Lab 2:
Let's Git it on

SENG 265

Mattermost

- This semester, we are doing a trial-run of using a Slack spin **Mattermost** for student communication

https://mmost.csc.uvic.ca

- Login with your netlink credentials
- You have already been included in private channels run by your TA.
  - If you are not in a channel, or in the wrong channel, just as the your TA to add you.
- This is a trial, so if you find bugs or have issues contact **@adash42**

What is Git?

- Git is an open source version control system or repository (repo)
  - Free Git servers: BitBucket, GitHub
  - Allow multiple users to modify the same content without **conflict**
- This is done by keeping track of a **stream of snapshots** called a **branch**
  - By default, the main branch is called the **master**
  - **We will not be covering branching, do so at your own risk.**
- Basically, a Git Repo is a folder with metadata that describes all the activity (changes) to the repository.
  - A Git clone (your locally stored folder) will contain a copy of the metadata
  - The metadata is stored in a hidden folder called **.git** that is located in the root your local git clone.

Why use Git?

- **Free storage**
  - Most Git services are stored on clouds or are backed up. If you computer crashes, you have you code stored somewhere else and be recovered!
- **You’re an idiot**
  - Accidentally deleted a file?
  - Or the wrong piece of code?
  - Broke something that was working?
  - You can go back to previous working versions of you code!
- **File conflicts**
  - When you get in larger groups, if multiple people are working on the same files, how do you keep the changes straight!
  - Git helps but doing this automatically when it can, and asks when it can’t

Setting up Git config

- First, we're going to setup some global variables for Git (this only needs to be done once)
  - `$ git config --global user.name "<Your Name>"`
  - `$ git config --global user.email "<netlink>@uvic.ca"`
  - `$ git config --global user.editor "vim"`
    - To verify you set it up correctly, type :
      - **$ git config --list**
- You can also view or edit your global Git configuration by:
  - `$ cat ~/.gitconfig`
  - `$ vim ~/.gitconfig`

Getting an existing Git Repository

- We will be using the UVic git repositories created for you.
  - Open Terminal and go to the folder we created last lab
    - `$ cd ~/Desktop/seng265`
    - This is where we'll store our **working directory** (our local copy of the repo)

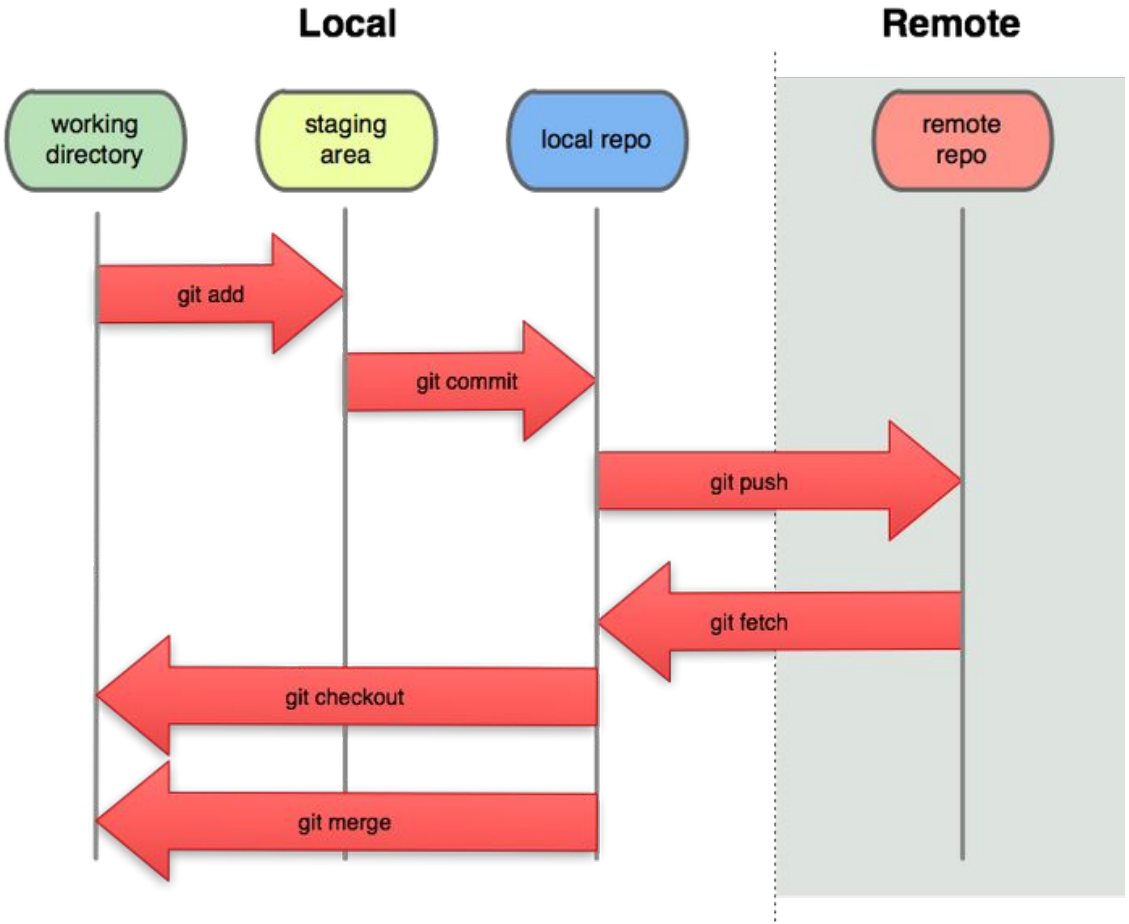
- Download (clone) a copy of an existing remote repository
  - **$** `git clone ssh://<netlink>@git.seng.uvic.ca/seng265/<netlink> [mygit]`
    - When you clone, you can optionally rename the repository
      - In this case, we are naming it **mygit**

And behold ...

- Magic, there is a bunch of stuff!
  - To view all the files and folders:
    - `$ ls -lah`
  - **.git**
    - This contains some of the metadata stored on the remote server
  - **cheatsheets**
    - This contains copies of cheatsheets for vim, linux, c, python, etc.
  - **a1, a2, a3, a4**
    - This is where you will store your assignments for submissions
- So now we have folders, what do we do with it?

First some terminology

- Working directory
  - This is your copy of the repo stored on your computer. This folder will contain your edits.
- Staging Area
  - These are changes that have been flagged by you to be saved locally, **but have not yet been done.**
    - You stage a file by **adding** it.
- Local repo
  - This is your local copy (or **snapshot)** of the repository.
  - The remote server **does not know about it.**
- Remote repo
  - This is the copy of the repository shared by everyone with access to the repo.
  - Rule of thumb, only working code should go here!



https://greenido.files.wordpress.com/2013/07/git-local-remote.png?w=696&h=570

First commit

- Usually, when you do you first commit you add a **README.md** or **.gitignore** file.
  - Go to your working directory (~/Desktop/seng265/mygit)
  - Using vim, create a file called **.gitignore**
    - **$ vim .gitignore**
  - Add the following lines to the file:
    - * ← Ignore all files
    - !*.* ← Don’t ignore files with extensions
    - !*/ ← Don’t ignore directories
    - *.o ← Ignore object files
    - *.pyc ← Ignore python compiled files
    - *.swp ← Ignore vim temporary backup files
    - *~ ← Ignore emacs/notepad++ temporary backup files

First commit

- To see changes in our working directory:
  - **`$ git status`**
  - We have untracked changes.
- To add files type:
  - **`$ git add .gitignore`**
- To see our **staged** file:
  - **`$ git status`**
  - This means we can now save the changes to this file to our local repo by creating a snapshot.
- To create a snapshot:
  - **`$ git commit -m “Your message for the log”`**
  - The commit message cannot be left empty!

First commit

- So, now are we done?
  - Nope! The remote server still has no idea what we just did.
  - To verify this, go back to ~/Desktop/seng265
    - `$ git clone ssh://<netlink>@git.seng.uvic.ca/seng265/<netlink> deletme`
    - `$ cd deleteme && ls -lah`
    - Is the gitignore file there?
- For you assignments, your code MUST be on the remote servers at the assignment deadline.
- To do so, you need to sync with your remote repository. Go back to **mygit**:
  - `$ git push`
    - This may fail if you have a file conflict!

To reiterate ...

- The basic Git workflow goes something like this:
  - You modify files in your working directory, and check to see what needs to be saved.
    - **`$ git status`**
  - You commit, **staging** your changes and creating a new **snapshot**
    - **`$ git add <modified file>`**
    - **`$ git commit -m “Your message about the commit”`**
  - You publish your local changes to the **upstream** on the remote server
    - **`$ git push`**
  - You can grab the latest upstream from the remote server
    - **`$ git pull`**
- For these commands to work, you must be in a working git folder!

Working with Git

- You can use some of the same Linux commands in git!
  - $ git rm <some_file>
  - $ git mv <some_file>
  - $ git grep <pattern>
- So let's take a look at what we've done so far, to view the log run:
  - **`$ git log`**
    - commit <hash>: This is the hash ID for the commit snapshot
    - Author: Who published the snapshot
    - Date: When the snapshot was published
    - Then the message for the submit
  - Remember, anyone can see your comments … including the professor!

Merge Conflicts

- This is when there is a conflict between your local file(s) and the remote server version.
- How do they happen?
  - $ git add filename.c
  - $ git commit -m "made some wild and crazy changes"
  - $ git pull

    from ssh://me@git.seng.uvic.ca/seng265/me
    * branch master -> FETCH_HEAD
    Auto-merging filename.c
    CONFLICT (content): Merge conflict in filename.c
    Automatic merge failed; fix conflicts and then commit the result.
- You've (or someone else) pushed changes upstream from another computer and now there is conflicts!

Merge Conflicts

- To view the differences:
  - $ git mergetool
    - There are graphical tools as well, but this is installed on the linux.csc server.
- Want to keep your changes?
  - `$ git checkout --ours filename.c`
- Want to keep the remote server version?
  - `$ git checkout --theirs filename.c`
- Create a new snapshot and push it upstream
  - `$ git add filename.c`
  - `$ git commit -m "using theirs/ours"`
  - `$ git push`

Common Scenarios

- I pushed changes from another computer, how to I update this computer?
  - `$ git pull`
- Uh oh, I deleted my file and a need to grab another copy!
  - `$ git reset -- <file>`
- What changes have I done so far?
  - `$ git show`
    - Shows all changes
  - `$ git diff <file>`
    - Just shows changes in <file>

Common Scenarios

- I screwed up, can I go back a commit I know works??
  - `$ git reset --hard <working commit hash or origin/head>`
    - **DANGER:** You will lose all local changes!
- Uh, I don't want to lose my local stuff ...
  - `$ git stash`
    - Save my local changes in a queue
  - `$ git reset --hard <working commit hash>`
  - `$ git stash pop`
    - Re-apply my local changes ... merge conflicts will likely occur!
- I get an error that I don't have access when I push ?
  - If you don't have write-permission, you can only make local changes
  - Please let your TA know if this occurs with your UVic git.

Common Scenarios

- I'm super lazy, can I do any of these faster?
  - `$ git config --global alias.<alias_name>  <git command>`
  - Example:
    - `$ git config --global alias.st  status`
  - An advanced gitconfig is on Connex with some fancy aliases provided by Amanda
    - Add aliases to your ~/.gitconfig using vim. Use at your own peril!
- Remove untracked files:
  - `$ git clean [-n] -f`
    - `-n` Runs the command but doesn't actually removee files.
- To get more help information on a command:
  - `$ git <command> --help`

Example C Project

- Let’s look into an example of how to set up an assignment folder
  - In our newly setup git repo, create a folder called **lab2** and go into it
- Copy the following files into **lab2** from Connex
  - **helloworld.c**: Main file that will run a function from hello.c
  - **hello.c**: Contains hello() function that we call from main
  - **hello.h**: Include file for hello.c
  - **Makefile**: File that we use to compile our programs
- If you have issues downloading the files individually, download the zip folder.

Example C Project

- Stage our C project files and new folder:
  - `$ cd lab2 && git add hello* Makefile` **or, just add the folder**
  - `$ git add lab2`
- Create a snapshot that includes our latest changes:
  - `$ git commit -m “Lab 2 C example”`
- And send our snapshot upstream to the remote server (sync)
  - `$ git push`

Example C Project

- Now let's compile! Makefiles make this super easy, just run:
  - **`$ make`**
- You will C an executable file called **helloworld** so run it
  - `$ ./helloworld`
- Run $ **`git status`,** what's missing?
  - The helloworld, helloworld.o and hello.o files aren't listed!
  - That's because of our **.gitignore** file!
- You will be following this same format for your assignments
  - **DON'T** add executables and compile code (*.o, *.pyc) to your assignments
  - **DO** use Makefiles

Final Comments

- USE THE GIT!
  - Almost every job you will have from now on will use a version control system
  - Git is by far the most popular
  - Make meaningful comments when you commit code

- Git will save you time when (and you will)
  - Accidentally erase a file
  - Do something really stupid and need to recover an older version of a file
  - Work from multiple computers and keep everything in sync

Practising C/Python

http://exercism.io/