cis15
advanced programming techniques, using c++
lecture # I.2

topics:

- unix fundamentals

what is UNIX?

- Unix is an operating system (like Windows).
- OS? - A program that coordinates and oversees the resources of a computer and provides and interface for the users to interact with the machine.
- which means it is a program that runs on a computer that makes it possible for you to use the computer (typically to run other programs)
- in some ways it is relatively old
  - the first Unix was written in the 1970s
  - it turns out that this is a strength :-)
- in some ways it is relatively new
  - there are new versions of UNIX coming out all the time
- there are many “flavors” of UNIX (flavors here is an official term)
  - OS X, Linux, SunOS, etc.

UNIX is and isn't a WIMP

- you are probably most familiar with “WIMP” environments (“window, icon, menu, pointing device”)—style of interacting with the computer using these elements
- while many UNIX systems support this kind of interaction, basic UNIX functionality does not need them
- this is both a strength and a weakness...
- it also means that you need to learn to use the *command line* (which we'll do today)

a little UNIX history

- developed at AT&T Bell Laboratories in the 1970's
- released and distributed free of charge since AT&T was not allowed to compete in the computer industry at the time
- primarily created initially by Ken Thompson and Dennis Ritchie, coming after an interactive, multiuser operating system they had conceived earlier called *multics*—this became jokingly “unics” which evolved into UNIX and was released in 1971
- but early UNIX wasn't perfect, and so researchers at UCal Berkeley created a cleaner version, released in 1982 as “BSD” (Berkeley Software Distribution)
- later, in 1991, Linus Torvalds (Finland), developed a version of UNIX for personal computers—Linux
- today, there are basically four main versions of Unix:
  - System V UNIX (stems from original AT&T version)
  - BSD UNIX (Berkeley)
  - Linux
  - OS X (Mac)

features of UNIX

- "open" software — *non-proprietary*, meaning that no single company or person owns it or is in charge of developing and/or maintaining it
- *multi-tasking* — meaning multiple programs can be running at one time, even on a single CPU system; this is called *timesharing* where the operating system provides small slices of time to multiple programs; switching between which one is actually running in any given millisecond is imperceptible to the user
- components:
  - *kernel* — resident in computer's main memory; primary resource manager; task/process manager
  - *file system* — organizes files into directories
  - *shell* — interactive component that lets users enter *commands* on a "command-line" at a prompt (e.g., `unix$`)
  - *commands* — set of system utilities that come with the operating system which the user can invoke from the command-line

taking command!

- our use of UNIX will be with the Apple Mac OS X operating system installed on the laptops
- OS X includes a graphical environment built on top of a fairly standard flavor of UNIX
- we will focus on the elements that are standard UNIX (and not OS X specific)
- to do this, we will use the *OS X Terminal* utility (I will tell you when something is an OS X extension by indicating this, as above, so you learn what is standard UNIX and what are OS X components)
- when you run this, you get a window with something like:

```
/Users/student$
```

- this is the *command line*—a line on which you type commands! (you did this last class)
- the bit of text on the command line before you type anything is called the *prompt* (e.g., `/Users/student$` or `unix$`)

invoking commands

- in UNIX, the way you get the operating system to do things is to type instructions on the command line and then hit the "return" or "enter" key
- the things you type are the names of commands or programs you want the system to run
- for example, typing:

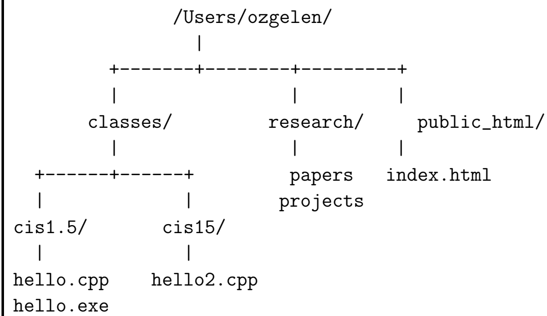
```
unix$ date
```

  after the prompt (and hitting return) gives you the current date and time

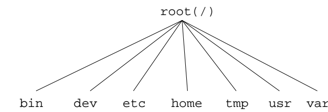
the UNIX filesystem

- the *filesystem* is the part of UNIX that organizes and keeps track of data that is stored on the computer's hard disk or on any storage device
- you need to know a bit about how it works
- as you already know, a *file* is a collection of related data
- UNIX has data files (also called "ordinary" or "regular" files), and it also has:
  - *device files* (special files), which represent pieces of hardware like the screen, or a printer, or a USB memory key
  - *directory files*, which organise ordinary and device files
- directory files (or just "directories") are similar to the folders you are familiar with from Windows
- the file system is *hierarchical*, organized into "trees" (see next slide)

user directory tree example



system directory tree example



- bin: most of the commonly used UNIX commands
- dev: device files
- etc: administrative files (including the password file)
- home: home directories (OS X uses Users)
- tmp: temporary files
- usr: a variety of stuff, depending on the version of Unix
- var: frequently varying data

location, location, location

- every file has an *address*
- that is its location in the filesystem
- UNIX calls this location its *path*
- for example, a file called `myprog.cpp` that is in my home directory will have an (absolute) path(name) of:

```
/Users/ozgelen/myprog.cpp
```


more location

- in a sense, the command line has a location as well
- each time you have a Terminal window open, it is “looking at” a directory
- you can find out which directory it is by typing:

```
unix$ pwd
```

- If I do this right after I open the terminal, I get:

```
/Users/ozgelen
```

moving around

- we can move between directories
- if I'm in `/Users/ozgelen` and I type

```
unix$ ls
```

I get a listing of that directory, something like:

```
admin      code    courses
myprog.cpp papers
```

- to move to the directory `code`, I would then type:

```
unix$ cd code
```

- both `ls` (list) and `cd` (change directory) are UNIX commands

more moving around

- if I'm in `/Users/ozgelen/code` and I want to move back to `Users/ozgelen`, I can type:

```
unix$ cd /Users/ozgelen
```

or

```
unix$ cd ../
```

- `../` is like saying "the parent of the current directory".
- don't mistype. `./` means "this directory", so:

```
unix$ cd ./
```

has no effect (i.e., it changes to the current directory...)

moving things

- if I'm in `/Users/ozgelen` and I want to move `/Users/ozgelen/myprog.cpp` to `Users/ozgelen/code`, I can type:

```
unix$ mv myprog.cpp /Users/ozgelen/code
```

or

```
unix$ mv myprog.cpp code
```

- using:

```
unix$ mv myprog.cpp code/prog2.cpp
```

will not just move the file, but will also change its name.
- using `cp` rather than `mv` will copy the file rather than move it

moving things again

- if I'm in `/Users/ozgelen/code` and I want to move `/Users/ozgelen/myprog.cpp`into `Users/ozgelen/code`, I can type:

```
unix$ mv /Users/ozgelen/myprog.cpp .
```

or

```
unix$ mv ../myprog.cpp .
```

- the dot (`.`) is also like saying "here"
- (In fact saying "." is exactly the same thing as saying "./").

windows in UNIX

- generic "windows" facilitate user access to multiple tasks ("processes") running at the same time
- *window manager* controls "look & feel" of windows
- X Windows developed at MIT (Massachusetts Institute of Technology) for use with UNIX; still the most popular with all flavors of UNIX, even available for Macs

basic UNIX commands

- commands have options or parameters or "switches"
- *switches* start with "–"
- some commands...
  - man
  - pwd
  - cd
  - ls
  - mkdir
  - rmdir
  - cp
  - mv
  - rm
  - chmod
- UNIX IS CASE-SENSITIVE!!! i.e., pay attention to upper-case versus lower-case letters

- Similarly,

```
unix$ who
```

tells you who is using the computer (not so helpful on a single-user machine), and:

```
unix$ exit
```

or

```
unix$ logout
```

will stop the terminal window from running.

man — get help (display manual page)

man — display manual pages (get help!)
man man — display manual page for the *man* command
man ls — display manual page for the *ls* command
man -k file — list all commands with the keyword *file*

```
unix$ man pwd
PWD(1)                          FSF                          PWD(1)

NAME
       pwd - print name of current/working directory

SYNOPSIS
       pwd [OPTION]

DESCRIPTION
       Print  the full filename of the current working directory.
...
```

pwd — print working directory

```
unix$ pwd
/Users/ozgelen/teaching/cis15/notes
```


cd — change working directory

```
unix$ pwd
/Users/ozgelen/
unix$ cd classes
unix$ pwd
/Users/ozgelen/classes
```


ls — list the files in the current directory

ls -aF — list all files and show their file types

```
unix$ ls -aF
./
../
.bashrc
classes/
mail/
hello.cpp
```

ls -l — list files in long format

```
unix$ ls -l hello.cpp
-rw-r--r--    1 ozgelen  faculty   187 Sep 5 10:45 hello.cpp
```


mkdir — make (create) a directory

```
unix$ ls -aF
./
../
.bashrc
classes/
mail/
hello.cpp
unix$ mkdir junk
unix$ ls -aF
./
../
.bashrc
classes/
junk/
mail/
hello.cpp
```

rmdir — remove (delete) a directory

```
unix$ ls -aF
./
../
.bashrc
classes/
junk/
mail/
hello.cpp
unix$ rmdir junk
unix$ ls -aF
./
../
.bashrc
classes/
mail/
hello.cpp
```


cp — copy a file

```
unix$ ls -aF
./
../
.bashrc
classes/
mail/
hello.cpp
unix$ cp hello.cpp hi.cpp
unix$ ls -aF
./
../
.bashrc
classes/
mail/
hello.cpp
hi.cpp
```


mv — move (rename) a file.

```
unix$ ls -aF
./
../
.bashrc
classes/
mail/
hello.cpp
unix$ mv hello.cpp howdy.cpp
unix$ ls -aF
./
../
.bashrc
classes/
mail/
howdy.cpp
```


rm — remove (delete) a file

```
unix$ ls -aF
./
../
.bashrc
classes/
mail/
hi.cpp
howdy.cpp
unix$ rm hi.cpp
unix$ ls -aF
./
../
.bashrc
classes/
mail/
howdy.cpp
```

chmod — change file mode

- 9 characters: `-uuugggooo`
- WHO: u = user, g = group, o = other users, a = all users (u + g + o)
- WHAT: r = read, w = write, x = execute
- MODE: + = allow, - = don't allow

```
unix$ ls -l hi.cpp
-rwxr-xr-x    1 ozgelen  faculty   187 Sep 5 10:45 hi.cpp
unix$ chmod a+w hi.cpp
unix$ ls -l hi.java
-rwxrwxrwx    1 ozgelen  faculty   187 Sep 5 10:45 hi.cpp
```


other UNIX commands

- **diff**: command used to compare the contents of two files
  `Unix$ diff file1.txt file2.txt`
- **more**: command used to list the contents of a file (only works well with plain text files!)
  `Unix$ more file1.txt`
- **wc**: command used to count (and display) the number of lines/words/characters in a file
  `Unix$ wc file1.txt`

special characters: wild card matching

- you can use special characters on the unix command-line as "wild cards" in order to apply a command to a set of files that have similar characteristics
- the general wild card character is asterisk (*), which matches to anything (zero or one or more of any character)
- for example:
  `$unix ls *.txt`
  will list any files that end with `.txt`, such as `file1.txt` and `file2.txt`
  or
  `$unix ls A*`
  will list any files that start with `A`, such as `Abc.txt` and `A_to_Z`, but not `aA`
  or
  `$unix ls A*Z`
  will list any files that start with `A` and end with `Z`, such as `AAAZ` and `A_to_Z`, but not `AAAZ.txt`
- remember, file names and commands are *case sensitive!*

- a single character wild card is question mark (?), which matches to one character
- for example:
  `$unix ls A?.txt`
  will list files such as `AB.txt`, but not `A.txt` or `AAA.txt`
- we will do more with pattern matching and *regular expressions* later in the semester

redirection

- you can "redirect" the output of a command or program to a file using the *redirection* symbol: >
- for example:
  `$unix wc file1.txt > file2.txt`
  will count the number of characters, words and lines in `file1.txt` and store the result in `file2.txt`. if you want to see the result, then you have to display `file2.txt`:
  `$unix more file2.txt`
- redirection will create a new file (or first delete it if it exists) and then write the command/program output to the new file
- if you want to preserve the contents of the file to which the output is being redirected, you can *append* to the end of the file using >>
- for example:

```
$unix wc file1.txt >myfile.txt
$unix wc file2.txt >> myfile.txt
$unix more myfile.txt
```


using C++ under UNIX

- in cis1.5, you used an integrated development environment (IDE)
- probably you used either Dev C++ or CodeBlocks
- the important operations that this IDE allowed you to carry out were:
  - editing a C++ program
  - compiling a C++ program
  - running a compiled program
- you can carry out *exactly* the same steps under UNIX!
- the way that you carry out the steps is different
- you can let me know at the end of the term which you think is easier and which you like better!

editing a C++ program

- we edit our C++ programs using a *text editor*
- we will use **emacs**
- according to the GNU project (who provide it):

  Emacs is the extensible, customizable, self-documenting real-time display editor
- emacs is *free software* (yeah!)

- `http://www.gnu.org/software/emacs/`

what is *free software*?

- emacs is free in the sense that you have:
  - The freedom to run the program, for any purpose (freedom 0).
  - The freedom to study how the program works, and adapt it to your needs (freedom 1).
  - The freedom to redistribute copies so you can help your neighbor (freedom 2).
  - The freedom to improve the program, and release your improvements to the public, so that the whole community benefits (freedom 3).
- Access to the source code is a prerequisite for freedoms 1 and 3.

compiling a C++ program

- to compile our C++ programs, we will use another GNU product — g++, the GNU C++ compiler
- we run the compiler (as we run any UNIX command) by typing on the command line
- to compile the program `myprog.cpp` we need to type:

```
unix$ g++ myprog.cpp
```

  at the prompt.
- If there are errors, g++ will report them on the screen
- If there are no errors, g++ will run silently

using g++

- if we just type:

```
unix$ g++ myprog.cpp
```

  then g++ will create an output file called:

```
a.out
```

- If we want a more meaningful name, then we have to give one, like:

```
g++ myprog.cpp -o myprog
```


running a C++ program

- once your program has compiled successfully, you can run it!
- the compiled program, `myprog` is now something that can be run, just like any other Unix command
- all you hve to do, more or less, is to type its name:

```
unix$ ./myprog
```

- you precede the name of the executable (`myprog`) with the "dot slash" (verb+./+) to tell the operating system the location of the file you want to run, i.e., the current directory (hence the use of "dot" (`.`))
- any output that `myprog` produces will be displayed on the screen

summary

- this lecture introduced some of the basic ideas that you will need to know about the UNIX operating system
- we concentrated on the things that you will need to know in order to:
  - create, edit and save files
  - compile C++ programs
  - run C++ programs
- under the UNIX operating system, Mac OS X specifically