SENG265: Software Development Methods (Summer 2019)

Lab 02 – Git

Week of May 20th

Authors: Nirav Galani*

*Built on the works of Amanda Dash and David Johnson

A short reminder of remote logins

- To ssh into engineering lab computers
  - ssh <netlink username>@ugls.ece.uvic.ca
- To ssh into computer science lab computers
  - ssh <netlink username>@linux.csc.uvic.ca

From your personal computer …….

- How to copy files/directories from engineering lab computers to a local location(on your computer)
  - Usage:
    - scp [-r] <source path> <destination path>
  - Do not ssh into the remote location
  - scp [-r] <netlink_username>@<ugls.ece.uvic.ca>:</path/to/file>\ <path/to/local/location>
- How do you copy files/directories from a local location to the engineering lab computers??

This week

- Git
  - Configure and clone your individual repo
  - Git basics
    - add
    - commit
    - status
    - log
    - push
    - pull

Setting Up Git

- 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 git repositories created for this class.
  - Open Terminal and go to the seng265 folder we created last lab
    - $ cd ~/Desktop/seng265
    - This is where we will store out working directory (local copy of the repo)
- Clone a copy of the repository created for this class
  - $ git clone ssh: //<netlink_username>@git.seng.uvic.ca/seng265/<netlink_username>
- $ git status
  - What do you see?

Look inside the working directory

- $ change your current directory to your cloned local repository directory
  - $ cd <netlink_username>
- $ ls –lah
- Notice the following items in there
  - .git
    - Currently contains some of the metadata stored in the remote server (master branch). When we make changes, this is where it will be stored.
  - cheatsheets
    - Cheatsheest for bash, vim, python, C, makefile, git

First commit

- Let us add a .gitignore file.
  - With you current directory as the working directory (seng265git), using vim, create .gitignore
    - $ vim .gitignore
  - Add the following lines in the file
    - *.o ← Ignore C object files
    - *.pyc ← Ignore python compiled files
    - *.swp ← Ignore vim temporary backup files
    - *~ ← Ignore emacs/notepad++ temporary backup files
- .gitignore is used to tell git what types of files and folders to ignore (i.e. not track)
- .gitignore is not required, but it does address a lot of potential worries

Add and commit .gitignore

- $ git add .gitignore
  - .gitignore is not added and ready to be stored in the local rep
- $ git status
  - What do we see?
- $ git commit –m “Adding git ignore”
  - This will now place .gitignore into the local repo
  - The commit message cannot be left empty
- $ git status
  - What do we see?

Readme.txt

- Create a file readme.txt inside. You may write anything you want in it.
- Now add and commit this file.
- $ git status
- What is in the remote repository? Does it contain .gitignore and readme.txt?

Lab02

- Create a folder lab02
- Create or place any files you wish inside lab02
- Now add and commit lab02
  - $ git add lab01
  - $ git commit –m “Any message you want”
- Does the remote repository contain lab02
- $ git status

Update the remote repository

- The remote repository does not contain the files we have committed. We will need to push
- $ git push
- $ git status
- Now the remote repository and the local repository are the same.

What have we done so far?

- $ git log
- This will produce a log of what we have done so far
- Note that this is a log for the local repository. Therefore if there are changes that have not been pushed, it will be inconsistent with the master branch on the remote repository
- $ git status

More ……

- Create a folder lab01
- Place all your files and directories from your work in lab01 into this folder.
- Place lab01 into the remote repository.
- Use git status as needed
- Make changes to read.txt using vim
- Make any other changes you would like
- If your lab 01 work is incomplete, then complete it.
- Update the remote repository.

git diff

- You are working on a file abc.py, and you think, what changes have I made from the abc.py that is in my local repository??
- $ git diff abc.py

- Create a clone of this repository on your personal computer.
- You now have access to the remote repository from 2 different computers
- From your personal computer, create a new file testing.py that prints a simple message when run – “Hello World”.
- $ git add testing.py
- Does this make testing.py available to your lab computer?
- $ git commit –m “creating program”
- Does this make testing.py available to your lab computer?
- $ git push

- In the lab computer,
- $ git pull
- Notice now that your local repository has testing.py
- Make changes to testing.py and update the repository so that the changes are available to your personal computer.

Final Comments

- USE 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
  - Try to submit early! Late submissions will not be accepted if things go wrong!

- If you're ever stuck on a conflict or not sure if you've pushed properly, just reclone into a different directory