Git book: https://git-scm.com/book/en/v2

Basic Git https://rogerdudler.github.io/git-guide/

Git workflow: https://gist.github.com/blackfalcon/8428401

Tutorials: https://www.atlassian.com/git/tutorials/source-code-management

Basic unix command: https://www.tipsandtricks-hq.com/basic-unix-commands-list-366

What is version control:

A category of software tools that help a software team manage changes to source code over time. Version control software keeps track of every modification to the code.

Why do we need version control?

- Loss file or changes
- Recover from older version
- Compare differences
- Check log, who did what
- Different Releases

Source code management best practices:

- Commit often
- Make sure to work on latest file
- Detailed notes
- Review change before committing
- Use branches
- Agree on a workflow

Git

- Open source
- Developed in 2005 by Linus Torvalds(yes, the creator of the Linux operating system kernel)
- Distributed Version Control System(Rather than have only one single place for the full version history of the software as is common in once-popular version control systems like CVS or Subversion (also known as SVN). In Git, every developer's working copy of the code is also a repository that can contain the full history of all changes.)
- performance, security and flexibility
- hard to learn

cheat sheet:

https://www.atlassian.com/git/tutorials/atlassian-git-cheatsheet

Setting Up a GitHub Account

Creating a Repository

Creating a Repository in a Local Environment

Initializing a repository locally entails using git init and mapping the local repository to its corresponding remote repository:

1. Create a directory for the application using the following command: **mkdir TestGit**:
2. Change the working directory to the project directory using the following command: **cd TestGit**
3. Initialize the repository using the following command: **git init**.(This creates a new subdirectory named `.git` that contains all of your necessary repository files)

Recording changes

Remember that each file in your working directory can be in one of two states: *tracked* or *untracked*. Tracked files are files that were in the last snapshot; they can be unmodified, modified, or staged. In short, tracked files are files that Git knows about.

1. Check status: git status
2. Create new file: echo Change me > change-me; echo Delete me > delete-me
3. Check again: git status
4. Track the file: git add change-me delete-me (then check status, now both are tracked and staged)
5. echo Change again >> change-me: >> append instead of override
6. git status: now listed as both staged and unstaged. If you commit now, will use the version last ran "git add". If you modify a file after you run git add, you have to run git add again to stage the latest version of the file.
7. git diff: That command compares what is in your working directory with what is in your staging area. The result tells you the changes you've made that you haven't yet staged. (`git difftool --tool-help to see what is available on your system`)
8. git diff --staged: This command compares your staged changes to your last commit
9. git commit: open vim, vim cheat sheet https://vim.rtorr.com/
10. or git commit -m initial
11. git commit –a will skip staging. BE CAREFUL
12. git rm: remove from staging area, no longer tracked
13. git mv file1 file2: equivalent to mv then git rm git add

Optional:

Difference between add . add –U and add -A

(https://stackoverflow.com/questions/572549/difference-between-git-add-a-and-git-add)

- git add . stages new files and modifications, without deletions
- git add -u stages modifications and deletions, without new files
- git add -A stages all changes (doing both add . and add -u)

1. echo OK>>change-me
2. rm delete-me
3. echo Add me > add-me
4. git status
5. git add . then check status, then git reset
6. git add –U then check status, then git reset
7. git add –A then check status

short status: git status –s

- ??: aren't tracked
- two columns: left-staging area; right-working tree
- A: new
- M: modified

Ignore: https://git-scm.com/book/en/v2/Git-Basics-Recording-Changes-to-the-Repository

you can create a file listing patterns to match them named `.gitignore`

An asterisk (`*`) matches zero or more characters; `[abc]` matches any character inside the brackets (in this case a, b, or c); a question mark (`?`) matches a single character; and brackets enclosing characters separated by a hyphen (`[0-9]`) matches any character between them (in this case 0 through 9).

```
# ignore all .a files
*.a
# but do track lib.a, even though you're ignoring .a files above
!lib.a
# only ignore the TODO file in the current directory, not subdir/TODO
/TODO
# ignore all files in any directory named build
build/
# ignore doc/notes.txt, but not doc/server/arch.txt
doc/*.txt
# ignore all .pdf files in the doc/ directory and any of its subdirectories
doc/**/*.pdf
```

Tagging: https://www.atlassian.com/git/tutorials/inspecting-a-repository/git-tag

Viewing commit

1. git log
2. git log –p -2:
   `-p` or `-patch`: show the difference (the *patch* output) introduced in each commit.
   `-2`: show only the last two entries.

Undoing things

1. git reset <file>: Remove the specified file from the staging area, but leave the working directory unchanged.
2. git reset: Reset the staging area to match the most recent commit, but leave the working directory unchanged. This unstages all files without overwriting any changes.
3. git reset --hard: Reset the staging area and the working directory to match the most recent commit. In addition to unstaging changes, the `--hard` flag tells Git to overwrite all changes in

the working directory, too.

Cloning an Existing Repository

Instead of getting just a working copy, Git receives a full copy of nearly all data that the server has. When you first clone a repository, all of your files will be tracked and unmodified because Git just checked them out and you haven't edited anything.

1. Back to jswanson: cd ..
2. git clone https://github.com/jswanson79/GitDemo.git (That creates a directory named GitDemo, initializes a .git directory inside it, pulls down all the data for that repository, and checks out a working copy of the latest version.)
    - If you want to clone the repository into a directory named something other than GitDemo, you can specify the new directory name as an additional argument:
      git clone https://github.com/jswanson79/GitDemo.git myGitDemo
    - git clone --branch <tag> <repo>: Clone the repository located at <repo> and only clone the ref for <tag>.
    - git clone -depth=1 <repo>: a clone of <repo> is made and only the most recent commit is included in the new cloned Repo.

3. cd GitDemo check status
4. git remote:
5. git remote: List the remote connections you have to other repositories.
   git remote –v: also list the URL
6. git remote add java https://github.com/jswanson79/JavaFiles.git: add a new remote Git repository as a shortname(java) you can reference easily
7. git checkout master: make sure you are on master
8. git pull origin master: all up to date with the remote repo. Pull = fetch + merge
9. create new branch: git checkout -b jswanson-branch(replace with your netID) : `create a new branch from master as well checkout out that new branch at the same time`
10. git branch: `list out the branches in my local repo and place a * before the branch that is checked out.`
11. `Create a file in your branch called netID.txt, edit it, then commit`

Merging code:

1. Make sure that I have the latest version of the feature branch from the remote repo
   $ git checkout jswanson-branch
   $ git pull origin jswanson-branch
2. Make sure that the feature branch is up to date with master, while in the feature branch, execrate the following:

$ git pull origin master

If there are any conflicts, best to address them here.

3. Now that I know that the feature branch is up to date with the remote repo and that it has the latest code from master, I can now merge these branches. I also need to make sure that my local master branch is up to date as well.
   $ git checkout master
   $ git pull origin master
   $ git merge --no-ff jswanson-branch
   no fast-forward: This flag keeps the repo branching history from flattening out.
4. Push to remote repo:
   git push origin

To get all remote branches use fetch --all. Before checkout the remote branch, it won't show in branch. Has to use branch –r.