Git Basic Terms & Concepts

To err is human. To really screw up requires git.

In the table below, git command always start with **git**. Concepts will typically start with some other word or term.

Command or Concept	Description
change set	Git doesn't work with files. It works with change sets
tracked vs. untracked files	Tracked files are those that git know about and under version control. Untracked files are files that git doesn't care about.
staging area	This is also know as **the index**. This area acts as a intermediate state between the working directory and the commit. This is where a tracked file is placed after changes have been made and a **git add [file name]** has been executed.
working directory	The location of your git project. This is your local directory where your project files live.
commit hash	This is 40 digit hex value that identifies a commit. Usually only need to give git 4 or 5 digits to specify given commit.
HEAD pointer	This is the pointer that points to the most recent commit (or tip) of a branch.
git init	Initializes a repository (aka repo) from where ever this command is executed.
git status	Shows the contents of the staging area as well as lists the contents of untracked files
git add *[file name]*	This does two different things depending on the context. If the file was previously untracked, this command tells git to start tracking the file. If the file is being tracked
git commit -m *[commit message]*	Commits the files that you have staged (i.e. added to your index). Good commit messages make it easier for you navigate history.
git log	This command has a lot of options. For example, my favorite is: git log --oneline -5 which lists a summary of the last 5 commits on a single line. Feel free to explore.
git checkout *[branch name]*	Important that your working dir is clean before running this command. Use the -b option to create a new branch and navigate into it
git checkout -b *[branch name]*	Create a new branch from the existing branch you executed this command from. As with a regular checkout, make sure your working dir is clean before running this command.
git branch	lists the branches in your project repo.
git branch -d *[branch name]* git branch -D *[branch name]*	Deletes a branch. Using **-d** tell git to remove the branch only if the working directory is clean. Using the **-D** tells git to remove the branch regardless of whether there are uncommited changes or not.
git merge *[branch name]*	The branch you are on when you execute this command is considered the **receiver**. This is the branch that will receive the changes from the merge with *[branch name]*. This command will generally result in one of 3 outcomes: 1) a fast-forward merge, 2) a

	recursive merge, or 3) a merge CONFLICT
git merge --abort	This is command comes in handy if you do a merge and run into a conflict which you just don't want to resolve at this time. Running this command will put you back into the state you were in before you attempted your merge.
git remote	Lists the remotes the local git project knows about. Using -v option lists urls to push and fetch (which are usually the same). Remotes tell your local git installation where to push remote changes.
git remote add *[remote name]* *[url to remote]*	In order to get your local changes up to github, you need need to tell git where to send those changes. A **Remote** is a pointer which tells git where to push changes you made locally. For example, this is the command I use to create a remote for a new project I want to share on github: git remote add origin https://github.com/MichaelSzczepaniak/*[project name]* Once you've created a remote, you can **push** changes to it.
git push -u origin *[branch name e.g. **master**]*	This is typically the version of the push command you use when making the first push of your project. The **-u** option sets the upstream location for future pushes done from this branch. This essentially tells git to remember where push future changes so you don't have to type this full command in the future.
git push	Once you've told git to remember the upstream location (as described in the previous command), you don't need to type the full *git push -u origin [branch name]* command. You just need to do *git push*
git clone *[project url] [alias]*	Clones git project locally. If **[alias]** is omitted, the project name is used by default.
clone vs. fork	Greg mentioned this in his talk. Do you recall this distinction?
git fetch origin *[remote branch]*	Fetches the latest version of the remote and saves it into the special ***origin/[remote branch name]***. NOTE: You still need to do a merge to get this into your local branch. **git pull** combines **git fetch** and **git merge** together for convenience, but I **highly discourage** using this command until you've been using git for awhile and really have a feel for what this is doing. See graphic: 200 state immediately after a fetch (need to merge to master).jpg
git reset --soft *[hash]* git reset --mixed *[hash]* git reset --hard *[hash]*	After this workshop, you should be able to understand what this write up is talking about: https://gist.github.com/tnguyen14/0827ae6eefdff39e452b

Two ways to work with a Github remote:

1. Create remote first then, clone locally
    - Easiest way to start a **new project** you know you'll want to have on Github.
    - Here's how:
        1. Navigate the project on github (e.g. https://github.com/MichaelSzczepaniak/PredictNextKBO)
        2. Click the **Clone or download** button and copy the url
        3. Run the **git clone** command described above using the url you just copied as the *[project url]* parameter
2. Create local repo first, create remote, push local to remote
    - Good way to get an **existing project** out to Github.
    - Here's how:
        1. Use the **git init** command to initialize a project locally
        2. Create some files and/or make some changes
        3. Make a few commits to your local repo using the **git add** and **git commit** commands
        4. Log into your github account. If don't have an account, create one.
        5. Click the **Repositories** link near the top of the page
        6. Click the green **New** button
        7. Fill out the form with the repo name and description
        8. Click the **Public** radio button (unless you want to pay for a private repo...)
        9. Click the **Create repository** button. You should now have an empty repo to push your work to
        10. Note information git provides you when you create your project. You'll need this information to create your remote that will enable you push your work up to this remote repo.
        11. Back on your local system, create a remote from within you local project using the **git remote add origin *[url to remote]*** command described above.
        12. Run the **git push -u origin master** command described above to push your local project to the remote. You'll only need **git push** for future pushes since set the upstream parameter as described above.