CSCI 4061: Unix Basics

Chris Kauffman

Last Updated:
Wed Jan 27 03:56:05 PM CST 2021
Logistics

Reading: Stevens and Rago
- Ch 1: Unix Overview
- Ch 2: Unix Standards (skim)
- Ch 7: Processes (Up Next)

Goals Today
- Finish C review
- Unix Basics
- Process Sys Calls

Assignments
- HW01/Lab01 up
- Due Mon 2/1
- Project 1 is coming

Lab01 Today
- Comments?
- Discord work alright?
Wrap up C Programming Exercises

- Finish Reviewing C programs from last time
- Answer any pressing questions
- Reminder: C programming links on Canvas Homepage
Access to Unix Machines

Several options described in a class tutorial getting access to Unix
https://www.cs.umn.edu/~kauffman/tutorials/unix-environment

▶ CSE Labs
  ▶ Via SSH
  ▶ Via http://vole.cse.umn.edu

▶ Windows: Maybe Windows Subsystem for Linux (WSL)
▶ Mac OS X: No native environment, use a Virtual Machine
▶ Any: Install VirtualBox to host a Unix you like
▶ Install Native Linux or BSD: "Now you’re playing with power!"
Unix Standards: POSIX

POSIX defines what you an plausibly expect on unix-like systems like Linux/BSD. Includes

▶ C libraries for system calls, standard libraries
▶ Basic layout of file system and naming conventions
▶ Some Devices such as /dev/null
▶ Presence of a shell and certain utilities like cat, grep, ... 

Distinction: C Standard vs Unix Library

▶ Lots of systems have a C compiler which has the C standard library: printf(), fopen(), pow() etc.
▶ Unix systems have additional, separate libraries for Unix-specific stuff like read(), fork(), kill()
▶ Some branches of Unix have their own special, special versions of these like Linux clone()
Will discuss briefly tools that are useful for interacting with Unix in a "command shell"

- Shell / Terminal / Command Line / Non-graphical login, etc.
- Most of the discussion is widely applicable to any Unix system
- A few parts are specific to the **Bash** shell specifically (alternatives exist but Bash is default on many systems)
Command Line: Basic File System Navigation

<table>
<thead>
<tr>
<th>Command</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pwd</code></td>
<td>print the current directory</td>
</tr>
<tr>
<td><code>cd folder</code></td>
<td>change directory / folder</td>
</tr>
<tr>
<td><code>ls</code></td>
<td>list files in directory</td>
</tr>
<tr>
<td><code>cd ~</code></td>
<td>change to home directory</td>
</tr>
</tbody>
</table>

> `pwd`
/home/kauffman

> `ls`
1103-F2017  aurs  Downloads  Hello.class  Hello.java~  PathClassLoader.txt
4061-F2017  Desktop  Dropbox  Hello.java  misc  public_html

> `cd 4061-F2017`

> `ls`
exams  lectures  Makefile~  projects  schedule.html~  schedule.org~  textbook
labs  Makefile  misc  schedule.html  schedule.org  syllabus

> `pwd`
/home/kauffman/4061-F2017

> `cd lectures`

> `pwd`
/home/kauffman/4061-F2017/lectures

> `ls`
00-course-mechanics.org  00-course-mechanics.tex  01-introduction.org  01-introduction.tex
00-course-mechanics.org~  01-introduction-code  01-introduction.org~  02-unix-basic.c
00-course-mechanics.pdf  01-introduction-code.zip  01-introduction.pdf  02-unix-basics.org

> `cd ~`

> `pwd`
/home/kauffman

> `ls`
1103-F2017  aurs  Downloads  Hello.class  Hello.java~  PathClassLoader.txt
4061-F2017  Desktop  Dropbox  Hello.java  misc  public_html
Typical Unix Directory Structure

- rooted at /, reachable via 'cd /'
- user directories such as /home/kauffman/
## Determining File Types

<table>
<thead>
<tr>
<th>Command</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>file something.ext</code></td>
<td>try to determine the type of given file</td>
</tr>
</tbody>
</table>

> `file xxx`  
xx: UTF-8 Unicode text, with very long lines  
> `file test.txt`  
test.txt: ASCII text  
> `file www`  
www: directory  
> `file 4061-F2017`  
4061-F2017: symbolic link to `/home/kauffman/Dropbox/teaching/4061-F2017`  
> `file 4061-F2017/`  
4061-F2017/: directory  
> `cd 4061-F2017/lectures/`  
> `file 01-introduction-code.zip`  
01-introduction-code.zip: Zip archive data, at least v1.0 to extract  

> `file 02-unix-basics-code/no_interruptions.c`  
02-unix-basics-code/no_interruptions.c: C source, ASCII text  

> `file 02-unix-basics-code/no_interruptions.o`  
02-unix-basics-code/no_interruptions.o: ELF 64-bit LSB relocatable, x86-64, version 1 (SYSV), not stripped  

> `file 02-unix-basics-code/a.out`  
02-unix-basics-code/a.out: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked, interpreter `/lib64/ld-linux-x86-64.so.2`, for GNU/Linux 2.6.32, BuildID[sha1]=ff87934737b0e48b891d27573ae8a2e5687c46a, not stripped
Searching and Manipulating Text

<table>
<thead>
<tr>
<th>Command</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat file.txt</td>
<td>show contents of file in terminal</td>
</tr>
<tr>
<td>less file.txt</td>
<td>&quot;page&quot; text file, press &quot;q&quot; to quit</td>
</tr>
<tr>
<td>grep 'expression' file.txt</td>
<td>show lines matching expression in file</td>
</tr>
<tr>
<td>grep 'expression' *.txt</td>
<td>search every .txt file for lines</td>
</tr>
<tr>
<td>find .</td>
<td>show all files recursively from current directory</td>
</tr>
<tr>
<td>find .  -name '.*.c'</td>
<td>find all C source files recursively</td>
</tr>
</tbody>
</table>

These are very handy, worth knowing about, but won’t be covered in detail during our course. Try the relevant session in *Tool Time Lectures* if curious.
Editing Files

- There are fancy text editors like Atom (free, GUI, on lab machines)
- Then there are the old-school terminal editors like these:

<table>
<thead>
<tr>
<th>Command</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>vi</td>
<td>modal editing, terse, powerful, ALWAYS present</td>
</tr>
<tr>
<td>emacs</td>
<td>modes for editing, extensible, mostly available, ♡</td>
</tr>
<tr>
<td>nano</td>
<td>simple, podunky, usually available</td>
</tr>
</tbody>
</table>

- Learn some vi or emacs: long-term worthwhile investment
- Comes in real handy when you need to edit but there is no graphical login
File Permissions / Modes

- Unix enforces file security via *modes*: permissions as to who can read / write / execute each file
- See permissions/modes with `ls -l`
- Look for series of 9 permissions

```bash
> ls -l
total 140K
-rwx--x--- 2 kauffman faculty 8.6K Oct 2 17:39 a.out
-rw-r--r-- 1 kauffman devel  1.1K Sep 28 13:52 files.txt
-rw-rw---- 1 kauffman faculty 1.5K Sep 26 10:58 gettysburg.txt
-rwx--x--- 2 kauffman faculty 8.6K Oct 2 17:39 my_exec
---------- 1 kauffman kauffman 128 Oct 2 17:39 unreadable.txt
-rw-rw-r-x 1 root     root 1.2K Sep 26 12:21 scripty.sh
```

----

PERMISSIONS

- Every file has permissions set from somewhere on creation
Changing File Permissions

<table>
<thead>
<tr>
<th>Command</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ls -l</strong></td>
<td>long listing of files, shows permissions</td>
</tr>
<tr>
<td>chmod u+x file.abc</td>
<td>make file executable by user</td>
</tr>
<tr>
<td>chmod o-rwx file.abc</td>
<td>remove permissions from other users</td>
</tr>
<tr>
<td>chmod 777 file.abc</td>
<td>everyone can do anything to file</td>
</tr>
</tbody>
</table>

```
> ls
a.out  no_interruptions.c  no_interruptions.c~  no_interruptions.o
> ls -l
total 40K
-rwxrwx--- 1 kauffman kauffman 8.5K Sep 7 09:55 a.out
-rw-r--r-- 1 kauffman kauffman 955 Sep 7 09:55 no_interruptions.c
-rw-r--r-- 1 kauffman kauffman 883 Sep 7 09:54 no_interruptions.c~
-rw-rw---- 1 kauffman kauffman 2.4K Sep 7 11:59 no_interruptions.o
> chmod u-x a.out
> ls -l
total 40K
-rw-rwx--- 1 kauffman kauffman 8.5K Sep 7 09:55 a.out
-rw-r--r-- 1 kauffman kauffman 955 Sep 7 09:55 no_interruptions.c
-rw-r--r-- 1 kauffman kauffman 883 Sep 7 09:54 no_interruptions.c~
-rw-rw---- 1 kauffman kauffman 2.4K Sep 7 11:59 no_interruptions.o
> ./a.out
bash: ./a.out: Permission denied
> chmod u+x a.out
> ./a.out
Ma-na na-na!
```
Manual Pages

<table>
<thead>
<tr>
<th>Command</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>man ls</td>
<td>Bring up the manual page for command <code>ls</code> 'space' scrolls down, 'q' quits</td>
</tr>
</tbody>
</table>

> man ls | cat
LS(1)                     User Commands                      LS(1)

NAME

ls - list directory contents

SYNOPSIS

ls [OPTION]... [FILE]...

DESCRIPTION

List information about the FILEs (the current directory by default). Sort entries alphabetically if none of -cftuvSUX nor --sort is specified.

Mandatory arguments to long options are mandatory for short options too.

-a, --all
    do not ignore entries starting with .

-A, --almost-all
    do not list implied . and ..
## Program Search PATH

<table>
<thead>
<tr>
<th>Command</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>echo $PATH</td>
<td>show where shell looks for programs</td>
</tr>
<tr>
<td>PATH=$PATH:/home/kauffman/bin</td>
<td>also look in my bin directory</td>
</tr>
<tr>
<td>PATH=$PATH:.</td>
<td>also look in current directory</td>
</tr>
<tr>
<td>PATH=.</td>
<td>ONLY look in the current directory</td>
</tr>
</tbody>
</table>

```bash
> echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/bin:/usr/lib/jvm/default/bin:
/usr/bin/site_perl:/usr/bin/vendor_perl:/usr/bin/core_perl:/home/kauffman/bin:
/home/kauffman/Dropbox/bin:/home/kauffman/code/bin:/home/kauffman/code/utils:.
```

Search directories are separated by colons in Unix

Note: PATH is a notable **Environment Variable**. We’ll discuss these soon and how they relate to processes.
Exercise: Compilation

1. What command is typically used to compile C programs?
2. What is the default name of a compiled program on Unix and how can it be changed?
3. What function does a runnable C file need to have to make a program?
4. Must every C file have that special function? Can you compile C files without that special function?

Write down your answers as a team for screen sharing
Answers: Compilation

1. What command is typically used to compile C programs?
   > gcc myprog.c

2. What is the default name of a compiled program on Unix and how can it be changed?
   > ./a.out
   > gcc -o mprog mprog.c
   > ./myprog

3. What function does a runnable C file need to have to make a program?
   - main() must be present in at least one C file to make program

4. Must every C file have that special function? Can you compile C files without that special function?
   > gcc -c funcs1.c # produces funcs1.o
   > gcc -c funcs2.c # produces funcs2.o
   > gcc -o prog funcs1.o funcs2.o # link object files
   # Either funcs1.c or funcs2.c had main(), not both
   > ./prog
make and Makefiles

- Example of a **build system**
- Very old system, many newer ones but a good starting point
- Discussed in HW01 which is due soon
  - Get some experience creating a Makefile
  - Will be a required element for Projects
How make and Makefile Works

Build up dependencies recursively

- A tree-like structure (actually a DAG)
- Run commands for the lowest level
- Then go up a level
- Then up another ...
- Can recurse to subdirectories to use other Makefiles as well
- Makefile describes dependencies between source/program files and commands to generate/compile

Makefile Format

target1 : dependency1 dependency2
    do command 1
    then do command 2

target2 : target1 dependency3
    do command X
    then do command Y
# Showing and Murdering Running Processes

<table>
<thead>
<tr>
<th>Command</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ps</code></td>
<td>show running processes associated with terminal</td>
</tr>
<tr>
<td><code>ps a</code></td>
<td>show ALL running processes</td>
</tr>
<tr>
<td><code>ps u</code></td>
<td>show all processes for me</td>
</tr>
<tr>
<td><code>kill 1234</code></td>
<td>send process 1234 the TERM signal</td>
</tr>
<tr>
<td><code>kill -9 1234</code></td>
<td>send process 1234 the KILL signal</td>
</tr>
<tr>
<td><code>pkill a.out</code></td>
<td>send process named a.out the TERM signal</td>
</tr>
<tr>
<td><code>pkill -9 a.out</code></td>
<td>send process named a.out the KILL signal</td>
</tr>
<tr>
<td><code>top</code> or <code>htop</code></td>
<td>interactive process monitoring / signaling</td>
</tr>
</tbody>
</table>

```
> ps
   PID TTY TIME CMD
  8050 pts/1 00:00:00 bash
  8061 pts/1 00:00:00 ssh
11033 pts/1 00:00:00 ps
> ps u

USER    PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND
kauffman 724 0.0 0.0 201092 5520 tty2 Ss1+ Sep06 0:00 /usr/lib/gdm/gdm-x-session --run-script cinnamon-session-cinnamon
kauffman 726 0.1 0.5 691872 94388 tty2 Rl+ Sep06 2:08 /usr/lib/xorg-server/Xorg vt2 -displayfd 3
kauffman 737 0.0 0.3 603020 49496 tty2 S1+ Sep06 0:00 cinnamon-session --session cinnamon
kauffman 784 0.0 0.1 565264 23008 tty2 S1+ Sep06 0:00 /usr/lib/cinnamon-settings-daemon/csd-orient
```
Exercise: Summarize the most important Unix Commands

1. Discuss what the most important Unix command line concepts and commands are for beginners
2. Explain these in your own words, how to use them, and why they are important
3. How do these commands interact with the operating system? What role does the OS play in the command?

Write down your answers as a team for screen sharing