10 Remote Repositories

Great job! We've gone ahead and created a new empty GitHub repository for you to use with Try Git at https://github.com/try-git/try\_git.git. To push our local *repo* to the GitHub server we'll need to add a remote repository.

This command takes a *remote name* and a *repository URL*, which in your case is https://github.com/try-git/try\_git.git.

Go ahead and run git remote add with the options below:

git remote add origin https://github.com/try-git/try\_git.git

{

**git remote:** Git doesn't care what you name your remotes, but it's typical to name your main one origin.

It's also a good idea for your main repository to be on a remote server like [GitHub](http://github.com) in case your machine is lost at sea during a transatlantic boat cruise or crushed by three monkey statues during an earthquake.

}

# 1.11 Pushing Remotely

The push command tells Git where to put our commits when we're ready, and boy we're ready. So let's push our local changes to our **origin** repo (on GitHub).

The name of our remote is origin and the default local branch name is master. The -u tells Git to remember the parameters, so that next time we can simply run git push and Git will know what to do. Go ahead and push it!

git push -u origin master

{

**Cool Stuff:** When you start to get the hang of git you can do some really cool things with hooks when you push.

For example, you can upload directly to a webserver whenever you push to your master remote instead of having to upload your site with an ftp client. Check out [Customizing Git - Git Hooks](http://git-scm.com/book/en/Customizing-Git-Git-Hooks) for more information.

}

# 1.12 Pulling Remotely

Let's pretend some time has passed. We've invited other people to our GitHub project who have pulled your changes, made their own commits, and pushed them.

We can check for changes on our GitHub repository and pull down any new changes by running:

git pull origin master

{

}

# 1.13 Differences

Uh oh, looks like there have been some additions and changes to the octocat family. Let's take a look at what is different from our last commit by using the git diff command.

In this case we want the diff of our most recent commit, which we can refer to using the HEAD pointer.

git diff HEAD

{

**HEAD** The HEAD is a pointer that holds your position within all your different commits. By default HEAD points to your most recent commit, so it can be used as a quick way to reference that commit without having to look up the SHA.

}

# 1.14 Staged Differences

Another great use for diff is looking at changes within files that have already been staged. Remember, staged files are files we have told git that are ready to be committed.

Let's use git add to stage octofamily/octodog.txt, which I just added to the family for you.

git add octofamily/octodog.txt

{

**Commit Etiquette:** You want to try to keep related changes together in separate commits. Using 'git diff' gives you a good overview of changes you have made and lets you add files or directories one at a time and commit them separately.

}

# 1.15 Staged Differences (cont'd)

Good, now go ahead and run git diff with the --staged option to see the changes you just staged. You should see that octodog.txt was created.

git diff –staged

{

**Commit Etiquette:** You want to try to keep related changes together in separate commits. Using 'git diff' gives you a good overview of changes you have made and lets you add files or directories one at a time and commit them separately.

}