General Git

- Keeps track of history, allows for re-instating a previous version and for finding out who made the alteration
- Don't need to ever make a final copy of code called finalcode.cpp
- Don't need to comment out code to save it for later if it is important
- Branching and allowing development of different functionalities concurrently
- Git is distributed, getting rid of the need for a central potentially corruptible central server
    - Each person has their own copy of the full repository, and has access to the full history of commits
    - Don't need internet connection to make commits, or read previous version, (also doesn't take up much memory)
    - Github acts more as a central repo
- Is meant for agile development

Workspace

- Local directory
- Locally modified

Index (stage)

- Will be considered in the next commit

Local Repository (HEAD)

- Committed items

Remote Repository

- Known as remote, can refer to a remote repo on your device (remote) or a satellite server (upstream) for others to have access

Initialize

In order to make a repository recognized by git, a “git repository” able to be staged and committed, you must initialize the repository

- Use **git init** while in the directory

git clone __url__

- Copies remote into local workspace
- Establishes the remote repository as the “origin”
- Is used once in the beginning to initiate the structure of the project
    - git pull and git merge are used to update, this is for initialization

Forward Commands

git add

- Adds files to index
- “git add .” adds all files

git commit

- Everything in index goes to the local repository
- Requires a message along with the commit (git commit -m “kasjdfpoijapofijw”)

git commit -a

- Stage and commit all files at once

git push __remote repo__ __current branch__

- Local repository to the remote
- git push origin :remote_branch_name
    - Will delete the remote branch (command does replace :____ with ____:)

Backwards Commands

git fetch origin

- Brings remote repo items to your local repository
- Won't change workspace (allows for comparison and the decision to take them or not)
- Git fetch origin branch_name:branch_name
    - Will grab the remote branch_name contents and put it in the local branch_name

git merge ___branch to be merged___

- Joins branches together
- Merges branch to be merged into branch you are currently on
    - Switch to master, then merge into the master
    - Fast forward merge: move head to the latest command present in the branch to be merged
    - If the master has already progressed beyond the state from which the branch started, then a three way merge will occur, i. e. a new commit will be created containing all of the changes/ info on master from the branch to be merged
        - See rebase
- git mergetool provides a visual environment to fix conflicts
- git merge master origin/master

git pull

- Brings remote repository stuff to your workspace directly
    - Combines a git fetch and a git merge
    - If there are new files it will merge them and preserve new work (won't just make your repo look like the remote repo)

Rebasing

- If you want to merge but don’t want the hassle of a new commit due to three way merging, a rebase can be performed which will change the starting point of the branch, merging in the changes and then using a fast forward merge on the new branch
- Git pull --rebase origin master
    - Rebase processes commit by commit, this will modularize the process making it less dangerous and easier to catch problems
- If there is a conflict, resolve the issue locally, then git add <file>, git rebase --continue
- If there is a huge mistake: git rebase --abort

Branches

- Do some potentially dangerous changes without affecting the original “master” branch

git branch

- Displays info about the current branches
- Asterisk indicates which branch is the current branch
- git branch -r, displays info about the remote branches

git branch __new branch name__

- Creates a new branch with exact copy of code you are currently on

git checkout __branch__

- Will switch to the other branch

git checkout -b __new branch name__

- Creates and switches to a new branch

git merge __branch you want to merge with the current one__

- Combines all files, attempts to resolve conflicts
- Generally you will be on master and merge the new changed branch with the master
    - In this case the new branch is the provider branch
    - Master is the receiver

git branch -d __branch you want to delete__

- Deletes the branch

If there is a conflict between the two branches that you want to merge, the auto merging fails and the conflict is displayed to the user to be fixed manually. Delete all of git’s special markings when changing the document

vim __conflicting file__

- Displays information about what specifically is conflicting in the file
- Separator displays the differences between the first and the second instance of the file

Whoops

Head is the most recent commit

Restore the file in your working directory to the state of the last committed file in the local repo

- git checkout HEAD __filename__
    - See checkout section for information

Unstage a file from the staging area

- git reset HEAD filename
    - Resets file in stage to be the same state as the last head commit
    - Does not discard changes from the working directory

Undo the commit but maintain the changes that I have made

- git reset --soft HEAD^

Undo the commit and revert changes to the file

- git reset --hard HEAD^

Reset the state of the project to a previous version present in the git log

- git reset SHA(first 7 chars)
    - Use git log to get SHA
    - Will reset to that state and you will lose all progress afterwards

Other

git diff master origin/master

- Difference between local and remote master
- Can be used after fetch

git diff HEAD works to display changes on the workspace and comparing it to the HEAD

- What are the differences between local and workspace?
- I will decide to take local changes or not, potentially commit my own work instead
- + lines added
- - lines taken away

git diff __filename__

- Difference between workspace and index
- Used to check what will be committed if you commit

git help __command__

- Provides man page for the command

git status

- Displays info about the directory, changes that have been made etc.
- Displays info about what's been happening with the git add and commit commands

    - Which files are currently unstaged

git log

- Shows the commit history of the project
- Each commit displayed with a commit identifier
- Author and message shown
- git log -p
    - Provides a more detailed view of commits
- git log --graph --decorate --oneline
    - Most aesthetic presentation, easiest to read
- ~(tilde)symbol is used to make a relative reference to the commit
    - ~1=back one
    - ~3=back three

git show __first few letters of id shown from git log__

- Displays the differences and more info in the specific commit

Git is integrated into eclipse, and is called egit. It displays a visual vcs system. Go to git perspective and select a repo to use it.

A .gitignore file will automatically exclude from the repository a specific type of file. (ex. Derived files)

- This is useful when uploading many files at once

Example setup:
Create a git repo
Create a directory in your file system
git init the directory
git remote add origin __url from github__
Create and edit a file
git add
git commit
git push origin master
Origin is known as the remote name now
Master is the local branch

SSH can be used to not have to enter the username and password each time
Generate a public and private key using “ssh-keygen -t rsa -C "jbooth2015@yahoo.com"”
Copy the public key and add it to the list of SSH keys on github
When adding the repo into your directory, use the command “ git remote add origin git@github.com:jaybooth4/Test-repository.git”

Checkout

- Used for checking out files, commits, and branches

Checkout a branch

- git checkout <branch>

Checkout a commit makes the working directory match the commit

- This will move the “HEAD” of the project to the previous commit (detached HEAD state)
- git checkout <commit>
- This is a read only operation
- git checkout master   to return to most recent

Checkout a file, replaces file in working directory as well as staging area

- git checkout <commit> <file>
    - This will not change the position of the head

Revert

If in a commit you delete a function, but then several commits later you realize that you need that function, but also don't want to lose the progress that you've made over the commits in between, you can revert a specific commit. This means that anything added in that commit compared to the commit before it will be combined and added/deleted (merged) with the most recent commit to make a new commit. Git checkout can be used to mimic this behavior, ex. git checkout SHA file, bringing the file to the current directory. The difference is that the revert will make a new commit, in which you can specify what you reverted and why.

- git revert SHA

Reset

This will move the head of the project to a specific commit and destroy any subsequent commits. Be very careful while using this. Use revert for a specific and small change, compared to reset for a large experimental change that resulted in unfixable errors. Only use git reset on a local experiment gone wrong, never a public change.

- git reset HEAD~2
- git reset SHA
- Flags:
    - Soft: makes the head point to the commit, working and index stays the same
    - Mixed: makes the head point to the commit, index also changes, working stays
    - Hard: all reset
- If you staged a file and want to unstage it, use git reset __filename__

Amend

If you want to amend the previous commit without creating a new commit, this command will merge the index with the commit and replace the current with a new merged one

- Git commit --amend

Create a shortcut for a git command

- git config --global alias.<alias-name> <git-command>

Clean up (delete) all untracked files

- git clean

Rebase
- It is possible to rebase a branch in the commit history

Reflog
- With every change in position of HEAD the reflog is updated
- git reflog

Workflow
Centralized
- Clone from git first, make edits which are saved locally
- Merge/rebase locally, then push to the remote
    - See rebase
- Example without rebasing:
    - git checkout master
    - git pull
    - git pull origin __branch__
    - git push
        - Move to master, update master, merge with remote branch, push master

Feature branch workflow
- Create branches for each feature in the project
- Create a local branch
- Push local branch to central and initiate a pull request
    - If you have developed a new component and are requesting others to pull it into their local repositories to look it over, establish a pull request on github (opens a forum for review and discussion about the code)
    - Git push -u origin ___branch-name___
        - -u establishes it as a tracking branch, meaning that it will automatically push to this branch if you run the command git push

Interact with branches and a remote repo
- https://www.youtube.com/watch?v=Luo-xKWD6aw

Forking
- Create a branch on your own github account, call this origin
- Pull frequently from upstream (where you forked from)
- Submit pull requests to upstream when you are ready
- http://www.dataschool.io/simple-guide-to-forks-in-github-and-git/
- git remote -v displays current settings about origin and upstream
- With forking, git pull from upstream, push to origin

UPDATE A BRANCH

```
git fetch upstream
git checkout __branch__
git merge upstream/__branch__
```