**GIT NOTES**

Type to initialize a git repo.

**git init**

Type to get git status

**git status**

Type to “track” items in the **Staging Area**.

**git add [FileName]**

Note: **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.

# Committing:

To store changes made to the repo itself:

**git commit -m** **"**Enter your comment here, such as adding new file or changed properties of x**"**

Using wildcard to add all of one type of files like so:

**git add '\*.txt'**

The git log. To access the git log, type the following.

**git log**

# Remote Repositories

This command takes a *remote name* and a *repository URL*, which in the test case is <https://github.com/try-git/try_git.git>.

So it would be typed as such:

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

# Pushing Remotely

The push command tells Git where to put our commits when we're ready, and now 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.

**git push -u origin master**

# 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**

# Differences

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**

You can unstage files by using the git reset command

**git reset octofamily/octodog.txt**

# Undo

git reset did a great job of unstaging octodog.txt, but you'll notice that he's still there. He's just not staged anymore. It would be great if we could go back to how things were before octodog came around and ruined the party.

Files can be changed back to how they were at the last commit by using the command:

**git checkout -- <target>**

**git checkout -- octocat.txt**

# Branching Out

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**

**Switching Branches:**

Tpye in the following to switch to a new branch

**git checkout [Name of Branch]**

**git checkout clean\_up**

**Switched to branch 'clean\_up'**

# Removing All The Things

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'**

# Commiting Branch Changes

**git commit -m "Remove all the cats"**

Remember, the words in the quotes are whatever you need to describe your action, such as “Added X variables”.

# Switching Back to master

**git checkout master**

# Preparing to Merge

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**

# Keeping Things Clean

You can use git branch -d <branch name> to delete a branch.

**git branch -d clean\_up**

# The Final Push

Push everything you've been working on to your remote repository.

**git push**

DONE!