Linux Essentials for Application Developers

David Levine
levined@ieee.org
last update: 28 Mar 2019

License

Contents

- Introduction
- Linux
- Unix
- The console
- Text tools
- File and directories
- Memory and processes
- bash
- Networking
- System administration
- Software development tools

Introduction

back to Contents

Introduction

- The purpose of this document is to provide sufficient information for an experienced programmer to become proficient in a Linux environment
- Notation:
  - commands that are entered at a shell (command-line) prompt are in **`fixed-with font`**
  - command arguments are ***`italicized`***
  - ... indicates an argument that can appear any number of times
  - examples are indented:

```
    echo 'text to display'
```

- Exercises have the answer on the following slide

back to Contents

What is Linux?

- Free and open source multiuser operating system
  - Free: anyone is free to do whatever they would like with the code, subject to license agreement
  - Open source: the source code is publicly available for anyone to study and modify
  - Operating system: the executing code (program) that manages all system resources, including hardware and software
  - Multiuser: supports simultaneous use by multiple users
- Some refer to GNU/Linux when referring to the entire operating system, including GNU tools and the Linux kernel

Where is Linux?

- Linux is a free and open source operating system
- Linux runs on everything from embedded IoT devices to supercomputers
- The Android cellphone operating system is based on the Linux kernel
- So Linux is running on billions of devices worldwide

Linux distribution families

- A Linux distribution typically includes the Linux kernel, a suite of programs, and a package distribution system
- Most currently supported Linux distributions derive from one of these distributions, first released in 1993 (except Android):
  - Slackware/SuSE
  - Debian
  - Red Hat
    - most of the distribution-specific examples in this deck will be based on Red Hat
  - Android, first released in 2005
- A. Lundqvist and D. Rodic have prepared a fascinating GNU/Linux Distribution Timeline, at
https://upload.wikimedia.org/wikipedia/commons/5/58/Linux_Distribution_Timeline_with_Android.svg

Can I emulate Linux on Mac or Windows?

- On Mac, open the terminal app. Most of the commands in this deck will work on it. Though some, such as find, are slightly different
  - Mac OS X and later are based on Mach, a Unix variant
- On Windows, Cygwin provides the feel of Linux. To install, download and run https://www.cygwin.com/setup-x86_64.exe
  - its package management is different but intuitive
- Alternatively, Linux virtual machines can be run on Mac, Windows, or any other platform that supports them

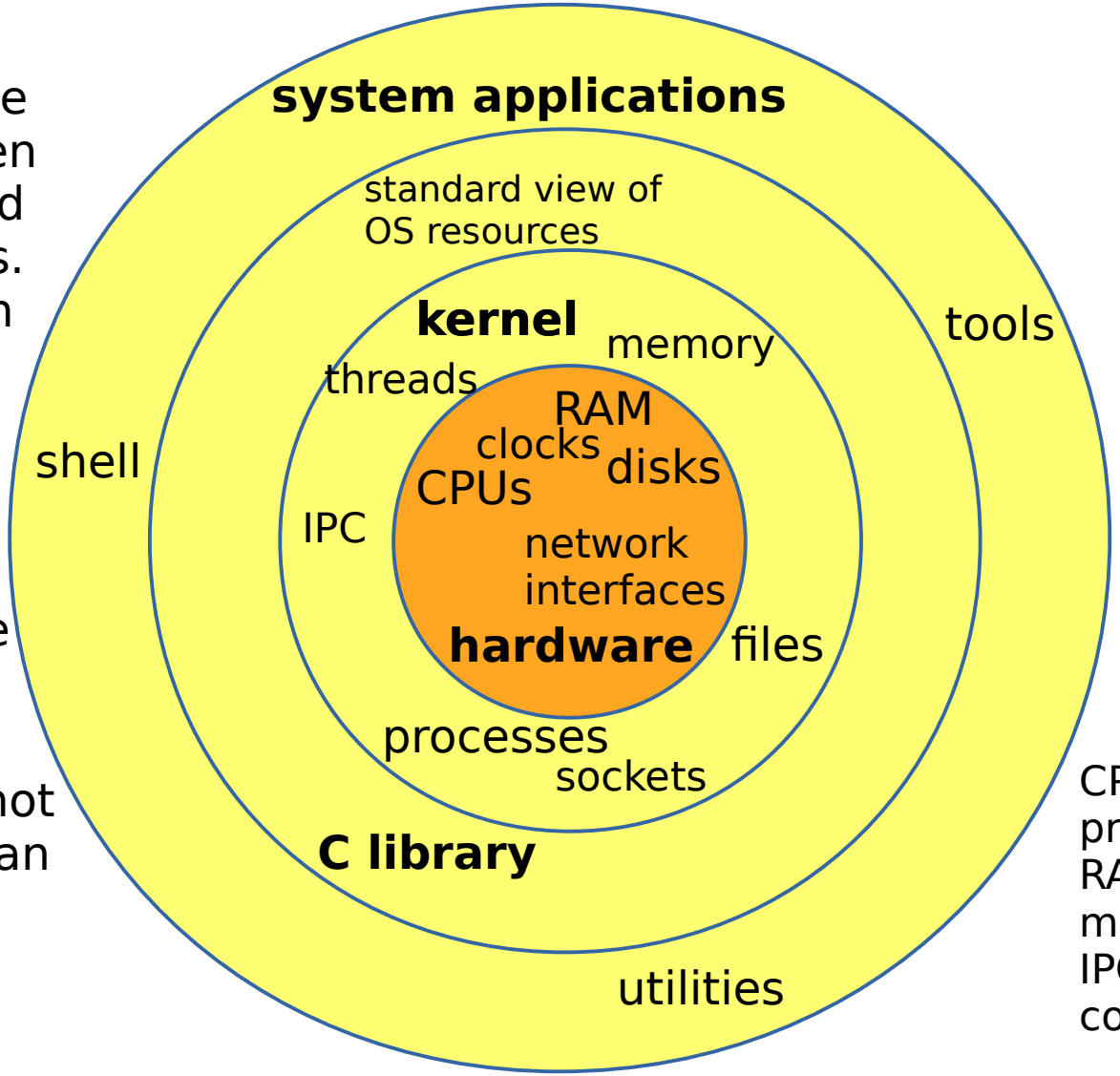
back to Contents

Where did Linux come from?

- MIT, Bell Labs, and GE developed Multics in mid-1960s
  - Multiplexed Information and Computer Services
- AT&T Bell Labs developed Unix in the early 1970s for internal use
  - Uniplexed Information and Computer Service
  - stylized as Unix, trademarked as UNIX®
- Unix licensed to others starting in 1983 as AT&T System V
  - University of California, Berkeley developed UCB version
  - POSIX (Portable Operating System Interface) standardizes user and programmer interfaces
- Linus Torvalds developed Linux in 1991 in response to restrictive licensing of Unix

What is an operating system?

An OS is a software resource manager between the hardware and user applications. Each OS layer (in yellow here) provides resources to layers above through a well-defined interface



User applications (not shown here) can access OS facilities and resources

CPU = central processing unit
RAM = random access memory
IPC = interprocesss communcation

The Unix philosophy

- As summarized by Doug McIlroy in 1978:

1. Each program (application) should do one thing, and do it well
2. The output of one program can be the input to another
    - Loose coupling, i.e., not dependent on particular formats
3. Design and build software to be tried early
    - Rapid prototyping, agile
4. Build and use tools, even for one use

Command basics

- Commands have a name, any number of options, and any number of arguments
  - options are also called flags or switches, and may themselves take arguments
    - all are separated by whitespace: one or more space or tab characters
  - options are indicated with a leading single or double dash
    - single-dash options are usually single letter
    - double-dash options are usually more descriptive, and preferred in scripts and documentation
  - arguments usually don't start with a dash
  - some commands use a double-dash, with no other characters, to separate the command options and arguments

Commands and programs

- A command executes a either a built-in shell command or external program
  - built-in shell commands, documented in the shell man page
  - an external program is contained in an executable disk file
    - execute permission is required for the user to execute the program
  - an external program may either be a shell script or binary
    - shell scripts should start with line of the form:

      #! /bin/bash

      and can contain any commands for that shell
    - binary files contain executable code, or code in and intermediate form that is executed by another program such a Java virtual machine

Help

- Every command should have a man page
  - view with man(1p) command, e.g.,

    `man less`
  - shows man pages related to a topic

    `man -k topic`
  - man can be run directly on a man page

    `man /usr/share/man/man1/ls.1.gz`
- Most commands have a --help, -h, or -? help option

Reference manual sections

- Manual pages are arranged in sections, see those with

```
man man
```

- To specify a specific section when using man, precede the topic with the section number, e.g.,

```
man 3 sleep
```

  - Without the section, man would show the documentation for the sleep(1p) command

Reference conventions

- Commands, system calls, file formats, and other Unix entities are often referred to with their manual section in parentheses the first time the command is mentioned, e.g.,
  - less(1) is the less command, described in section 1 of the reference manual
  - regex(7) documents POSIX regular expressions in section 7 of the reference manual
- In this document, POSIX standard commands are shown with a section 1p reference, e.g, more(1p). The corresponding section 1 man page shows any non-POSIX features of the command
  - Section 3p man pages correspond to Section 2 pages
- Non-POSIX commands are shown with a section 1 reference, e.g, less(1)

The console

back to Contents

Keyboard: prompt shortcuts

- These are the default (emacs mode) bindings. vi bindings can be selected by entering **`set -o vi`**
  - Ctrl-a/Ctrl-e: beginning/end of line
  - Ctrl-b/Ctrl-f: backward/foward one character
    - or left/right arrows
  - Ctrl-p/Ctrl-n: previous/next command in history
    - or up/down arrows
  - Ctrl-r: search back through command history
  - Ctrl-u: erase input
    - Can be used at password prompt
  - Ctrl-d: end of input
- The reset(1) command fixes a “messed up” terminal

Keyboard: tab completion

- bash and other shells offer tab completion
- After entering the first few characters of a command, hitting the Tab key will complete if unique
  - If there isn't a unique completion, nothing happens
  - Then, hitting the Tab key a second time shows all possible completions
    - Enter more characters until the completion is unique
- Tab completion also works for filenames
- Tab completion may also be supported for command arguments

Keyboard: job control shortcuts

- Ctrl-z: put job in background
- Ctrl-s/Ctrl-q: stop/resume foreground job
- Ctrl-c: terminates foreground process by sending it the keyboard interrupt signal
- Ctrl-\: terminates foreground process by sending it the quit signal
  - Only try if Ctrl-c doesn't terminate the process

Keyboard: virtual consoles

- Ctrl-Alt-F1 through Ctrl-Alt-F6 (typically): go to the specified virtual console.
  - X windows runs on one of them, usually the first
- If your X windows environment enters an unusable state, you might be able to switch to another virtual console, login, and try to kill an errant process
  - Example:
    1. Run a process, such as xlock(1), in a window
    2. Switch to another virtual console using Ctrl-Alt-F2
    3. Login
    4. Kill the process started in step one using the kill(1p) command
    5. Logout
    6. Switch back to X windows using Ctrl-Alt-F1 (usually)

The X Window System™

- The X Window System™ provides the basis for the graphical user interface (GUI) of most Linux distributions, except Android
  - The X Window System is a trademark of The Open Group
  - The latest version is X11R6.4
  - Also called X11, X windows, or X
- Client-server based
  - The X server renders and manages displays, including pointer and audio
  - Graphical applications are clients
- X.Org provides the server, Xorg(1), implementation for many distributions

Using X

- Start using startx(1) or xinit(1)
  - usually done for the user on login
- The DISPLAY environment variable controls where windows are displayed
  - Default value is something like :0
  - Must be set manually if unset, e.g., after su

```
export DISPLAY=:0.0
```

Window managers

- With an X server running the user can start a *window manager*
  - Again, this is usually done for users in a system that has been configured with graphical interfaces
  - Sometimes run user's ~/.xinitrc script, which should end by starting a window manager
  - Or, the startx program may start a particular window manager
- The window manager supports root (on the display background) menus, manages physical displays, manages application windows, and manages events such as keyboard and pointer (mouse) input, and audio output
- Example window managers include mwm, twm, and xfce

xterm(1)

- xterm(1) is a terminal emulator for X
- Each invocation opens a new window with a terminal (command) prompt
  - Runs the user's shell, as set in /etc/passwd
  - A user can change their shell with chsh(1)
- Appearance can be customized in ~/.Xresources file
  - Usually read by xrdb(1)
  - Contains settings for other X clients, not just xterm

Desktop environments

- Desktop environments such as GNOME and KDE provide a higher level abstraction
- Include a GUI server such as X
- Include a window manager
- Include applications for various user-level tasks

Text tools

back to Contents

Text tools

- View: cat(1p), less(1), od(1p) or hexdump(1), wc(1p)
  - more(1p) is a predecessor of less that allows only forward movement
  - od and hexdump show the content as bytes
- Search: grep(1p), strings(1p), cut(1p), join(1p)
  - fgrep (grep -F) and egrep (grep -E) are variants that disable and enhance the grep regular expressions
- Compare: diff(1p), cmp(1p)
- Modify/transform: python(1), sed(1p), tr(1p)
  - perl(1) and awk(1p) are older tools that are sometimes useful

less customization

- Options to less can be stored in the LESS environment variable. That can be set your shell profile such as ~/.bash_profile, e.g.,

```
export LESS=eMqRsX
```

-e removes the need to use “q” to terminate less

-M causes less to prompt verbosely

-q suppresses alerts when trying to view before the beginning or past the end of the file

-R interpret ANSI color escape sequences, so that text colors are retained

-s squeezes multiple blank lines into one

-X disables clearing the screen on termination

grep

- grep searches for text matching a regular expression pattern

  grep [*options*] *pattern [file...]*

  - -i case insensitive
  - -c just returns count
  - -h suppresses filename in output, with multiple files (not POSIX)
  - -E (or egrep): extended regular expression pattern
  - -F (or fgrep): literal pattern

grep regular expressions

- This is just a general introduction, please see the grep(1p) and regex(7) man pages for details
- An item can be a single character, a character class, or a parenthesized group of items
- A period, **.**, matches any single character
- Some special characters (except with -F or fgrep) specify the quantity of the preceding item
  - * matches any number of the item
  - + matches one or more of the item
  - ? matches zero or one of the item

grep regular expressions, cont'd

- Some special characters (except with -F or fgrep) restrict the match
  - ^ matches at the beginning of a line
  - $ matches at the end of a line
- [] indicates a character class, such as [A-Za-z0-9] or [:alpha:]

grep examples

- Count occurrences of “error” or “warning” in log files

```
grep -cEi 'error|warning' files
```

- Set exit status to zero if “error” or “warning” occurs in log files

```
grep -Eiq 'error|warning' files
echo $?
```

- Show all processes that match pattern, except grep itself, e.g.,

```
ps -ef | grep '[p]ython'
```

  - without the brackets in the pattern, the output would include “grep python”, which probably is not of interest

List of dictionary words

- /usr/share/dict/words contains a list of English words
  - if the words package is installed
  - /usr/share/docs/words/readme.txt describes the contents
- Can be useful, though it is not authoritative

Dictionary word exercise 1

- List all 6-letter words, excluding proper names, that start with 'a' and end with 'x'

  solve exercise before proceeding

Dictionary word exercise 1

- List all 6-letter words, excluding proper names, that start with 'a' and end with 'x'

```
grep '^a....x$' /usr/share/dict/words


adieux
afflux
amplex
anatox
approx
auspex
```

Dictionary word exercise 2

- List all 6-letter words, including proper names, that start with 'a' and end with 'x'

  solve exercise before proceeding

Dictionary word exercise 2

- List all 6-letter words, including proper names, that start with 'a' and end with 'x'

```
grep '^[Aa]....x$' /usr/share/dict/words
```

or

```
grep -i '^a....x$' /usr/share/dict/words
```

Adds the following to the output of the previous exercise:

```
Astrix
Atarax
```

Files and directories

back to Contents

File and directory names

- File and directory names can contain any character except /
  - Though it's best to avoid non-printable characters in names
- Names are case sensitive
  - For example, foo and Foo refer to two distinct files and/or directories
- File and directory names are just entries in tables
  - Removing a file does not remove its contents from disk
  - Though in practice, recovering the contents of a removed file is difficult, or impossible if the contents were overwritten for use by another file

File types

- The most common file types are:
  - Plain file, containing bytes
  - Symbolic link, which points to another file
    - created with **ln -s**
  - Directory, which is an entry in a directory's list of contained files
  - Special files, often used as the OS interface to a hardware device
- The file(1p) command shows the type of a file
- Plain files are in turn categorized based on their content
  - /etc/mime.types lists many types
- In general, filename extensions are not authoritative

File management commands

- cp(1p): copies one or more files, -p preserves ownership, permissions, and timestamps

  `cp [-p] *foo bar*`

- rm(1p): removes (deletes) a file

  `rm [-i] *foo[...]*`

- mv(1p): move (rename) a file

  `mv *foo bar*`

- touch(1p): create new file or update last modification time on existing one
- file(1p): show the type of a file
- stat(1): show ownership, permissions, and timestamps

Directory management commands

- mkdir(1p): makes one more more new directories
  - The -p option allows creation of multiple levels
- rmdir(1p): removes one or more empty directories
  - So does rm -r
- ls(1p): lists contents of one or more directories
- cd(1p): changes current directory
- pushd/popd: shell built-ins to change current directory, pushing/popping to/from a stack
  - dirs displays the contents of the stack

File and directory permissions

- Permissions restrict actions on a file or directory
  - The actions are read, write, and execute
    - A file can only be executed if it allows execute permission
  - Separate permissions for owner, group, and all others
- ls -l and stat(1) show permissions
- chmod(1p) allows owner, and root, to change permissions

rm

- The rm(1p) command removes files and directories
- The -i option causes rm to request confirmation from the user of each removal
  - Many people add a shell alias to their profile to always enable -i. This can be overridden by adding -f when executing the command, or bypassing the alias with \rm
- The -r option is required to remove a directory
  - all of the files and directories are removed first, recursively
- Removal does not actually remove the data in the file
  - removal just unlinks the file name from its directory
  - the space for the data can be reused by the operating system

rsync

- rsync(1) is a handy and efficient tool for copying files and directories to a remote machine, or on the local machine
  - synchronizes files or directories
- The general format is:

  `rsync [options] src…` ***`dest`***

- The source(s) and destination can be local or remote files or directories. Remote files and directories are indicated with a leading [*user*@]*host:*
- There is one subtlety: source directories should always include a trailing **/**
- Usually use -a (--archive) option with directories
- Other common options: -v for verbose, --delete to delete, --exclude to exclude specified files/directories

rsync examples

- Copy directory on local machine

  rsync -av *`/path1/dir/ /path2/dir`*

- Copy remote directory to local machine

  rsync -av *`remote:path1/dir/ path2/dir`*

  - Can use absolute or relative paths in either of the above examples

- Copy, excluding .o files

  rsync -av --exclude '*.o' *`remote:/path1/dir/ path2/dir`*

- Test if contents of two directories are identical

  rsync -nav --delete *`dir1/ dir2`*

File compression

- Compression can drastically reduce the size of a large text file
- Compression effectiveness on a binary file depends on its content; it can even increase the size

Compression	File extension	Introduced in
compress	.Z	1983
gzip	.gz	1992
bzip2	.bz2	1996
lzma	.lzma	1999
xz	.xz	2009

Common compression formats

- Each compression format has a corresponding program of the same name, see man pages for usage info

File compression examples

- decompress a gzip file

  ```
  gzip -d foo.gz
  ```

  or

  ```
  gunzip foo.gz
  ```

- detect type of contents of gzip file

  ```
  gzip -cd foo.gz | file -
  ```

  or

  ```
  zcat foo.gz | file -
  ```

- compress file using bzip2

  ```
  bzip2 foo
  ```

File archives

- Archives are used for packaging a collection of files

Archive format	File extension	Compressed
ar	.a	no
ZIP	.zip	yes
tar	.tar	no
gzipped tar	.tgz	yes
cpio	.cpio	no
gzipped cpio	.cgz	yes
7-Zip	.7z	yes

Common archive formats

- See man pages for usage info

File archive examples

- view contents of ar archive

```
ar t foo.a
```

- extract contents of gzipped tar archive

```
tar xvpf foo.tgz
```

- extract contents of gzipped cpio archive

```
zcat foo.cgz | cpio -imdv
```

or

```
gzip -cd foo.gz | file -
```

- create compressed tar archive

```
tar cvzf foo.tgz files… directories…
```

Disk management

- Disks are divided into *partitions*, e.g.,

```
Device     Boot      Start        End    Sectors  Size Id Type
/dev/sda1  *          2048  163842047  163840000 78.1G  7 HPFS/NTFS/exFAT
/dev/sda2        163842048  348162047  184320000 87.9G 83 Linux
/dev/sda3        348162048  409602047   61440000 29.3G 83 Linux
/dev/sda4        409602048  488397167   78795120 37.6G  5 Extended
/dev/sda5        409606144  411703295    2097152    1G 83 Linux
/dev/sda6        411705344  473151487   61446144 29.3G 83 Linux
/dev/sda7        473153536  488396799   15243264  7.3G 82 Linux swap
```

- fdisk(1) can view and modify local disks
- df(1p) reports partition space usage

Filesystems

- Each partition contains a *filesystem* e.g.,

```
Filesystem     1K-blocks     Used Available Use% Mounted on
/dev/sda2       90581112 79042092   6914636  92% /home
/dev/sda3       30106488 14921476  13632628  53% /mnt/Fedora28
/dev/sda5         999320   505968    424540  55% /boot
/dev/sda6       30109560 15885504  12671520  56% /
```

- mkfs(1) creates a filesystem on a partition

Devices: /dev/null

- /dev/ contains special files and directories, e.g., to easily access hardware
  - Documented by name, e.g., null in manual section 4
- /dev/null: reads nothing, like reading an empty file; discards data written to it
  - POSIX grep does not have --quiet, so:

```
grep foo file >/dev/null && echo file contains foo
```

  - Can be used to discard unexpected error messages, though will also discard unexpected error messages

```
grep foo * 2>/dev/null
```

Devices: /dev/zero, RNGs

- /dev/zero: reads endless stream of 0 (null) bytes

```
head --bytes 10 /dev/zero | od -x
```

- /dev/full: always full device, useful for testing behavior of programs that write to full disk

```
ls >/dev/full
```

- /dev/urandom: returns random bytes, non-blocking so relies on pseudorandom number generator
- /dev/random: random number generator (RNG) that blocks to achieve sufficient entropy

Devices: hardware

- /dev/cdrom, /dev/cdrw, /dev/dvd, or similar: CD or DVD drive(s)
- /dev/sd*: disk drives, starting with /dev/sda
  - Partitions are numbered following the drive letter, starting with /dev/sda1
  - sd refers to Small System Computer Interface (SCSI), which might be emulated by the disk driver
- /dev/hd*: non-SCSI hard disk drives
- /dev/fd*: floppy disk drives

Devices: terminals

- /dev/tty*: terminals (consoles)
  - You can write to, and read from, terminal /dev/tty
- /dev/pts/*: pseudoterminals
  - Used for windows: each window is associated with a different pseudoterminal
  - You can write to, and read from, your own pseudoterminals:

```
echo 'guess who!' >/dev/pts/0
```

  - The w(1) command shows all terminals and pseudoterminals in use
- /dev/ttyS*: serial terminals
  - Used for connection to serial devices such as networking equipment and embedded computers

Devices: I/O

- /dev/stdin, /dev/stdout, /dev/stderr: input and output (I/O) devices
  - Correspond to the 3 standard streams associated with every Unix program
  - Within the program, each stream is associated with a *file descriptor (fd)*
    - An fd is an integer handle to the open file
    - 0 for stdin, 1 for stdout, 2 for stderr
- stdin (standard input) is for console (keyboard) input
- stdout (standard output)is for normal program output
- stderr (standard error) is for error messages or other unusual output

Devices exercises

- Write a simple, non-blocking shell command to display 10 random bytes

  solve exercise before proceeding

- Write a command to read input from your own terminal and display it on the terminal

- Write a command to read input from your own terminal and not echo back to the terminal

Devices exercises

- Write a simple, non-blocking shell command to display 10 random bytes

```
head --bytes 10 /dev/urandom | od -x
```

or

```
head -c 10 /dev/urandom | od -x
```

- Write a command to read input from your own terminal and display it on the terminal

  solve exercise before proceeding

- Write a command to read input from your own terminal and not echo back to the terminal

Devices exercises

- Write a simple, non-blocking shell command to display 10 random bytes

```
head --bytes 10 /dev/urandom | od -x
```

or

```
head -c 10 /dev/urandom | od -x
```

- Write a command to read input from your own terminal and display it on the terminal

```
cat </dev/tty
```

- Write a command to read input from your own terminal and not echo back to the terminal

solve exercise before proceeding

Devices exercises

- Write a simple, non-blocking shell command to display 10 random bytes

```
head --bytes 10 /dev/urandom | od -x
```

or

```
head -c 10 /dev/urandom | od -x
```

- Write a command to read input from your own terminal and display it on the terminal

```
cat </dev/tty
```

- Write a command to read input from your own terminal and not echo back to the terminal

```
cat </dev/tty >/dev/null
```

Reading CDs and DVDs

- Mount a CD or DVD using mount(1):

```
mount /dev/cdrw /mnt/tmp
```

  - The device might be something else, such as cdrom or dvd. The mount point can be any directory and must already exist. It usually should be empty, because its contents won't be visible after the mount.

- The mount point, */mnt/tmp/*, can then be used like any other directory. Though it is read-only, so it can't be modified.

- Unmount the filesystem using umount(1):

```
umount /mnt/tmp
```

Writing CDs and DVDs

- CDs and DVDs usually contain ISO9660 filesystems
- Copy a directory hierarchy to a so-called iso image:

  mkisofs -DRU -max-iso9660-filenames -o *image.iso directory*

- Mount an iso image so its contents can be viewed:

  mount -o loop *image.iso /mnt/tmp*

  - On Macs, the command is:

    hdiutil attach *image.iso*

  *u*mount as usual with umount (hdiutil detach on Macs)

- Burn an iso image to a CD:

  cdrecord -v dev=/dev/*cdrw* -data *image.iso*

- Burn an iso image to a DVD:

  growisofs -dvd-compat -Z /dev/dvd=*image.iso*

Image files

- The ImageMagick package has useful utilities for viewing and manipulating image files
- View with display
  - left click to access modifications menus
  - right click to manage files
- Convert to different format with convert

```
convert foo.jpg foo.png
```

- List all formats supported by convert

```
convert -list format
```

- For more complex image manipulation, try gimp
- The ffmpeg package has useful utilities for viewing and manipulating video files

PDF files

- View pdf files with xpdf(1)
- Adobe has an old version of acrobat reader available at http://ardownload.adobe.com/pub/adobe/reader/unix/9.x/9.5.5/enu/AdbeRdr9.5.5-1_i486linux_enu.rpm
  - It's 32-bit, so might require many the installation of many library packages
- The poppler-utils package contains tools for manipulating PDF files, including:
  - pdfinfo(1)
  - pdfseparate(1)
  - pdfunite(1)
  - pdftotext(1)
- unoconv(1) can convert documents to and from pdf

PostScript files

- Before PDF, PostScript was a widely used electronic document format
  - Both PostScript and PDF were created by Adobe
- Still used by printers
- And used as an intermediate format when converting text to PDF, e.g.,

```
enscript -Bp- foo.txt | ps2pdf - foo.pdf
```

  - enscript(1) isn't always installed on modern systems

Unix and DOS files

- Unix files use ASCII LF (line feed, newline), hex 0A, to indicate the end of a text line
- Microsoft products use a two-character line end sequence: ASCII CR (carriage return), hex 0D followed by LF
- Mac OS X and later use LF, prior used just CR
- dos2unix(1) and unix2dos(1) convert files between the formats

```
dos2unix -n dos_file.txt unix_file.txt

unix2dos <unix_file.txt >dos_file.txt
```

- The ascii(7) man page lists the 128 ASCII characters and their hex values

Memory and processes

back to Contents

Memory managment

- top(1) shows current usage and uses of CPU and memory
- free(1) reports current memory usage, e.g,
-
```
$ free -h
              total        used        free      shared  buff/cache   available
Mem:          7.5Gi       670Mi       4.6Gi        82Mi       2.3Gi       6.5Gi
Swap:            0B          0B          0B
```

- vmstat(1) reports virtual memory statistics

Process management

- top(1) lists processes as well as CPU and memory utilization
- ps(1p) lists processes

```
ps -efj | less
```

  - To see all of your own processes: **`ps x`**
  - Options preceded with **–** are different than options not preceded with dash (BSD options)
- pgrep(1) searchs for processes by name
- kill(1p) sends a signal to a process, see next slide
- pkill(1) kills processes by name

Process termination

- Ctrl-C or Ctrl-\ usually terminate the foreground process
- Background processes can be usually be terminated by looking up its pid and sending one of these signals:

  kill -INT *pid*

  kill -QUIT *pid*

  kill -TERM *pid*

  kill -KILL *pid*

  - -KILL (or -9) should only be used as last resort, because it does not allow the process to perform any cleanup operations

  – Some programs can also be killed with -HUP, but some other programs use -HUP to cause reload of a configuration file

Date and time

- UNIX times start (are 0) at midnight of January 1, 1970, the *epoch*
- The date(1p) command gets and sets the current system date and time
- A service, ntpd, is often used to synchronize the system date and time with known sources
  - **`ntpq -p`** can be used to query the status of ntpd

Timing a command

- The time(1p) command (also a built-in bash command) shows how long it takes to run another command
  - **`time -p`** provides a familiar output on many different systems
    - real (elapsed), user (CPU time spent executing the user program), and system (CPU time spent executing system processes on behalf of the user for the program)
- Example:

```
time -p sleep 2
```

produces

```
real 2.00
user 0.00
sys 0.00
```

back to Contents

The shell

- The shell provides a so-called command line
  - it waits for a user to enter a command, then executes the command
- You can do just about anything from the command line
- Most modern Linux installations default to the bash shell
  - Bourne-again shell
- Other shells include Bourne shell (/bin/sh), csh, tcsh, and Korn shell (ksh)
  - Shells are listed in /etc/shells, but it usually contains other programs that aren't useful for interactive, i.e., command line, use

Shell basics

- A shell provides the following capabilities:
  - a command prompt, where command are entered
    - $ by default, but can be set to anything
  - ability to redirect input to and output from commands
    - < and > to redirect standard input and output, respectively, from a file
    - | to pipe the standard output of one command to the standard input of another
  - job control
  - parameter substitution
  - a language with conditional and iteration constructs

Types of commands

- Internal (built-in) command
  - When entered at the command prompt, the shell performs the command without loading it from a file
  - Includes control flow such as conditionals and iteration
  - Varies by shell, but often includes **`echo`** and **`test`**
- External command
  - Contained in a file, so must be loaded each time it is run
- Alias
  - An alias is just another name for a command
- Function
  - A function can be defined, that when run performs a sequence of specified commands

PATH search

- External commands are found on the user's PATH
  - PATH is an environment variable
  - PATH contains a colon-separated list of directories
  - When the file containing an external command must be found, the shell looks for it in each directory, in order, of the PATH
    - Not used if the file *absolute* or *relative* path
      - An absolute paths starts with **/**
      - A relative path contains at least one **/** but does not start with a **/**
        - Subdirectory, e.g., *foo/program*
        - Sibling directory, e.g., *../dir/program*
        - Current directory, *./program*
- To prevent an attack by a malicious program in current directory, do not include **.** in PATH

Job control

- Appending & to a command starts it in the background
  - If the command's standard input and error are redirected to /dev/null, it can be detached from the terminal, e.g.,

    ```
    command >/dev/null 2>&1 &
    ```

  - 2>&1 redirects standard error to the standard output stream
- Ctrl-z followed by bg(1p) can be used to move the foreground job to background
- fg(1p) brings a background job to foreground

Command history

- bash stores the history of commands
  - View using the history built-in command
  - Saved in file named by $HISTFILE
    - Defaults to ~/.bash_history
    - Disabled by unsetting HISTFILE
- Can be navigated at command prompt using Ctrl-p, Ctrl-n, Ctrl-r, Ctrl-s, etc.

Shell variables

- Shell variables are assigned with =, without any spaces, e.g.,

```
myvar=foo
```

- Be sure to quote values with special characters

```
myvar='foo bar'
```

- The value of a shell variable is accessed by preceding the name with $

```
echo $myvar
```

- Brace expressions can provide a default or alternate value, see Parameter Expansion in the bash(1) man page

```
echo ${myvar:+replacement value}
```

Environment variables

- Every program execution has an *environment*, which includes *environment variables*, each of which is a unique name and a (possibly null) value
- Shell variables can be designed to be in the environment

```
export MY_PORT=1000
```

  - bash allows combined assignment and export, Bourne shell separates them
- A program cannot modify its parent's environment
- A parent's environment is passed to its children
  - At the time the child is forked: subsequently changing the parent's environment does not alter the environment of the child process

Environment control

- The source (or **.**) built-in command can update the current environment from commands in a file

```
echo ${X:-X not set}
cat >/tmp/X <<<X=x
. /tmp/X
echo ${X:-X not set}
```

- The -i option to env(1p) suppresses the environment transfer from parent to child

```
env -i program
```

  and add specific settings

```
env -i FOO=bar program
```

Shell variable exercise

- What is the output from the first and last of this sequence of commands?

```
echo ${X:-X not set}
```

solve exercise before proceeding

```
echo X=x >/tmp/X
. /tmp/X
echo ${X:-X not set}
```

solve exercise before proceeding

Shell variable exercise

- What is the output from the first and last of this sequence of commands?

```
echo ${X:-X not set}
   X not set
echo X=x >/tmp/X
. /tmp/X
echo ${X:-X not set}
   x
```

Shell aliases

- Shell aliases are typically defined in an initialization file so that they are always available
- A common alias is 'rm -i' for rm(1p), so that it prompts for every removal

```
alias rm='rm -i'
```

- Can an alias be bypassed? How do I remove many files at once without responding to the prompt for each?
  - Prepending a backslash to the command bypasses its alias:

    ```
    \rm -fr foo
    ```

  - Alternatively, the full path to an external command can be used to bypass the alias:

    ```
    /bin/rm -fr foo
    ```

- An alias can be removed with unalias

bash initialization files

- Per user:
  - ~/.bash_profile is sourced for the user's login shells
  - ~/.bashrc is also sourced by each new interactive shell
- System wide:
  - /etc/profile is sourced for login shells
  - Other files, such as /etc/bashrc and the files in /etc/profile.d/, may be sources as well
- Cleanup files can also be created, see the FILES section of the bash(1) man page

Shell command prompt

- Default command prompt is $ for Bourne shell and derivatives
  - Except it is # for superuser
  - Can be changed by setting PS1 through PS4 parameters (variables)
- The bash function on the next slide can be used to customize the primary bash command prompt
  - To always enable it, add the function to your ~/.bash_profile
  - To try it once, put it in a file and **`source`** the file
  - See PROMPT_COMMAND shell variable description in the bash(1) man page for more information

bash function to customize prompt

```
if [ "$PS1" ]; then
  export PROMPT_COMMAND=prompt_command
  function prompt_command() {
    local exit_status=$?
    local reset='\[\e[0m\]'
    local red='\[\e[1;31m\]'
    local yellow='\[\e[0;33m\]'
    local blue='\[\e[0;34m\]'

    if [ $exit_status = 0 ]; then
      #### hostname in blue, directory in yellow
      PS1="${blue}\h ${yellow}\W${reset}$ "
    else
      #### plus exit status of previous command, all in red
      PS1="${red}[exit ${exit_status}] \h \W${reset}$ "
    fi
  }
fi
```

bash tidbits

- ~ expands to the user's home directory when encountered by the shell in appropriate context
  - Bourne shell does not have this expansion
- {x..y} is the sequence of numbers from x to y
- $((expr)) evaluates an integer math expression
  - For floating point, use bc(1p), **`bc -lq`**
- (command) executes command in a *subshell*
  - To run in a different environment, employ complex output redirection, and other obscure purposes
- The bash man page has nearly 6,000 lines of useful information

bash exercises

- Write a one-line command sequence to print “done” after 30 seconds

  solve exercise before proceeding

- How do you execute a program named **xyz** in your current directory (assuming no **.** in your PATH)?

bash exercises

- Write a one-line command sequence to print “done” after 30 seconds

  ```
  sleep 30 && echo done
  ```

  or less concise

  ```
  if sleep 30; then echo done; fi
  ```

- How do you execute a program named **xyz** in your current directory (assuming no **.** in your PATH)?

  solve exercise before proceeding

bash exercises

- Write a one-line command sequence to print “done” after 30 seconds

  ```
  sleep 30 && echo done
  ```

  or less concise

  ```
  if sleep 30; then echo done; fi
  ```

- How do you execute a program named **xyz** in your current directory (assuming no **.** in your PATH)?

  ```
  ./xyz
  ```

Shell scripts

- If you're going to perform a sequence of commands more than once, put them in a file, called a shell script
- Check each command for success
  - In bash and other Bourne shell derivatives, can exit execution of the immediately on any failure by setting -e shell variable in (near the top of) the shell script

```
set -e
```

- true(1p) and false(1p) for avoiding or forcing their respective exit status
  - E.g., to proceed if no errors in a log file:

```
grep -iq error logfile || true
```

Shell (script) facilities

- Conditionals (if, &&, ||), iteration (for, while, until)
  - && and|| allow concise conditional execution, e.g.,

```
grep -q foo file && echo found || echo not found
```

- Input (<) and output (>, >>) redirection, pipes (|)
  - > overwrites output file, >> appends to it
- *Here documents* and bash *here strings* allow in-line provision of text input, e.g.,

```
cat >file <<-EOF
   this text input will be redirected to the file
EOF
```

Basic output

- echo(1p) outputs its argument
  - Most shells contain a built-in version, which may be slightly different
  - Examples:

```
echo $LESS
echo $((6*7))
```

- printf(1p) outputs using C-style formatting
  - bash contains built-in version
  - Example, to print hex value of a decimal number:

```
printf '%x\n' 255
```

Output redirection

- bash supports redirection of output in every imaginable way, and then some. Some very simple examples:
  - redirect stdout to file: ***command >file***
  - redirect stderr to file: ***command 2>file***
  - redirect stdout and stderr to same file: ***command &>file***

    or: ***command >file 2>&1***
    - stdout is buffered but stderr is not, so they may be interleaved messily
  - redirect stderr through pipe: ***command1 |& command2***

    or: ***command1 2>&1 | command2***
  - redirect stdout to stderr: ***command 1>&2***

Command substitution

- Command substitution executes a command and provides its output
  - Bourne shell provides backticks, ``
  - bash also provides $(), which can be embedded
- Examples

```
start_dir=`pwd`
myid=$(id -u)
```

find(1p) and xargs(1p)

- find(1p) searches for files meeting specified criteria
  - The criteria can include properties such as filename, permissions, type, owner, group, size, and time of creation, last update, or last access
- xargs(1p) applies an operation to the arguments fed to its standard input
- Together, find and xargs provide a powerful tool to search files for specified content
  - Example, find all .txt files under my home directory that contain the string *Foo*

```
find ~ -name '*.txt' -print0 | xargs -0 egrep -l Foo
```

  - The print0 option to find and -0 option to xargs allow proper handling of filenames with special characters

Living on the command line

- bc(1p) calculator

```
bc -lq
```

- cal(1p) calender

```
cal -3
```

- date(1p) shows the current date and time
  - And components can be selected with the + format option
  - To show seconds since epoch:

```
date +%s
```

Command-line word lookup

- Here is a bash function to provide command-line access to online dictionary:

```
function dict() {
    # First parameter is word to look up.  Can be optionally be
    # appended with :database.
    # Second, optional, parameter is either d (define), m (match),
    # or show (for first arguments of db or strat).
    curl dict://dict.org/${2:-d}:$1
}
```

- To set up, add that function to your ~/.bash_profile and source it

```
. ~/.bash_profile
```

- To use:

```
dict word
```

Networking

back to Contents

Networking: hosts

- Every node (machine) on an Internet network has an address
  - IPv4: 32 bit address, e.g., 10.0.0.1
  - IPv6: 128 bit address, e.g., 2601:018f:0902:7700:0000:0000:0000:0004
- Names are more convenient than numbers, so they are mapped to addresses using host tables and domain name service (DNS)
  - The host table is stored in file /etc/hosts and/or is available from DNS servers
  - DNS servers are listed in /etc/resolv.conf(5)
  - View with **`getent hosts`** or host(1)
- Local host is 127.0.0.1 (IPv4) and ::1 (IPv6)

Networking: ports

- Each host can offer services on ports, numbered 1 through 65535
- Names are more convenient than numbers, so they are mapped to services using a services file and network resources
  - The services table is stored in file /etc/services and/or is available from network servers
  - View with **`getent services`**
- Examples of ports include:
  - 22 for ssh
  - 80 for http
  - 443 for https

Networking: protocols

- A protocol is a standardized arrangement for data
- Internet Protocol (IP) forms the foundation for the Internet by supporting addressing and routing
- Transmission Control Protocol (TCP) provides reliable connection-oriented message communication using IP
- User Datagram Protocol provides datagram communication using IP
  - connectionless, unreliable
  - A datagram includes a header and payload (data)

Web file retrieval

- curl(1) and wget(1) retrieve web content

```
curl -O url
```

```
wget url
```

- w3m(1) and lynx(1) can be useful for quick viewing of a web page or html file

ssh

- ssh runs a shell on a remote machine using an encrypted connection
- ssh is safest to use with public-private key pair
  - ssh-keygen(1) can generate the key pair
  - Some installations disable use of passwords
  - Some installations disable remote logins by root
- Two options are useful, but only use on trusted networks:
  - -X option forwards X11 session, allowing remote windows
  - -A option forwards authentication agent (ssh-agent) connection
- scp can copy files and directories to/from remote machines; it is useful when rsync is not installed

ssh-keygen

- Example usage:

  ssh-keygen -t rsa [-f *keyfile*]

- Keys are stored in ~/.ssh directory, which should not be accessible by others

  chmod go-rwx ~/.ssh

- it's good practice to create a separate key pair for each group of remote hosts that you use
- Never allow anyone else to access any of your private keys
- Public keys may be provided to others. If added to the ~/.ssh/authorized_keys file on a remote machine, you will be able to access the account on that machine without a password

Networking tools, basic

- ping(1) can tell you if you can reach a host
  - Unless the host has disabled ping responses
- traceroute(1) can show you the path to a host
- tcpdump(1) captures network packets
  - Should specify the network interface or **`any`**
  - Has many options to, e.g., write or read captures to a file, filter packets by host or protocol, display numbers instead of names, display the packet contents. A simple example is:

```
tcpdump -i any -s0 -w all_traffic.pcap not arp
```

Networking tools, for testing

- telnet(1) attempts to open a TCP connection to a host
  - To see if sshd is running on a specific host:

    ```
    telnet 10.0.0.2 22
    ```
- nc(1)/ncat(1)/netcat(1) create a trivial TCP or UDP client or server
  - Input/output can be redirected from/to a file
  - Example UDP server/client pair, in separate windows or on separate machines:

    ```
    nc -lu 5500
    ```

    ```
    nc -u localhost 5500
    ```

    - To terminate, Ctrl-d in client and Ctrl-c n server

Networking configuration

- ifconfig(1) lists all (active) network interfaces
  - -a option adds inactive interfaces
- To list all Internet addresses of the host:

  `ifconfig | egrep 'inet '`
- To bring an interface up:

  `ifup` ***`ifname`***
- To bring an interface down:

  `ifdown` ***`ifname`***
- To list all routes from the host:

  `route`

  or

  `netstat -rn`

System administration

back to Contents

User accounts

- Every account has a name, sometimes called a *login*
- The /etc/passwd file, and/or a directory service, maintains the accounts on a system
  - getent(1) shows account information

    `getent passwd [`*`name`*` ...]`
- Each account can be a member of any number of groups
  - getent(1) shows the members of a group

    `getent group [group`*`name`*` …]`
- Internally, accounts and groups are represented by numbers
- id(1p) shows the numbers, and group memberships of any account

The superuser and su command

- Unix and Linux systems have a privileged user account
  - referred to as the superuser
  - login (account) name is **root**
  - can perform any operation on the system
    - strictly minimize use, and use carefully
- The su command allows a user to substitute as another user, including the superuser.
  - requires the user, other than the superuser, to know the password of the substitute user account
  - superuser can substitute as any other user without knowing their password
  - su discouraged in favor of sudo

sudo command

- The sudo (superuser do) command is the preferred mechanism to execute a command as another user, including the superuser.
  - does not require password of substitute user account
  - does require setup
  - tracks usage in, typically, /var/log/secure
  - To run a command as superuser, just prepend with sudo, e.g.,

    ```
    sudo less /var/log/secure
    ```
- Setup is via files added to /etc/sudoers.d
  - edit them using sudoedit, for safety

RPM package management: rpm

- Red Hat-based systems use rpm and dnf for package management. rpm can refer to:
  1. the rpm (RPM package manager) system
  2. the rpm program
  3. an actual package file
- All software on the system is packaged into package files, which have a .rpm extension.
  - rpm files are stored in cpio format, if low-level manipulation is needed
  - a software package may comprise multiple rpms, e.g.,
    - bzip2-1.0.6-28.fc29.x86_64
    - bzip2-devel-1.0.6-28.fc29.x86_64
    - bzip2-libs-1.0.6-28.fc29.x86_64
    - bzip2-static-1.0.6-28.fc29.x86_64

RPM package management: rpm

- The rpm progam can be be used to install, remove, or upgrade packages on a system. It can also be used to query installed packages and to query an rpm file. E.g.,

  `rpm -ivh` ***`rpm-file`***

  `rpm -e` ***`package`***

  `rpm -q` ***`package`***

- rpms are built from source rpm packages
  - the rpm-build program can be used for the build
  - source rpm files have .srpm file extensions
- The rpm program should usually not be used directly
  - instead, use dnf

rpm examples

- show the info of an rpm

```
rpm -qip foo.rpm
```

- list the files that would be installed by an rpm

```
rpm -qlp foo.rpm
```

- list the dependencies of an rpm

```
rpm -qp -R foo.rpm
```

- For each of the above, remove the `p` to query an installed package
- list the actual contents of an rpm

```
rpm2cpio foo.rpm | cpio -it
```

- extract /etc/bar.cnf from an rpm

```
rpm2cpio foo.rpm | cpio -imdv ./etc/bar.cnf
```

rpm exercises

- List all of the rpms installed on a system

  solve exercise before proceeding

- List the information about, and files contained in, the installed **words** rpm

- List the information about the package that contains the **w** command

rpm exercises

- List all of the rpms installed on a system

  `rpm -qa`

- List the information about, and files contained in, the installed **words** rpm

  solve exercise before proceeding

- List the information about the package that contains the **w** command

rpm exercises

- List all of the rpms installed on a system

  `rpm -qa`

- List the information about, and files contained in, the installed **words** rpm

  `rpm -qil words`

- List the information about the package that contains the **w** command

  solve exercise before proceeding

rpm exercises

- List all of the rpms installed on a system

  `rpm -qa`

- List the information about, and files contained in, the installed **words** rpm

  `rpm -qil words`

- List the information about the package that contains the **w** command

  `rpm -qif $(type -p w)`

RPM package management: dnf

- To address deficiencies of rpm, especially with package dependencies, yum (Yellowdog Updater, Modified) was developed
- dnf (dandified yum) has replaced yum, with the same subcommands

```
dnf install package
dnf localinstall rpm-file
dnf upgrade package
dnf remove package
dnf whatprovides glob
```

dnf exercises

- What does this command do?

  `dnf whatprovides *bin/valgrind`

  solve exercise before proceeding

dnf exercises

- What does this command do?

```
dnf whatprovides *bin/valgrind
```

It lists all of the packages that provide a valgrind program, e.g.,

valgrind-1:3.14.0-7.fc29.x86_64 : Tool for finding memory management bugs in programs

Repo : updates

Matched from:

Other : *bin/valgrind

Service control with service(1)

- System V-based systems, including RedHat 6 and earlier, use System V init scripts to control system services
  - Scripts in /etc/rc.d/ subdirectories are used to start, stop, and check the status of each service
- Init scripts are run at system startup to start services in proper order
- After startup, an init script can be used at any time, e.g.,

```
/etc/rc.d/init.d/network status
/etc/rc.d/init.d/network restart
```

- The service command is the preferred way to run init scripts

```
service network status
service network restart
```

- chkconfig(1) can be used to manage the scripts
  - add, del, on, off, etc.

Service control with systemctl(1)

- RedHat 7 and later use systemctl(1) to manage services
  - systemd(1) manages the services
  - systemd is the first running process, so it has pid 1
- systemctl controls systemd, e.g.,

```
systemctl status network.target
systemctl restart network.target
```

- To list all services:

```
systemctl list-units
```

- systemd logs events to a journal, which can be queried using journalctl(1)

systemd

- The first process launched after boot is systemd(1)
  - All other *units* are *activated* by systemd
  - See systemctl(1), including its list-units command
- systemd replaces the System V init(1) command
  - init uses the concept of runlevels, which can vary but typically include:
    - 0: power off
    - 1: single user, without network
    - 2: multi-user without network
    - 3: multi-user
    - 5: multi-user with display manager, such as using X server
    - 6: reboot
  - telinit(1) changes runlevel

fork and exec

- All programs are launched using the fork and execve system calls
- fork(3p) creates a new *child* process that is an exact copy of the *parent* process, but with a different process ID (pid)
- execve(3p) executes a program
  - The C library contains wrappers such as exec(3), execl(3), and execv(3)
    - support different arrangements of program arguments
    - support PATH searching
    - support provision environment
- pstree(1), ps -eH, and ps axf show how all user processes descend from systemd

Name service configuration

- /etc/nsswitch.conf(5) configures each name service
- For example, this entry directs passwd information to be provided by sss (System Security Services Daemon) and files (/etc/passwd), in that order:

```
passwd:  sss files
```

- Another example is this entry, which provides host information from files (/etc/hosts) and DNS, in that order:

```
hosts:  files dns
```

- getent(1) obeys nsswitch.conf, providing quick access to the information provided by each name service

Name service exercises

- How can you view the passwd information of your own login?

  solve exercise before proceeding

- How can you view all (both IPv4 and IPv6) of the information about your local host?

Name service exercises

- How can you view the passwd information of your own login?

```
getent passwd $(id -u)
```

- How can you view all (both IPv4 and IPv6) of the information about your local host?

solve exercise before proceeding

Name service exercises

- How can you view the passwd information of your own login?

```
getent passwd $(id -u)
```

- How can you view all (both IPv4 and IPv6) of the information about your local host?

```
getent hosts | grep localhost
```

or

```
getent hosts localhost ::1
```

Shutdown

- For systems with poweroff(1) and reboot(1), they can be used from the command line to turn off or reboot, respectively
- Through the GUI, there should be shutdown and reboot selections
- The GUI may have a power button
  - red
  - with a 0/1 indication
- Older systems use the shutdown(1) command

```
shutdown -h now
```

Software development tools

back to Contents

Software build

- A traditional software build process includes configure, build, test, install
  - configure can be used to customize install directories, compile and link options, etc.
  - make runs the compile and link
  - make *targets* are typically used to test, install, clean up, etc.
- On one line:

```
./configure && make && make test && make install
```

Software analysis tools

- For dynamic (run-time) testing of native binaries, valgrind is indispensable
  - just prepend the invocation with valgrind
  - valgrind can significantly slow down execution, so it's not suitable for real-time programs
- Run-time profiling can help isolate performance bottlenecks
  1. Compile and link with -pg
  2. After execution, analysis with gprof
- http://brendangregg.com/linuxperf.html

Debugging symbols

- Programs must be compiled with an option, such as -g, to include debugging symbols in the linked file
- strip(1p) removes the debugging symbols, if desired at a later time
- size(1) shows the size of the program, excluding debugging symbols
- A debugger such as gdb(1) can run the program under precise control
  - breakpoints, to stop execution at specified points such as functions or source code locations
  - watchpoints, to stop execution on specified conditions such as variable values

Execution tracing

- strace(1) shows the system calls executed by a process
  - For example, to show all openat(3p) calls executed by ls(1p):

    ```
    strace -e trace=openat ls
    ```
- ltrace(1) shows the library calls executed by a process
- Both put their output on the standard error stream

Symbols

- nm(1p) shows the symbols in a binary object file
  - An object file is compiled code
    - Includes linked objects, which results in an executable binary file
      - excludes shell scripts
  - Symbols include names of data and functions
    - These names may be *mangled*, i.e., decorated with additional information such as function parameter types
      - Functions were originally named without any of their arguments, preventing overloading

Source code (version) control

- Source code control systems manage all **versions** of files and, with some systems, directories
  - support retrieval of any version at any time
  - support comparison of versions of text files
- Useful for more than code, such as documents, spreadsheets, and any other files that change
- Some support simultaneous updates by multiple uses
- Usually view version history as a tree with multiple branches
  - merge changes from one branch to another
  - typically one branch has designation such as main or master

git

- Developed in 2005 by Linus Torvalds in response to restrictive licensing of existing systems
- Pros: popular, powerful, free and open source, geared to multiuser environments, supports off-line use
- Cons: steep learning curve
- Githubs are commonly used for software distribution
  - e.g., https://www.github.com/

Basic git commands

- git init
  - one-time setup of local git repository
- git add
  - stage files/directories for commit to local repository into an *index*
- git commit
  - commit the index
- git pull/push
  - synchronize local repository with remote repository
- git status
  - view state of index
- git log/whatchanged
  - view history of local repository
- git diff
  - view changes, such as uncommitted changes in workspace

git bisect

- Bisect is an invaluable capability
  - Narrows down change in behavior (bug) to single commit
- Usage: initialize, then repeatedly checkout and test a version between known good and bad points

```
git bisect init
git bisect good
```

or

```
git bisect bad
```

- Can streamline with git run if a test program is provided
- To finish:

```
git bisect reset
```

User versus kernel space

- Memory regions that are in use by the kernel is referred to as *kernel space*
- Memory regions that have been allocated to a user program are referred to as *user space*
  - By default, memory allocated to one program cannot be accessed by other programs
  - Memory can optionally be *shared* with other programs

How do applications access the OS?

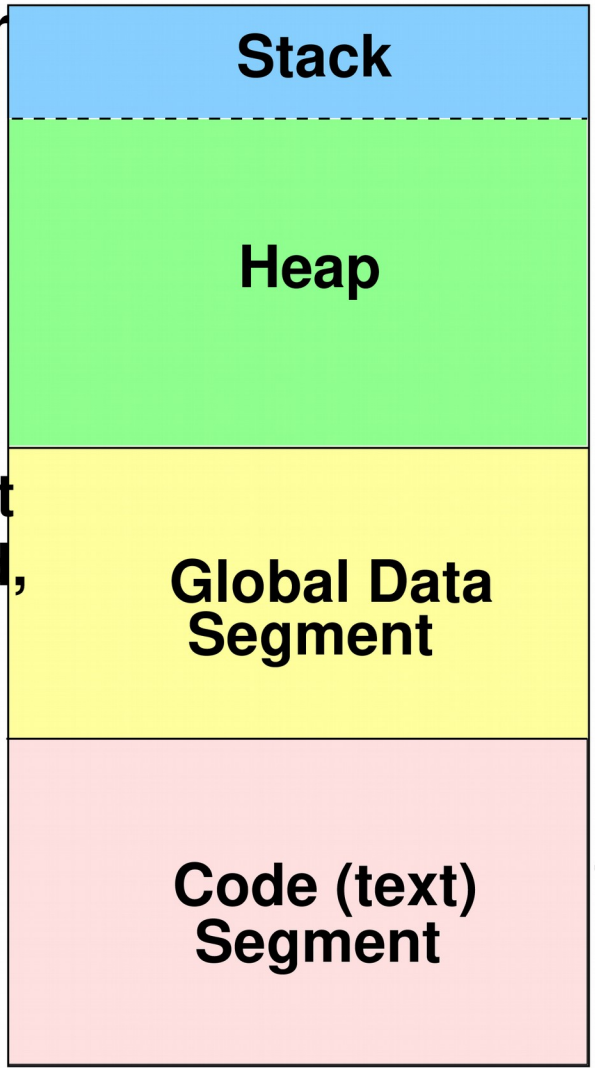
- The OS interface is accessed through *system calls*
  - Applications invoke a system call just like any other function call
  - Each system call is documented in a man page, in section 2 of the system manual
  - Example: the unlink system call removes a (file or directory) name from a filesystem
    - Its signature is **`int unlink (const char *)`**
    - Its one argument is the name and it returns a integer status
    - If the name has no more links and doesn't refer to a file that is opened by a running process, the file is removed

System call return values

- System calls often return an integer
  - Called a return value or status
- 0 indicates success
  - If a request was made, then it was performed successfully
- Non-0 indicates failure
  - The man page describes possible values and their meanings
  - The global variable **`errno`** is often set to provide further information on the failure
    - The man page for the particular system call lists the possible values
    - The errno.h(0p) man page lists the usual meanings for each value

Program memory layout

memory managed by compiler: used for each function call



function params, local objects

memory managed explicitly by application (new/ delete)

dynamically allocated objects

global data: object locations are fixed, but their values can be modified

global objects

code: read-only

function definitions