To initialize a Git repository

Git init

Run the git status command again to see how the repository status has changed

git status

create text file and try git status, It will give information about current status

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

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.

git commit -m "Add story"

you also can use wildcards if you want to add many files of the same type.

git add '\*.txt'

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.

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

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

git log

To push our local repo to the GitHub server we'll need to add a remote repository.

git remote add origin <https://github.com/try-git/try_git.git>

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.

git push -u origin master

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

git pull origin master

looks like there have been some additions and changes to the octocat family. Let's take a look at what is differentfrom 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

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

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

So now that octodog is part of the family, octocat is all depressed. Since we love octocat more than octodog, we'll turn his frown around by removing octodog.txt.

You can unstage files by using the git reset command. Go ahead and remove octofamily/octodog.txt.

git reset octofamily/octodog.txts

Files can be changed back to how they were at the last commit by using the command: git checkout -- <target>. Go ahead and get rid of all the changes since the last commit for octocat.txt

git checkout -- octocat.txt

When developers are working on a feature or bug they'll often create a copy (aka. branch) of their code they can make separate commits to. Then when they're done they can merge this branch back into their main master branch.

We want to remove all these pesky octocats, so let's create a branch called clean\_up, where we'll do all the work

git branch clean\_up

Now if you type git branch you'll see two local branches: a main branch named master and your new branch named clean\_up.

You can switch branches using the git checkout <branch> command. Try it now to switch to the clean\_up branch:

git checkout clean\_up

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

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

git commit -m "Remove all the cats"

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

Alrighty, the moment has come when you have to merge your changes from the clean\_up branch into the masterbranch. 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

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

Git is a Distributed Version Control system (DVCS). It can track changes to a file and allows you to revert back to any particular change.