Git Cheatsheet

Git is a distributed version control system (*DVCS*). A *DVCS* is just something that software developers use to keep track of, share, and discusses changes in the files that make up a project's source code.

What Git really keeps track of is the changes in a project's files in between user defined waypoints that are called *commits,* the changes could be new files and folders, the deletion of files/folders, or modified files that have insertions or deletions of text content. Applying a commit will make the necessary changes to take the files from their pre-commit state to their post-commit state.

The following is a quick summary of the most basic git commands you can use to track your projects:

`git init`

git init is the command to create a brand new git repository. In the terminal you can navigate to a directory that contains some code files (or any kind of files you want to keep track of) and type git init, and press enter.

`git clone`

git clone is the command you can use when instead of creating a repository you want to copy a repository that somebody has posted somewhere online like GitHub. When you use git clone it will create a new directory in the current directory in the terminal and put all of the project code tracked by git in that directory, here is an example:

```
git clone https://github.com/expressjs/express.git
```

This will clone the expressjs/express repository into a new directory called express. If instead you want to clone express into a different folder you can provide a name after the repo link:

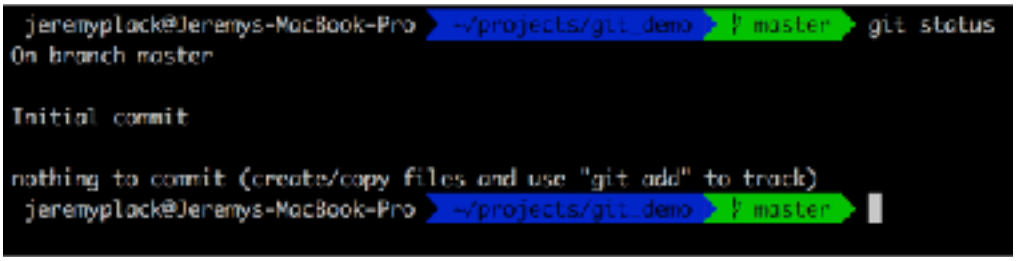
```
git clone https://github.com/expressjs/express.git my-express
```

This will clone into a new directory called my-express instead of express.

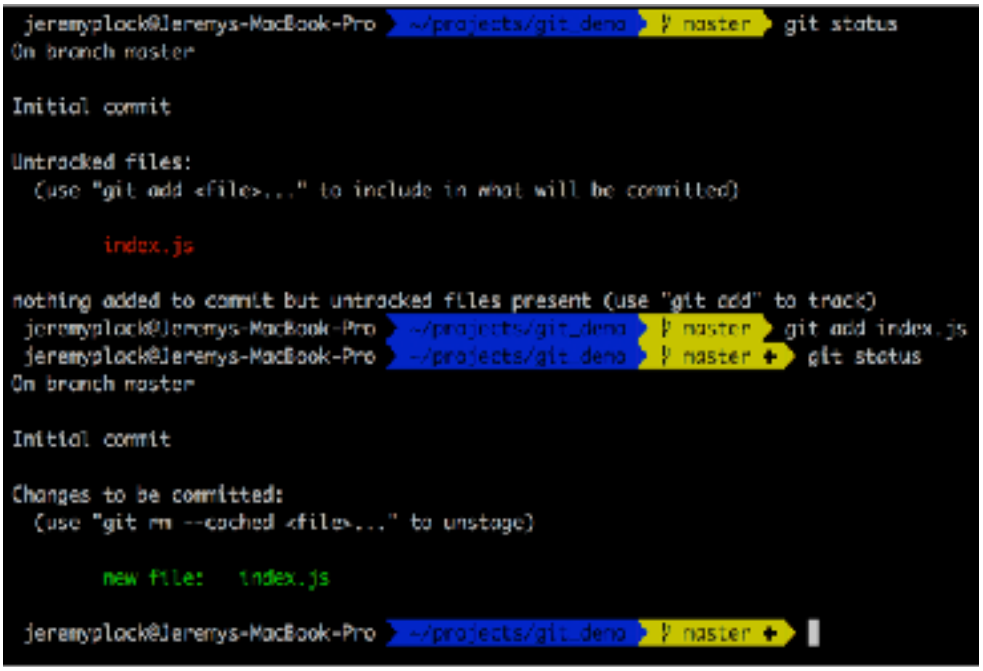
Git Cheatsheet

git status

git status is a command you can use in your git repository that will show all of the changes you've made since the last commit. Files that show up in red have changes that are not staged (ready) to be committed, you can either add them to the list of files to be committed or *checkout* the files, ***git checkout*** in this instance means to revert the file(s) to their state at the previous commit (before you modified them).
So in summary, ***git add*** is for adding the files to what will be committed, and *checkout* is for discarding the recent (uncommitted) changes in files that exist in prior commits, to for new files you can simply remove (*rm*) them.
If you stage a new (untracked) file with git add and then want to un-stage the file but not actually remove it you can use ***git rm —cached***



result of git status on a brand new repository.



result of git status after creating a new file and the after using git add

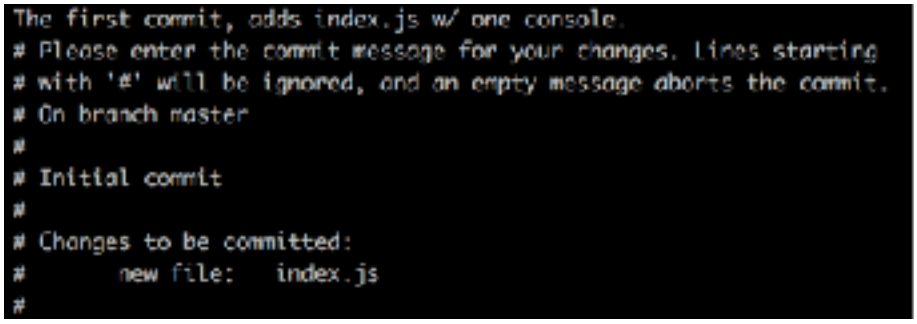
Git Cheatsheet

git add

As we saw earlier ***git add*** is used to add uncommitted changes to the list of things to be committed. This could be new files, or files that were previously committed but have new changes or have been deleted.
You can specify each file to be committed, if you have multiple files you can separate them by spaces. You can also use whats called a **regex** pattern to match multiple files, for example ***git add .*** will stage all of the files with changes.

git commit

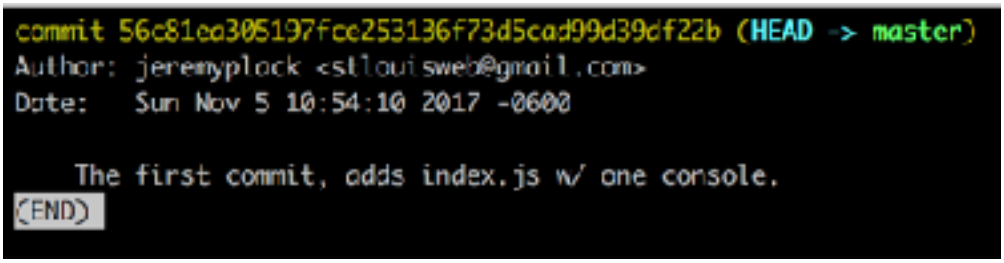
Once we've added files to be committed with ***git add*** we can use ***git commit*** to create a new commit with the changes. When you do this git will open up a text editor in your terminal called **VI** with a bunch of text in it. At the top of this text everything you type in will be saved as the commit message, so type a sentence to describe what the changes do. Below the top line are lines that show the difference between the old code and what is being committed so you can review this remind yourself of everything that has changed and fix something you might have missed.
If your happy with it type the letter *i* to enter **VI's** *insert* mode then type your commit message, then hit the *esc* key followed by *:wc* to exit insert mode and write quit, this will save the commit.



Git Cheatsheet

git log

git log is used to show a list of the commits that have been made in the current branch. To exit the log screen press *q*



git pull

git pull is a command you can use to pull down the latest changes from a remote location like GitHub. When you clone a repository git automatically creates a remote called “origin”, this is important to know because when you use ***git pull*** you need to supply a remote and a branch (more on branches later) like this:

```
git pull origin master
```

When your working on a project with multiple people you typically want to pull every time before you start working on the code, because it will be easier to modify the new code, then it is to find out you need to merge your set of changes with somebody else’s later.

git push

git push is the opposite of ***git pull***, it lets you push or upload your changes to a remote repository, so after you ***commit*** your changes you’ll want to push so that they live somewhere other than your local computer, and other users will be able to pull your changes so that they can review your changes and they can become part of the shared code repository.

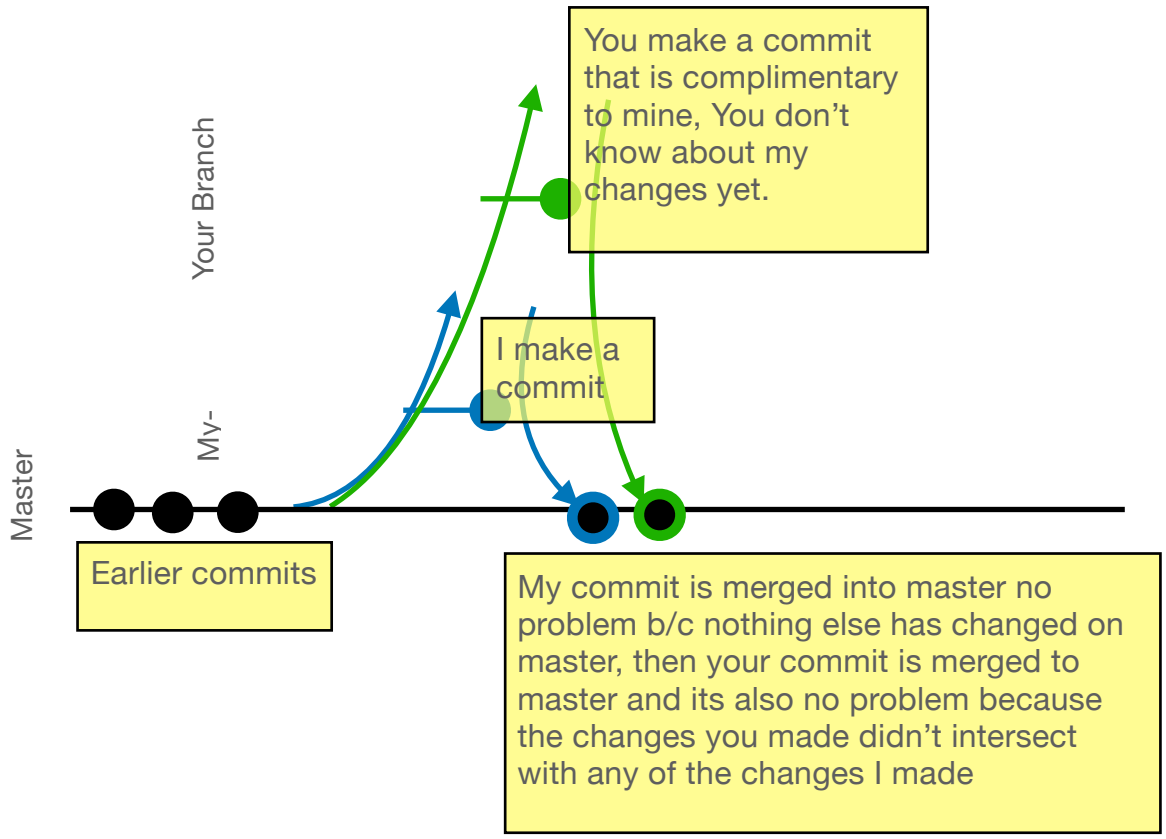
```
git push origin master
```

When your using push and pull GitHub would be considered an *upstream* remote. Don’t worry, if you try to push but forget to pull before hand git will handle integrating the changes properly or it will provide instructions for you to pull and merge manually.

Git Cheatsheet

git branch

Branches are Git's way of allowing for different versions of the same repository to evolve separately in parallel, and then potentially be merged back together if it makes sense to do so. Git repos start out with a main branch called master, you can add as many additional branches as you like at anytime, and you can delete or attempt to merge branches whenever you like as well.



This is the "happy path" git scenario, where team members can each implement separate features in parallel without running into each others code.
Bigger teams and projects that already have a lot of code will find this harder to achieve so team members will need to work together and use git to negotiate "merge conflicts" when two developers try to simultaneously modify the same lines of code.

`git branch` – lists all local branches

`git branch -a` – lists all local and remote branches

`git branch -d branch-name` – delete a branch

Git Cheatsheet

git checkout & git fetch

git checkout is a command that lets you checkout different points in the code's history called *refs,* **branches** are probably the easiest types of refs to work with for beginners, but tags, and the hashes (that long random looking string) on your commits are also refs.

`git checkout branch-name` –checkout an existing local branch

`git checkout -b branch-name` –create and checkout new local branch

`git fetch origin` –gets the refs for all of the remote branches so you can check them out with git checkout

You should typically ***git pull*** whenever you checkout a remote branch (or local branch which also has a remote branch)

Command	Description	Example
`git init`		
`git clone`		
`git status`		
`git add`		