Git Fu

“As you think, so you shall become.”

- Bruce Lee
Version Control?

Git?

Version Control:

- A complete record of all changes
- Can roll back to a previous version

Git:

- An open-source Version Control system
- Git is used to maintain the core Linux kernel
Staging and Commits
Forking and Branching

Take **someone else’s open-source code** and **modify it** for your own purposes

Source code **owner** can **approve a Pull Request**

**Internal developers** can “branch off” into another **side project** without affecting the production code

Is **mainly** used to **fix issues** and **bugs**

Branches can be **merged together**

Git has **one master branch** (the origin) where every other branch **originates from**
Git Workflows: Basic

- Always the start of product development
- Simple and quick to set up
Git Workflows: Feature Branch

- The **basic** workflow is **limited** in workflow freedom
- For each **new feature**, create a **new branch**
- **Merge** the feature branches **back into** the main

Image courtesy of [https://buddy.works/blog/5-types-of-git-workflows](https://buddy.works/blog/5-types-of-git-workflows)
Git Workflows: Gitflow

- Invented by Vincent Driessen in 2010
- Two parallel branches: Master and Develop
- Features are still on their own branches

Image courtesy of https://buddy.works/blog/5-types-of-git-workflows
Let’s Git Goin’
(a proper introduction)

1. Create a GitHub repository using their website
Don’t create a README file just yet!

2. On your local machine, mkdir a folder and cd there

3. Initialize a local repository with git init

4. Don’t forget to introduce yourself :)

5. Now you can create a README.md file:
   - Use your favorite text editor and put some descriptive text in there

6. Start tracking and commit your changes:
   - git status (see what needs to be done)
   - git add . (notice the “dot”, it adds folder files)
   - git status (check again...)
   - git commit -m “My first repo! Woohoo!”
   - git status (and last time...)

7. Time to upload your local repo to your remote:
   - git remote add origin <GitHub repo URL>
   - git remote -v (verify! verify!)
   - git push --set-upstream origin master (publish)

8. Check out your handiwork online!
Branching

1. Create a new branch with whatever name you like:
   - `git branch development`
     (I called mine “development”)
   - `git branch`
     (see what branches you now have)
   - `git checkout development`
     (switch to branch)

2. Add another file and see what happens:
   - `git add "new-file-name"`
   - `git commit -m "Added a new file to branch"`
   - `git push`
     (update the remote repo)
   - `ls`
     (print your files for a quick check)
   - `git checkout master`
     (switch to first branch)
   - `ls`
     (spot the difference...)

3. Let’s Merge:
   - `git merge development`
     (it’s that easy)
   - `git push origin master`
     (update the remote repo)

4. Rename a branch:
   - `git branch -mr development dev-work`
   - `git push origin dev-work`
     (upload the new)
   - `git push -d origin development`
     (delete the old)
Generating SSH Keys
(you’re a pro now)

1. That wasn’t so bad was it? Well, we can make the process easier by not having to type our username and password for every push

2. Introducing... SSH Keys!
   - `ssh-keygen -t rsa -b 2048`
     (note that this will name your key files to the default names of `~/.ssh/id_rsa` and `~/.ssh/id_rsa.pub`)

3. Save your SSH Key profile to GitHub
   - Use your favorite text editor to open `id_rsa.pub`
     WARNING! Look for `.pub`, we don’t want to share the private key!
   - Copy all of the keyfile text
   - Go to your GitHub settings page (click your profile picture in the upper-right corner and select “Settings” from the drop-down menu)
   - Click the “SSH and GPG keys” tab on the left
   - Click “New SSH key”
   - Enter a good title (e.g. “CSE-lab-computers”)  
   - Paste your copied PUBLIC key
   - Click “Add SSH key” and go try a push!
Git Made Easy (no command line!)

- It’s totally **up to you**
- Each has its **pros** and **cons**