JUST ENOUGH UNIX

What is Unix

- Unix is an operating system (like Windows).
- That means it is a program that runs on a computer, and which 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
 - OSX, Linux, SunOS and so on
 - There are many flavors of Linux also.

Unix is and isn't a WIMP

- You are most familiar with WIMP environments.
 - WIMP stands for "window, icon, menu, pointing device".
- While many Unix systems support this kind of interaction much Unix functionality doesn't need this.
- This is both a strength and a weakness.
- It also means that you need to learn to use the *command line*.

A little history

- Developed at AT&T Bell Laboratories in the 1970s.
- 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)
- All now have decent windowing environments.

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.
- Even a personal computer running UNIX has this ability.
- Typically this means that several people can use the same computer at the same time (though not the same keyboard and screen :-)

• Components:

- *kernel* resident in computer's main memory; primary resource manager; task/process manager.
- *file system* organizes files.
- 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 Unbuntu flavor of the Linux operating system used by the machines on our laptop cart..
- Like any modern Unix system, Ubuntu comes with a graphical interface..
- The bit we'll make use of is the standard Unix.
- However, we will work with the operating system through the Terminal utility, a version of the shell.
- When you run this, you get a window with something like: student>
- This is the command line. A line on which you type commands.
- The bit of text on the command line before you type anything is called the *prompt*.

Making commands

- In Unix, the way you get the operating system to do things is to type instructions on the command line (and then hit "return").
- The things you type are the names of programs you want the system to run.
- For example typing:

date

after the prompt (and hitting return) gives you the date.

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

will stop the terminal window from running.

The Unix filesystem

- The Unix *filesystem* is the part of Unix that organizes and keeps track of data.
- You need to know a bit about how it works.
- As you already know, a *file* is a collection of related data.
- Unix has files like this ("ordinary" or "regular" files) and 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.

Directory tree example

• The file system is organised into trees:



12

File system structure





More structure

- bin: most of the commonly used Unix commands
- dev: device files
- etc: administrative files (including the password file)
- home: home directories
- 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 call myprog.cpp that is in my home directory will have an (asolute) path(name) of:

/home/parsons/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:

pwd

• If I do this right after I open the terminal, I get: /home/parsons

Moving around

- We can move between directories
- If I'm in /home/parsons and I type

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: cd code
- Both 1s (list) and cd (change directory) are Unix commands.

More moving around

• If I'm in /home/parsons/code and I want to move back to home/parsons, I can type:

cd /home/parsons

or

cd ../

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

cd ./

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



• If I'm in /home/parsons and I want to move /home/parsons/myprog.cpp into /home/parsons/code, I can type:

mv myprog.cpp /home/parsons/code

or

mv myprog.cpp code

• Using:

mv myprog.cpp code/prog.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 /home/parsons/code and I want to move /home/parsons/myprog.cpp into /home/parsons/code, I can type:

mv /home/parsons/myprog.cpp .

or

mv ../myprog.cpp .

- The . 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
- The *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.

Basic Unix commands

• Some commands:

– man	– rmdir
– pwd	– cp
– cd	– mv
– ls	– rm
– mkdir	– chmod
• UNIX IS CASE-SENSITIVE!!!	

- Commands have options or parameters or "switches".
- *Switches* start with "–"

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*

• For example:

unix> man pwd PWD(1) PWD(1) FSF NAME pwd - print name of current/working directory SYNOPSIS pwd [OPTION] DESCRIPTION Print the full filename of the current working directory. --help display this help and exit --version output version information and exit NOTE: your shell may have its own version of pwd, which usually supercedes the version described here. . . .



• Print working directory

unix> pwd
/home/parsons/teaching/cis15/notes

cd

• Change working directory

```
unix> pwd
/home/parsons/
unix> cd classes
unix> pwd
/home/parsons/classes
```

ls

- List the files in the current directory
- **Is -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 parsons faculty 187 Sep 5 10:45 hello.cpp
cis15-spring2010-parsons-lectI.Ib
                                                                 27
```

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
```



```
unix> ls -aF
./
../
.bashrc
classes/
mail/
hello.cpp
hi.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 l parsons faculty 187 Sep 5 10:45 hi.cpp unix> chmod a+w hi.cpp unix> ls -l hi.cpp -rwxrwxrwx l parsons 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
while
unix> ls A*
will list any files that start with A, such as Abc.txt and A_to_Z,
but not aA
```

```
• Similarly
```

```
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
- There is much more to say about pattern matching and *regular expressions* but we probably won't get to it this 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
```

If you can't remember all that

• Buy the T-shirt



Using C++ under Unix

- In CIS 1.5, you used an integrated development environment (IDE).
- Typically you used 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.

Editing a C++ program

- We edit our C++ programs using an *editor*.
- One tool we can use for this is Emacs
- According to the GNU project (who provide it):
 - Emacs is the extensible, customizable, self-documenting real-time display editor
- Emacs is free software.



• http://www.gnu.org/software/emacs/

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.

Other editors

- If you don't like Emacs, then there are a couple of other options.
- Nano is another free editor.
- Gedit is another editor that comes with Ubuntu.
- Vi is yet another editor that is commonly used on Unix systems.

Compiling a C++ program

- To compile our C++ programs, we will use another GNU product.
- This is 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:

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.



like:

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

Running a C++ program

- Once your program has compiled successfully, you can run it.
- The compiled program, myprog.o is now something that can be run, just like any other Unix command.
- All you have to do, more or less, is to type its name:

./myprog.o

• 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:
 - Edit;
 - Compile; and
 - Run C++ programs

under the Unix operating system.