Chapter 1 - Introduction

Introduction

- Overview of Linux/Unix
- Shells
- Commands: built-in, aliases, program invocations, alternation and iteration
- Finding more information: man, info

Help in the Unix world

- The “man” program has been the primary means for local documentation for the Unix world. Use “man man” to find out more, and “man -k KEYWORD” to search for a keyword in the man pages.
- In Linux, the canonical place to find the most current man page information is at https://www.kernel.org/doc/man-pages/
- In most Unix/Linux distributions, the man pages are generally in directories like /usr/man/man? or /usr/share/man/man?
- The “info” program is useful, but it helps to be adroit with Emacs if you want to use it.

Linux

- Linux has been around for 30 years, and is a leading platform for operating system development
- Linprog machines use the CentOS distribution; other popular server distributions include Debian, Arch, Redhat, and OpenSuSE
- Linux Mint is apparently the most popular desktop distribution, though it's arguable that Ubuntu is

Unix in depth

- You can view the operating system presentation in layers
  - The kernel, which provides access to resource, both virtual and physical
    - State can be observed directly via /proc and /sys
  - The system calls, which allow us to access and modify resources

Unix in depth

- More layers
  - Libraries, which provide more digestible access to system calls; for instance, contrast getdirents(3) with getdents(2)
  - Processes, which take advantage of libraries (and sometimes direct system calls); an example of processes would be a shell, which provides a platform for a user to start other processes and script repetitious tasks

Working with these layers

- How do we create and manipulate these various layers?
  - Processes
    - fork(2), exec(2), wait(2), exit(2); clone(2)
  - Filesystems
    - mount(2), getdents(2), stat(2), statfs(2)
  - I/O
    - open(2), read(2), write(2), close(2)
    - socket(2) et al

Some definitions

- An *executable* is a file that can be "executed" an existing process.
  - A static executable is a standalone program that does not need any other runtime support
  - A dynamically linked executable requires the services of a runtime linker such as /lib64/ld-linux-x86-64.so.2 in order to dynamically load shared libraries
  - A "script" executable, which requires a separate interpreter such Perl or Python

More definitions

- A “process” is an activation of a program. Creating a new process is done by making a new entry in the process table; also, in Linux, a thread, which retains the execution context of the parent, also goes into the same process table.
- A “daemon” is a process that generally provides a service of some sort; it is either itself persistent, or is the child of some persistent process.

More definitions

- A “user shell” provices an environment that accepts keyboard input and provides output in order to allow a user to execute programs
- A “built-in” command to a shell does not cause the execution of a new process; often, it is used to change the state of a shell itself.
- An “alias” is a string that is to be expanded into another command
- A “variable” is a way to reference state in a shell. We also have “environmental” variables which specify state for a process.
- A “flag” affords us a method to specify options on the command line, generally indicated either by a single dash or a double dash

Filtering

- A “filter” should read from its stdin (file descriptor 0) and write to its stdout (file descriptor 1); any error or miscellaneous information should be written to stderr (file descriptor 2)
- Generally, filters should not read configuration files but should instead take their options from command line indicators like “flags”.
- Ideally, the output of one filter should be easily readable by another filter.

Unix file characteristics

- Unix files normally follow the paradigm of a "byte-stream"
- Filenames may consist of most Unicode UTF-8 characters except the NUL (ASCII 0) byte and the "/" (ASCII 47)
- Filenames are by default case sensitive (though you can do unusual things to create filesystems that are not)
- Periods are generally used for filename extensions.
- However, filenames that start with a period are generally not displayed by most core utilities unless an explicit flag is given (for example, ls -a)

Some popular filename "extensions"

- .c, .h for C files
- .pl, .pm for Perl files
- .py, .pyc for Python files
- .cpp, .c++, .CC for C++ files
- .s, .S for assembly files
- .o for object files
- .a for static libraries
- .so for dynamic libraries
- .gz for files compressed with gzip
- .bz2 for files compressed with bzip2
- .rpm for RPM files
- .tar for tarfiles

Speaking of filenames

- "Globbing" is a wildcard system for matching file and directory names (aka "path names")
- An asterisk "*" can match any number of characters in path name
- A question mark "?" matches any single character
- Square brackets "[ ]" let you specify a character class (unfortunately, square brackets are also used for other things)

Filesystems

- Directories are tree-structured, and begin at /
- "cwd" is the current working directory; you can use the /proc filesystem to see the current working directory for a process:

```
$ ls -l /proc/$$/cwd
lrwxrwxrwx [ ... ]  /proc/4665/cwd -> /tmp
```

Filesystem paths

- In Unix, we use / to distinguish elements in a path
- Absolute paths begin with / which means start at the root
- Relative paths start with any other character and are interpreted as begin relative the current working directory.

More on paths

- "." is a special path (and it is actually in the filesystem) that points at the current directory.
- ".." is also a special path (and also exists in the filesystem) that points at the parent directory ("/" (root) is its own parent.)
- "~/" is often understood by a shell as referring to home directory of the current user
- "~username/" is often understood by a shell as the home directory of "username"

Shortcutting path information

- It's tedious to type full path names, and relative pathnames are often not much better. To alleviate some of this tedium, you can specify an environmental variable "PATH" which gives some default locations to look for binaries. (Indeed, a surprising number of programs attempt to consult various "PATH"-type environmental variables; perhaps the most surprising – and dangerous – is LD_LIBRARY_PATH.)

Listing files

- The program /bin/ls can take many different options. “-l” shows a detailed listing, using one line per file; “-a” includes the dot files; “/bin/ls -d DIRNAME” shows information about directory rather than its contents

File permissions and user classes

- Each file in a filesystem has uid associated with it; this uid is referred as the "owner" of a file
- Each file in a filesystem also has a gid associated with it; this gid is referred to as the "group" of a file
- We use the term "other" to refer to all users who are not the owner or in the same group.

File permissions, rwx

- The “r” indicates permission to read
- The “w” indicates permission to write
- THe “x” indicates permission to execute

Changing permissions with chmod

- The program chmod accepts either octal notation, such as "chmod 755 /bin/ls" (which closely mirrors how permissions are accessed via system calls)
- Or it also accepts symbolic notation, such as "chmod og+w /etc/hosts"

Unlinking and removing files and directories

- A single file can have many links in a given filesystem (you can see the link count with “ls -l”)
- A process can use the system call unlink(2) to remove a file (but not a directory)
- A process can use the system call rmdir(2) to remove an empty directory

Unlinking and removing files and directories

- Users generally use programs like “rm” and “rmdir” to remove files and directories
- “rm -r” combines unlink(2) and rmdir(2) to recursively remove directories
- “rm -i” queries the user whether or not to remove a link
- “rm -f” means don’t complain about non-existent items
- “rmdir” lets you remove an empty directory

Manipulating files with link, cp, and mv

- "link FILE1 FILE2" creates a new link to a file (often called a hard link)
- "cp FILE1 FILE2" copies a file
- "cp -r DIR1 DIR2" copies a directory; creates DIR2 if it doesn't exist otherwise puts the new copy inside of DIR2
- "cp -a DIR1 DIR2" is like "cp -r", but also does a very good job of preserving ownership, permissions, soft links and so forth
- "mv NAME1 NAME2" moves a file or directory

Symbolic links

- Unix also supports the idea of a “symbolic” link, which is just an ordinary flat file which has a bit set that tells the kernel to treat the contents of the file as a path (often called a soft link)

Using “ln”

- “ln FILE1 FILE2” creates a hard link to FILE1 named FILE2
- “ln -s NAME1 NAME2” creates a soft link to NAME1 named NAME2
- “ln FILE1” creates a new hard link to FILE1 in the current directory
- “ln -s NAME1” also creates a soft link to NAME1 in the current directory

Displaying files

- “cat” takes stdin and sends it to stdout
- “cat FILE” opens FILE and sends it to stdout ” “more” provides a pager to view a file or stdin
- “less” provides a pager, but it additionally buffers stdin so that you can go backwards
- “head” displays the first lines of a file
- “tail” displays the final lines of a file
- “tail -f” displays the file, and blocks; if the file is appended to, it then shows the new lines

Standard i/o

- By default, processes start with three active file descriptors: 0, 1, and 2
- 0 is used for standard input
- 1 is used for standard output
- 2 is used for standard error

Redirection

- You can use $>$ and $<$ for simple redirection
- You can be explicit in bash and provide the file descriptor number, such as “ls 2$>$ /tmp/outfile”
- You can use » to append to a file.

Redirection via pipes

- In addition to redirection to a pathname, Unix has long allowed you to join the output of a process to the input of another process, such as “ls /usr/bin | wc -l” (the “|” indicates this redirection.)
- The “tee” program lets you split the output of one process to another into a separate bytestream to stdout.

Programs that report on other programs

- The most helpful is probably “which”, which can clear up quite quickly just what “gcc” is being referred to.
- “whereis” can also be useful since it provides a bit more information.
- “whatis” is occasionally useful.

Who else is around?

- The “who” and “w” programs can show you who else is using a system
- The “last” program can show you who has been using a system