Week 01-1

CSC209 Fall 2023

Dr. Demetres (dee) Kostas

September 12, 2023

Welcome back

- Week 1 (this week)
  - Class should have access now
    - quercus, Markus, PCRS
  - But Piazza isn't up yet
- I got some requests for quercus access
  - last class
  - but not many emails
    - please send an email with utorid, student #

Lab 1

- You have a lab due on Friday
- Make sure to follow the instructions
  - on quercus
- Lab starter code is on MarkUs
  - quercus has help for access

PCRS (prepare)

- There are videos
  - that will help for this lab
  - and for reviewing today and Thurs
  - they are not for marks
- Week 2's content
  - is available
  - and is **for credit**
  - due next Tuesday 10am

Terminals, shells, systems

- we're still learning which is which
- the *NIX systems
  - linux, MAC, Unix, FreeBSD, (not Windows)
  - Use a directory structure
    - like a tree with root '/'
  - files are stored in here
    - **everything is a file!**
    - files, devices, programs, directories

A shell

- Is essentially a scripting language
- that allows you to easily execute **other** programs
  - run files
  - e.g. `[user@machine path]$ program_name arg_0 ...  arg_N`
    - this is *in bash*
      - bourne again shell => BASH
    - other common shells are the same (to us)
      - so we just say shell

Execution with a shell

- we find ourself somewhere in the system
  - a directory within the tree structure
    - "working directory"
  - `[user@machine-name /some/path]$`
    - `/some/path/` is where we find ourselves
- We execute programs
  - e.g. `cd`, `ls`, or maybe `python`

Terminal

- *Every program* we execute
  - Three associated files (created by system)
    - stdin, stdout, stderr
      - (Standard Input, Output and Error)
- Our shell program is no different
- A terminal program
  - is a program that allows us
    - to type into stdin (nicely)
    - and view stdout and stderr

Shell

- The shell allows us to
  - run other programs
    - with **command line** arguments
      - *arguments to the main function*
  - view *their* stdout and stderr
    - printing and errors
- and do more... as we will see

So Terminals and shells?

- The terminal is a program that
  - allows us to interactively run a shell
    - a scripting language
      - designed to view/execute other programs
      - these programs navigate the system!
    - We will learn the scripting part
      - later in the course

Built-in programs to learn

- for system management and usage
  - look these up using the `man` command
    - cd, mkdir, ls, cp, mv, rm
    - cat, grep, head, tail
- Note these are programs
  - that are "executable" files
  - somewhere (see $PATH) in the system

Shell features

- Some important shell features
- We've seen how to execute
  - other programs
  - `$ prog_name arg0 ... argN`
    - we use `$` as short form for shell prompt
- Their stdout and stderr
  - is automatically printed
    - before the next `$`

Shell stdin

- if the program is reading input
  - this means it will read from stdin
- In the shell
  - the first way this happens
  - is it just waits for you to type something!
    - reads what was typed

Piping

- instead of typing in
- you can use the output of another program!
  - The `|` character connects the stdout
    - of the program (and arguments)
      - on the left
    - to the stdin of another program (+args)
      - on the right
- e.g. `$ ls -l | head -n 1`
  - what do you think this does?

Piping

- the programs are run simultaneously
- and you can string more than two
  - just keep adding `|`
- In general *NIX
  - programs do **one thing** *really well*
  - and we can glue them together
    - with tools like pipes
      - and more (later)

Redirection ($<$ and $>$)

- In bash, we can use regular files
  - as the destination for stdout
    - using `>`
    - e.g. `echo "Hello World" > hw.txt`
      - *create if* ***`hw.txt`*** *didn't exist*
  - or as stdin to a program
    - e.g. `head -n 1 < hw.txt`

Globbing

- when *using the shell*
  - note this is **not when another program**
    - is running
- You can use some regex-like patterns
  - to match multiple items
- Inserting
  - `*` matches any number of characters
  - `?` matches any *one* character
  - use `[]` to list possible characters
    - `[a-z]`, `[1-5]`, or even `[a-xz]`

C Programming

- Finally...
- Compiled language
  - compiled to machine code for specific devices

  - *in a format readable by specific systems*

- Long history of development
  - has influenced many modern languages
- Rooted in the 'B' programming language
  - 1950's 1960's

The C language

- first completed in 1972
- by 1978 it was an ANSI standard
- 1989 saw the C89 ammendment
- 1999 created the C99
  - **ISO standard**
  - and the version we will use in this course
- There were additional revisions in 2011 and 2018
  - but we will not discuss these

Compiling

- we've said multiple times
  - C is *compiled*
- So who does the compiling?
  - another program
    * called `gcc`
    * The GNU C compiler

Basic usage of `gcc`

- Open a terminal
  - (which starts a shell program)
- Then, gcc is a built-in program (like `ls` or `cd`)
  - so we just type `gcc`
- It will need arguments
  - using `$ gcc source_code.c`
    - this will compile the `source_code.c`
      - and produce an executable program!

Additional arguments

- In this course
- We will keep some additional arguments
- We will add `-Wall`
  - to tell us *all possible warnings*
- We will use `-std=gnu99`
  - to use the C99 standard
- We will also use `-o <prog_name>`
  - to specify a specific program name
    - (replace `<prog_name>` as needed)

So how would we compile?

- If we had a single source file
  - called `source_code.c`
- and wanted to call the resulting program
  - `super-python`
- We would type `gcc -Wall -std=gnu99 -o super-python source_code.c`