CSCI 104

Lab 0: Setup

(a.k.a. the longest damn lab ever)

THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.
COOL. HOW DO WE USE IT?
NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.

Getting Started

- Intro to 104 labs

- Intro to Git

- Register with Github
- Install a Linux virtual environment (Docker or VM) to build and test your code
- Configure an SSH key
- Configure Git

These steps are set up for the rest of the semester and are therefore REQUIRED. Make sure to complete this.

Lab Policies

Format

- Lecture (~10-20 mins):
    - The lab instructor will review this week's lab content, which involves a combination of covering new tools (e.g. Git, Makefiles, GDB) and reviewing lecture materials in more detail.
- Work Time:
    - After we cover all necessary content, you are free to work on the lab assignment.
    - If you get stuck, ask for help!
- Checkoff:
    - Before you leave lab, you must ask to see a CP and demonstrate your finished assignment. They will check you off and give you credit. We look for effort in labs, not success.

Checkoffs

- To get checked off, we’ll have you demonstrate your code, and sometimes ask review questions.
    - Then we will mark you off and you’ll get credit for the lab.
- If you don’t finish by the end of lab, and we can see that you have been working the whole time and made an attempt, then we will also check you off.
- *If you sit around idly for 2 hours and don’t demonstrate an effort, you will not get checked off.*
- Checkoffs will be done through a Google form that you will access on a CP’s computer; you will receive a confirmation email upon completion.

Policies

- Labs are mandatory. You **must** attend the lab you are registered for **in person**, and you MUST get checked off before the end of the lab session.
- **You may miss 1 lab session without penalty.** This excused absence should ONLY be used in the case of illness and not just a busy week. Contact your lab leader(s) ASAP if you were ill and missed lab session.
- If you are ill and have already missed a lab, then contact Tallulah Winston-Schrader (winstons@usc.edu) through email or Piazza to attend another section.
- **Labs are graded CR/NC.**
- https://bytes.usc.edu/cs104/labs/

Introduction to Git

Git

- Git is a “distributed version control system”. What does that mean?

Git

- Git is a “distributed version control system”. What does that mean?
- It helps solve two common problems in software engineering:
    - Distributing code between multiple people and computers, and keeping it up to date
    - Keeping track of changes to a codebase, and allowing you to restore old versions if needed

Git

- Git is a “distributed version control system”. What does that mean?
- It helps solve two common problems in software engineering:
  - Distributing code between multiple people and computers, and keeping it up to date
  - Keeping track of changes to a codebase, and allowing you to restore old versions if needed
- Git works on (essentially) a client-server model.
  - The Git server (GitHub) stores a master copy of the code.
  - The clients, your computers, download a copy of the code in a “clone” operation.
  - Clients can then update their copies in a “pull” operation, and can push their own changes using a “push” command.

Commands

- `cd your_directory`
    - `cd ..` to backtrack
- `git init` to create a repository.
- `git add filename` to stage changes to the repo.
    - You must explicitly tell git to stage changes to your repo!
    - Allows the user to choose which changes to push
    - `git add .` tells git to automatically stage all changes it detects.
- `git commit -m “your message”` to create a commit.
    - A commit represents a set of changes to files in the repository
        - E.g. “Add a new file X, change 3 characters on line 8 of file Y”
    - The commit only includes changes you have staged!

Commands

- `cd your_directory`
  - `cd ..` to backtrack
- **`git init` to create a repository.** (although I personally prefer `git clone`; either works)
- `git add filename` to stage changes to the repo.
  - You must explicitly tell git to stage changes to your repo!
  - Allows the user to choose which changes to push
  - `git add .` tells git to automatically stage all changes it detects.
- `git commit -m “your message”` to create a commit.
  - A commit represents a set of changes to files in the repository
    - E.g. “Add a new file X, change 3 characters on line 8 of file Y”
  - The commit only includes changes you have staged!

Commands

- `cd your_directory`
  - `cd ..` to backtrack
- `git init` to create a repository. (although I personally prefer `git clone`; either works)
- **`git add filename` to stage changes to the repo.**
  - **You must explicitly tell git to stage changes to your repo!**
  - **Allows the user to choose which changes to push**
  - **`git add .` tells git to automatically stage all changes it detects.**
- `git commit -m “your message”` to create a commit.
  - A commit represents a set of changes to files in the repository
    - E.g. “Add a new file X, change 3 characters on line 8 of file Y”
  - The commit only includes changes you have staged!

Commands

- `cd your_directory`
    - `cd ..` to backtrack
- `git init` to create a repository.
- `git add filename` to stage changes to the repo.
    - You must explicitly tell git to stage changes to your repo!
    - Allows the user to choose which changes to push
    - `git add .` tells git to automatically stage all changes it detects.
- **`git commit -m “your message”` to create a commit.**
    - **A commit represents a set of changes to files in the repository**
        - **E.g. “Add a new file X, change 3 characters on line 8 of file Y”**
    - **The commit only includes changes you have staged!**

Working with GitHub

- GitHub stores your repository the same way you store a repository on your local machine.
- When you clone a repo from GitHub, you copy the whole repo from GitHub onto your computer.
    - `git clone git@github.com:username/repo_name.git`
- When you pull from GitHub, you download all the commits that are present on GitHub but not in your local repo.
    - `git pull`
- When you push to GitHub, you upload all commits present in your local repo but not on GitHub.
    - `git push`

`.gitignore`

- **`.gitignore`** is a file that contains a list of filenames.
- If git sees a file with `.gitignore` in a directory, it will ignore the files listed when you do "`git add .`"
- You can use "`*`" as a wildcard to match multiple files
  - e.g. "`*.txt`" would ignore all files with the "txt" extension
- You can put "/" at the end of a name to ignore entire directories
  - e.g. "`ignored/`" would ignore all directories named "ignored"

GitHub Cheat Sheet

1. Do **NOT** run <u>**ANY**</u> git commands in Docker. This is easy to forget, but this is the #1 problem that we notice.
   a. Put a sticky note on your computer! Set a reminder! Anything!! Just don't do this!
2. ***OUTSIDE*** of Docker, here are basically the only git commands you need:

```
a.  Git pull
b.  Git add .
c.  Git commit -m "message explaining your changes"
d.  Git push
```

Lab Breakdown

Steps

1. Register with GitHub (happens on GH website)
2. Install Git (happens through local command line)
3. Setup Docker (yes, this is necessary- happens through localcommand line)
4. Configure an SSH Key (happens through local command NOT Docker shell)
5. Git Configurations Pt. 1 (happens through local command line)
6. Git Configurations Pt. 2 (happens on GitHub website)

FAQs

Q: Do you have a recommended code editor?
A: VSCode is a great free option.

Q: Why do we need a VM?
A: Because we need everyone to run their code on a Linux environment so we can verify testing and grading is accurate. Plus, this makes it easier for you so that you don't have to install all the tools we use like Valgrind locally!

Q: Should I use Docker or a traditional VM?
A: Use Docker 100% (please)

Q: Why do I need an SSH key?
A: Think of it like a password; SSH is what keeps your GitHub work secure.

Q: Are you sure I shouldn't use a passphrase for my SSH key?
A: Yes- you will likely regret it later if you do (otherwise you have to type it everytime you push).