Shell / Command Line

- echo: used like “print” in other languages
- ls: list files in directory
- ls -l: list files in directory with more information (l for long)
- ls -a: shows all files (including hidden ones)
- cd: move to a different directory
- cd ..: move one directory up
- cd -: return to previous directory
- cd ~: return to main working directory
- pwd: print working directory
- mkdir: creates a directory
- touch: create a file
    - touch test.txt: creates a file called test.txt
- mv: move files from a directory to another
- curl: used in command lines or scripts to transfer data (downloading)
- rm: delete files
- rmdir: delete directories
- grep: “global regular expression print” (a form of regex), processes text line by line and prints any lines which match a specified pattern
    - grep shell dictionary.txt | less

Version Control

- Commit: Saves a project’s files at a specific point in time
- Repository / Repo: directory which contains your project’s work, as well as a few hidden files (made up of commits)
- Working Directory: files seen in the computer’s file system
- Checkout: when content in the repository has been copied to the Working Directory
- Staging Area / Staging Index / Index: file in the Git directory that stores information about what will go into your next commit. Files on the Staging Index are poised to be added to the repository
- SHA: an ID number for each commit. 40 characters string composed of charters (0-9 and a-f)
- Branch: a new line of development is created that diverges from the main lien of development

Git

- git config –list –show-origin: displays git conifigurations
- git init: create a git repository inside the current working directory
- git clone: clones an existing repository, specified by a path (usually a URL)
    - git clone https://...
- git status: shows vital information, should be ran often
- git log: shows history and basic information of commits (type q to quit out)
- git log –oneline: shows history, but with shortened SHA and just commit message
- git log –stat: builds on git log with the additional information of changed files and number of added/removed files

- git log -p: will display files that have been modified, location of the lines that have been added/removed, and displays the actual changes that have been made (p for patch)
- git show: typically followed by a SHA (or shortened SHA), and will show just that one commit. Default show is "git show -p", but can be combined with other flags such as:
    - –stat: to show how many files were changed and the number of lines that were added / removed
    - -p: this is the default, but can be combined with –stat to show stat + patch information (git show SHA –stat -p)
    - -w: ignore changes to whitespace
- git add: moves files from the Working Directory to the Staging Index
    - git add file
        - for multiple:
        - git add file1 file2
        - git add folder1/file1 folder2/file2
        - SPECIAL CHARACTER: period: git add .
- git commit: commits the files located in the Staging Index
    - git commit –'m "type (short) commit message here": use for shorter messages to bypass opening of code editor
- git diff: see what changes have been made but not yet committed
- .gitignore: note the dot in front, THIS IS NOT A COMMAND, but placed in the same directory the hidden .git folder is in if you would like certain files in the directory to not be committed
    - Add files not to be committed to the .ignore file
    - globbing: using regex concepts in conjunction with .gitignore if you need to ignore numerous of the same type of file
        - blank lines can be used for spacing
        - #: marks line as a comment
        - *: matches 0 or more characters (any number of characters, including none)
            - Law* = Law, Laws, Lawyer
            - *Law* = Law, GrokLaw, Lawyer
        - ?: matches 1 character (any single character)
            - ?at = Cat, cat, Bat, bat
        - [abc]: matches a, b, or c (matches one character in bracket)
            - [CB]at = Cat, Bat
        - **: matches nested directories – a/**/z matches
            - a/z
            - a/b/z
            - a/b/c/z
        - Example: 50 JPEG images in the sample folder we don't want to commit
            - Samples/*.jpg
- git tag: draw attention to specific commits
    - git tag -a v1.0: creates an annotated flag labeled v1.0 (without -a, the tag becomes a lightweight tag)
    - typing just "git tag" will display all tags within the repository
    - git tag -d v1.0: deletes the specified tag

    - git tag -a v1.0 SHA: will create a tag for the commit with the specified SHA
- git branch: list all branch names in repository, create new branches, delete branches
    - git branch sidebar: creates a branch named sidebar
    - git branch sidebar SHA: creates a branch named sidebar at commit SHA
    - git branch -d sidebar: deletes branch sidebar
    - git branch -D sidebar: force deletion (Git won't let you delete a branch with new commits)
- git checkout: used to switch between branches
    - git checkout sidebar: switches to the sidebar branch
    - IMPORTANT NOTES:
        - Checkout will remove all files and directories from the Working Directory that Git is tracking
        - Checkout will go into the repository and pull out all of the files and directories of the commit that the branch points to
    - Can actually create branch and switch to it all at once: git checkout -b newbranch. Taking it further, you can have it branch at a specific branch, i.e. git checkout -b newbranch specbranch
- git log –oneline –graph –all: shows log of all commits from all branches
- git reset –hard HEAD^: use this command to undo an erroneous merge
- git merge: combine Git branches. Merging follows this process:
    - look at the branches its going to merge
    - look back along the branch's history to find a single commit that both branches have in their commit history
    - combine the lines of code that were changed on the separate branches together
    - makes a commit to record the merge
    - git merge <name-of-branch-to-merge-in>
    - Fast Forward Merge: branch being merged in must be ahead of the checked out branch, and the checked out branch's pointer will just be moved forward
    - Regular Type Merge: two divergent branches are combined, and a merge commit is created
- Merge Conflict Indicators:
    - <<<<<<< HEAD: everything below this line (until next indicator) shows you what's on the current branch
    - ||||||| merged common ancestors: everything below this line (until next indicator) shows you what the original lines were
    - =======: is the end of the original lines, everything that follows (until the next indicator) is what's on the branch that's being merged in
    - >>>>>>> heading-update: is the ending indicator of what's on the branch that's being merged in
- git commit –amend: alter the most recent commit. With a clean Working Directory, you can alter the commit and then resave as normal. Furthermore, you can add forgotten files to the commit:
    - edit the file(s)
    - save the file(s)

    - stage the file(s)
    - and run git commit –amend
- git revert: takes the changes made in the specified commit (git revert SHA), and does the exact opposite of them. This command makes a new commit itself
- Relative Commit References: notation used when navigating through parent commits in a commit graph:
    - ^: indicates parent commit
    - ~: indicates first parent commit
    - The parent commit:
        - HEAD^
        - HEAD-
        - HEAD^
    - The grandparent commit:
        - HEAD^^
        - HEAD-2
    - The great-grandparent commit:
        - HEAD^^^
        - HEAD-3
    - Merging with parents:
        - ^: first parent (branch checked out during merge)
        - ^2: second parent (branch specified to be merged in)
- git reset: this will reset the progress to a specified parent notation. What happens to the cut portion relies on the flag that is added:
    - git reset –mixed HEAD^: resets to the parent commit, places most recent commit in the Working Directory
    - git reset –soft HEAD^: resets to the parent commit, places most recent commit in the Staging Index
    - git reset –hard HEAD^: resets to the parent commit, places most recent commit in the Trash
- git branch backup: before any resetting, it is wise to create a backup branch

GitHub & Remotes

- Remote Repository: similar to a local repository but it exists elsewhere. Access via:
    - URL (by far the most common)
    - File path
- Git vs. GitHub: Git is a version control tool, whereas GitHub is a service to host Git projects
- git remote: manage and interact with remote repositories
    - will return blank if no remote repository is configured
    - cloned repositories will automatically have a remote
    - origin: shortname and defacto name to refer to the main remote repository (easier than the entire URL / file path)
        - git remote -v: will show the full path
    - git remote add origin URL: after creating a remote repository in GitHub, use this command to link your local repository to GitHub's remote repository

        - Note: origin is arbitrary, can be named whatever
        - The template command is “git remote add *shortname URL*”
    - git remote rename <original> <rename>
- git push: send local commits to a remote repository
    - git push <remote-shortname> <branch>
- git pull: syncs local repository with remote repository (usually a fast forward type)
    - git pull <remote-shortname> <branch>
- git fetch: retrieves commits from the remote repository, but does not automatically merge them (essentially retrieves the commits and places them in a new branch)
- fork: not a command, this a feature in GitHub (and other version control hosts) which allows a copy of an existing remote repository to be made, in which the user is now the owner
    - Note that forking and different from cloning. Cloning creates a copy to a local repository, while forking keeps the repository in a remote repository
- git shortlog: displays alphabetical list of names (of collaborators) and the commit messages that go along with them
- git shortlog -s -n: shows number of commits each collaborator has made
- git log –author=<author>: shows git log for a specific author
- git rebase: (be careful with this command) used for “squashing,” or combining commits together
    - git rebase -i HEAD~x: will combine all commits back to x
    - note that this is a good time to create the backup branch
    - commands used in conjunction with git rebase:
        - p or pick: keep commit as is (default)
        - r or reword: keep commit’s content but alter the commit message
        - e or edit: keep this commit’s content but stop before committing so that you can:
            - add new content or files
            - remove content or files
            - alter the content that was going to committed
        - s or squash: combine this commit’s changes into the previous commit (commit above it on the list)
        - f or fixup: combine this commit’s changes into the previous commit (commit above it in the list)
        - x or exec: run a shell command
        - d or drop: delete ethe commit
- git push -f: used to force push commits (likely will be needed if git rebase is used)