Git Reference:

Local Repo	
git init	Converts a local directory into a Git Repository
git status	See what is being tracked by Git within a Repository
git add . [file] [file] [file] *.txt	• Add everything within the directory to the repo • Add specific files to the repo • Add All txt files to the repo
git commit	A snapshot of all the files in the directory (repo)
git commit -m "<message>"	Write a message that describes the commit
git commit --amend	Alter the latest commit
git diff --staged <commit> <commit>^! <from commit> <to commit>	• See the changes within the working directory • See the changes in the Staged Area • See the changes within a specific commit • See the changes between a specific commit and the commit before it • See the changes between 2 specific commits
git branch …	Creates a parallel commit
git checkout …	Changes where HEAD is
git checkout -b …	Creates a parallel commits and moves HEAD there
git merge …	Merges parallel branches
git rebase …	• Takes a set of commits, copies them, and puts them down somewhere else • Keeps a cleaner commit log / history of the repository
git log	Used to see hashes
^	Move upwards one commit (per ^)

~<num>	Move upwards a number of times (don't use <>)
git branch -f master HEAD~3	• Directly reassign a branch to a commit • (moves the master brand to three parents behind HEAD)
git reset …	• Reverse changes by moving a branch reference backward to an older commit • Used for local git repos
git revert …	• Reverse changes and *share* those reversed changes with others • Used for remote git repos
git cherry-pick <commit> <commit> <commit>	Copy a series of commits below your HEAD
git cherry-pick <source>..<destination>	Copy a specific commit source to a specific branch destination
git rebase -i	• Open up a UI to show which commits are about to be copied below the target • Shows their commit hashes and messages • Ability to do 3 things: • Reorder commits by changing their order in the UI • Choose to completely omit some commits ○ toggling pick off means you want to drop the commit • Squash commits… essentially allows you to combine commits ○ (this aspect has been glossed over)
git tag v1 c1	• Permanently mark "milestones" that can be referenced like a branch • CANNOT change what is in the tagged commit • Tags exist as anchors in the commit tree that designate certain spots • Tags commit 'c1' as the 'version 1' prototype • if the commit is left off the command… then the Tags is placed on the HEAD
git describe <ref>	• Where you are relative to the closest "anchor" (aka Tag) • Useful when you need to get your bearings after moving too much within a commit tree • <ref> can be any commit • If <ref> is left out of the command, the default is HEAD • Output command: • <tag>_<numCommits>_g<hash> ○ <tag>: Closest anchor in history ○ <numCommits>: How many commits away that anchor is ○ <hash>: The commit being described

git checkout HEAD~^2~2	• ^<num> will choose a different branch when trying to move upwards from a merged commit. • No number will move upwards directly above the merge. • ^<num> will move up the next closest branch • ~<num> moves up from the lowest commit within that branch

REMOTE REPO

git clone	Create *local* copies of remote repositories
git fetch	• Downloads the commits that the remote has but are missing from our local repository • updates where our remote branches point (for instance… o/master) • Does Not: • Change anything about the local state • Update the master branch or change anything about how your file system looks right now
git pull	*Fetching* remote changes and then *merging* them at once
git push	Uploading local changes to a specified remote, updating that remote to incorporate new commits
git pull --rebase Or git pull -r	• *Fetch* and *Rebase* • Pulls from remote and then rebases the current commit below • Changes we have made locally appear as if they were based on the latest version available from the remote repo
git checkout -b totallyNotMaster o/master	Creates a new branch named totallyNotMaster and sets it to track origin/master
git branch -u o/master foo	• Set remote tracking on a branch • Set the foo branch to track o/master • if foo is currently checked out, you can leave it off • git branch -u o/master

git push <remote> <place>	• Tells git where the commits will *come from* and where the commits *will go* • By specifying both arguments, it *ignores where we are checked out* • <place> is essentially the "place" or "location" to synchronize between the two repositories • Where it comes from • <remote> is the remote repo name
git push <remote> <source>:<destination>	• <remote> is the remote repo name • <source> is the local commit we are pulling from • <destination> is the remote commit we are pushing to • If the destination doesn't exist, this command will create a new branch on the remote repo
git fetch <remote> <place> *Can run 'pull' the same way*	• <remote> is the remote repo name • <place> Grab all the commits from the remote branch and copies all commits that aren't present locally
git fetch <remote> <source>:<destination> *Can run 'pull' the same way*	• <remote> is the remote repo name • <source> is the local commit we are pulling from • <destination> is the remote commit we are pushing to • If the destination doesn't exist, this command will create a new branch on the local repo • This command does NOT typically get used… because you MUST be sure
git push <remote> :<destination>	• If there is no source, the <destination> gets deleted on the remote
git fetch <remote> :<destination>	• Creates a new branch locally
git stash	• A stack of changes on which you store any changes to the *Working Directory* • Place any modifications to the *Working Directory* on the stash
git stash pop	• Take the latest change in the stash and apply it to the *Working Directory* • Remove the latest stashed change before applying it again • I need further clarification on what's happening with this command
git stash apply	• Do not remove latest changes before applying them • Seek more clarification on what is happening with pop and apply modifiers
git stash list	• Inspect the current stash and list individual entries
git stash show	• Show the changes in the latest entry on the stash

git stash branch <branch name>	• Create a branch, starting from the HEAD and apply the stashed changes to that branch

Initial Setup

- git config --global user.name "Your Name"
- git config --global user.email "email@example.com"

Clone a repo off the remote server

- git clone git@example.com:user/repo local-repo-name

Create a new repo locally

- git init
- git add .
- git status
- git commit -v -m "Initial commit."

Connect your repo to a remote repo

- git remote add origin git@example.com:user/repo

Normal Git workflow

1. Add and edit file(s)
2. Check the status of changes using git status or git status -s for a simpler version
3. Check exactly what you've changed within the files with git diff or git diff --stat for a simpler version
4. Add the changes you want to commit to the "staging area" with git add
5. Double-check your stages changes with git status
6. Commit the staged changes to your local repo with git commit -m "message"

Change default strategy for *pull* to use *rebase* instead of *merge*

- git config --global pull.rebase true