Linux command line
&
Bash Scripting

Table of content

What we won’t be covering today

- Distributions
- Disks/Partitions/Filesystems
- Security/firewall
- Networking/server
- Archiving/compressing
- Text processing
- Customization/ricing

Customization / Ricing

- https://www.reddit.com/r/unixporn/comments/utepks/kde_solarized_light_didnt_expect_to_enjoy_a_light/
- https://www.reddit.com/r/unixporn/comments/q941ym/awesomewm_life_problems/
- https://www.reddit.com/r/unixporn/comments/m5522z/grub2_had_some_fun_with_grub/
- https://www.reddit.com/r/unixporn/comments/mwvw15/plasma_kwin_krohnkite_ghibli_from_boot_till/
- https://www.reddit.com/r/unixporn/comments/vmdbjs/bspwm_cooking_rice_on_the_zenbook/

Part 1:
Linux Basics

Basics

- *username@machinename* followed by current working directory
  - E.g
- $
  - Represents a regular user
- #
  - Represents a superuser
  - Or commands that should be ran with superuser privileges using sudo

Shell vs terminal emulators

- A shell is a user interface that passes commands to the operating system to carry out.
  - E.g Bash, zsh, powershell, fish, eshell

- Terminal emulators instead are the programs we use to interact with the shell. They open up a graphical for us to interact with the shell.
  - E.g Konsole, gnome-terminal, alacritty, xterm, iTerm2, PuTTY, windows terminal

Login shells vs non-login shells

- A login shell is the first process that executes under our user ID when we log in for an interactive session.
  - Typically source from
    - /etc/profile
    - ~/.bash_profile
    - ~/.bash_login
    - ~/.profile
- When we start a shell in a terminal in some existing session or in the GUI, we get a non-login shell.
  - Typically source from
    - /etc/bash.bashrc
    - ~/.bashrc

Login shells vs non-login shells

- To determine whether the current shell is a login shell or a non-login shell, run the following command
  - echo $0
- If you see -bash,
  - Then it’s a login shell
- If not,
  - Then it’s usually a non-login shell
- Another option for bash is to run the command
  - shopt login shell
    - login_shell on
    - login_shell off

Login shells vs non-login shells

- Non-login shells inherits from their parent process, typically a login shell so environment variables from the login shells normally get set for non-login shells too.
- It is also common to see the following lines in .bash_profile, which allows the login shell to source from .bashrc as well

```
if [ -f ~/.bashrc ];  then  .  ~/.bashrc;  fi
```

```
[[ -f ~/.bashrc]] && . ~/.bashrc
```

- On a side note, macOS typically starts a new login shell whenever we launch a new terminal. This might cause settings applied to .bashrc files to not apply to the shell session.

History

- Bash stores the last 500 commands by default
- Use arrow keys to view previous commands ran
- Alternatively, run the command history
- !*number*
  - Runs a specific past command
- !!
  - Runs last ran command
- history | grep *regex*
  - Searches for history for the regex passed in
- Ctrl + R
  - Keyboard shortcut to search through history

History

```
 1966  git status
 1967  git push
 1968  git status
 1969  code ../aac-backend/
 1970  vim test.txt
 1971  cat test.txt
 1972  cat test.txt | grep [a-z]\{4,8\}
 1973  cat test.txt | grep '[a-z]\{4,8\}'
 1974  cd cvwo/aac-frontend/
 1975  git branch
 1976  git status
 1977  git pull
 1978  git switch bryan/activity-session-registration-show/hide
 1979  git pull
 1980  git status
 1981  git merge master
 1982  code .
 1983  git abort
 1984  git commit --abort
 1985  git merge --abort
 1986  git status
 1987  git switch master
 1988  cd cvwo/aac-frontend/
 1989  yarn start
 1990  cd cvwo/aac-backend/
 1991  code .
 1992  make run
 1993  sudo service postgresql start
 1994  make run
 1995  cd cvwo/aac-checkcall-frontend/
 1996  git status
 1997  git branch
 1998  git switch checkcall-team/add-state
 1999  git switch checkcall-team/add-loading
 2000  history
bryan@DESKTOP-1PJF7VI:~$
```

Standard linux directories

- /
  - All files appear under single hierarchy
- /bin
  - User binaries
- /boot
  - Contains kernel, initial Ram disk image and bootloader
- /dev
  - Device nodes, linux philosophy of “everything is a file”
- /etc
  - System-wide config files
    - /etc/crontab
    - /etc/fstab
    - /etc/passwd

Standard linux directories

- /home
- /mnt
  - Typically present in older systems
  - Mount points for removable devices, though nowadays removable devices or printers are usually automatically mounted
- /usr/local
  - Programs for system wide use but not provided as part of the base installation from the distribution
- /usr/share
  - Shared data for stuff in /usr/share, e.g config files, icons etc
- /var
  - Stores data that is likely to change, e.g cache, db, mail, log files etc

Pkg management

- 2 main packaging systems
  - Debian-based distros such as Ubuntu, Linux Mint
    - .deb
  - Red Hat related distros such as Fedora, CentOS, openSUSE
    - .rpm
- Others
  - Arch-based distros
  - Distros that compile from source such as gentoo

Pkg management

- Package managers
  - yum
  - apt
  - pacman
- Makes things easy for installing and upgrading packages, helps prevent “dependency hell”

Other software installation methods

- AppImage
  - Typically fastest and smallest in size
  - However, least frequent updates and not as many packages
- Flatpak
  - Runs apps in a secure virtual sandbox that doesn't require root privileges
  - Typically biggest in size however
- Snap
  - Initially developed by Canonical for Ubuntu
  - Most packages compared to the other 2
  - However, bigger size & slower startup
- Docker

Basic commands

- ls
- pwd
- cat
- cd
- cp
- echo
- mv
- mkdir
- rm
- ln
- touch
- head, tail, less

Options & Arguments

- *Command -options arguments*

- Options can be either a single char preceded by a dash or a word preceded by 2 dashes
    - When using the ‘short’ options, we can chain multiple options together
    - Typically case sensitive

Options & Arguments

- ls --all
- ls -a

- ls -l

- ls -lah
- ls -alh
- ls -hal

ls

- -a, --all
  - Also lists hidden files
- -h, --human-readable
  - Display file sizes in readable format rather than bytes
- -S
  - Sort by file size
- -t
  - Sort by modification time

ls -l

```
bryan@DESKTOP-1PJF7VI:~/cvwo/aac-frontend$ ls -l
total 580
-rw-r--r--   1 bryan bryan   1493 May  9 16:59 README.md
drwxr-xr-x 928 bryan bryan  36864 May 20 09:03 node_modules
-rw-r--r--   1 bryan bryan   2859 Jun 22 11:54 package.json
drwxr-xr-x   4 bryan bryan   4096 May  9 16:59 public
drwxr-xr-x   9 bryan bryan   4096 May 27 16:05 src
-rw-r--r--   1 bryan bryan    553 May  9 16:59 tsconfig.json
-rw-r--r--   1 bryan bryan 529005 Jun 21 23:45 yarn.lock
bryan@DESKTOP-1PJF7VI:~/cvwo/aac-frontend$
```

ls -l

- Access rights
- No. of hard links
- Username of file owner
- Name of group that owns the file
- Size of file in bytes
- Date and time since last modification
- Name of file

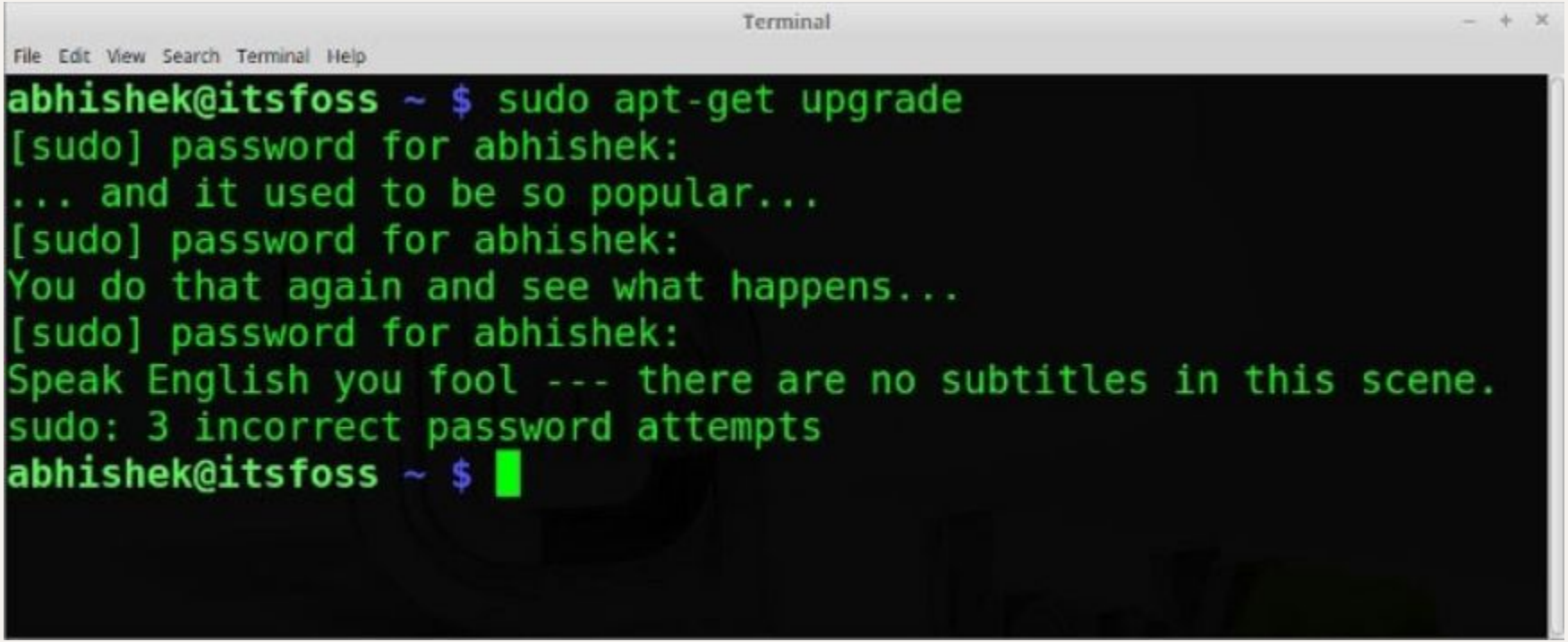
sudoers

```
#
# This file MUST be edited with the 'visudo' command as root.
#
# Please consider adding local content in /etc/sudoers.d/ instead of
# directly modifying this file.
#
# See the man page for details on how to write a sudoers file.
#
Defaults        env_reset
Defaults        mail_badpass
Defaults        secure_path="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/snap/bin"

# Host alias specification

# User alias specification

# Cmnd alias specification

# User privilege specification
root    ALL=(ALL:ALL) ALL

# Members of the admin group may gain root privileges
%admin ALL=(ALL) ALL

# Allow members of group sudo to execute any command
%sudo   ALL=(ALL:ALL) ALL

# See sudoers(5) for more information on "#include" directives:

#includedir /etc/sudoers.d
~
~
~
~
~
~
~
~
~
"/etc/sudoers" [readonly] 30L, 755C                                    1,1           All
```

sudoers

```
Cmnd_Alias  POWER       =   /usr/bin/shutdown -h now, /usr/bin/halt, /usr/bin/poweroff, /usr/bin/reboot
Cmnd_Alias  STORAGE     =   /usr/bin/mount -o nosuid\,nodev\,noexec, /usr/bin/umount
Cmnd_Alias  SYSTEMD     =   /usr/bin/journalctl, /usr/bin/systemctl
Cmnd_Alias  KILL        =   /usr/bin/kill, /usr/bin/killall
Cmnd_Alias  PKGMAN      =   /usr/bin/pacman
Cmnd_Alias  NETWORK     =   /usr/bin/netctl
Cmnd_Alias  FIREWALL    =   /usr/bin/iptables, /usr/bin/ip6tables
Cmnd_Alias  SHELL       =   /usr/bin/zsh, /usr/bin/bash
%power      ALL         =   (root)  NOPASSWD: POWER
%network    ALL         =   (root)  NETWORK
%storage    ALL         =   (root)  STORAGE
root        ALL         =   (ALL)   ALL
admin       ALL         =   (root)  SYSTEMD, KILL, FIREWALL
devel       ALL         =   (root)  PKGMAN
joe         ALL         =   (devel) SHELL, (admin) SHELL
```

sudoers (enable insults)



less

- PAGE UP or b
  - Scroll back
- PAGE DOWN or spacebar
  - Scroll forward
- h
  - Help
- q
  - Quit

Redirection

- Redirecting stdin and stdout
  - *Command < input_file > output_file*
    - > overwrites
    - >> to append
  - E.g
    - java *program_name < input_file > output_file*
    - cat movie.mpeg.0* > movie.mpeg
- Redirecting stderr
  - ls -l /bin/usr &> ls_output.txt
  - ls -l /bin/usr > ls_output.txt 2>&1

Redirection

- Piping
  - *command 1* | *command 2*
  - E.g
    - ls -l /usr/bin | less
    - ls -l /usr/bin | sort | uniq | less
    - history | grep ssh

Searching files

- locate *file_name*
- find
  - -type
    - f (file), d (dir), l (sym link)
  - -name *regex*
    - -iname regex (case insensitive name)
  - -size
  - -empty
    - Match empty files & dir
  - -user
- Search for broken sym links
  - find / -xtype l -print

Searching files

- find
  - Logical operator
    - -and, -or, -not
    - (*condition*)
  - E.g searching in ~ for all files with permissions not 0600 and dir with permissions not 0700, and size at least 100mb
    - find ~ \(-type f -not -perm 0600 -size +100M \) -or \(-type d -not -perm 0700 -size +100M \)

Searching files

- find
  - Actions
    - -delete
    - -ls
    - -print
    - -quit
    - -exec *command {} ;*
  - E.g
    - find ~ -type d -name cvwo
    - find ~ -type d -name cvwo -ls
    - find ~ -type d -name cvwo -exec ls -l '{}' ';'

Symbolic links

- Soft links
  - Similar to windows shortcuts
  - Text pointer to referenced file or directory
- Hard links
  - Indistinguishable from the file itself
  - When hard links are deleted, content of file continues to exist until all hard links are deleted

ln

- Hard link
  - ln *file link*
- Soft link
  - ln -s *item link*
    - Item can be a file or directory
  - When deleting the file/directory before the link, the link itself isn't removed, in which case we get broken links

Processes

- Ps
  - By itself shows only processes associated with current terminal session
  - ps x (shows all processes)
- Top & htop
- Bg & Fg
  - *Command* &
  - fg %*job_number*
- Jobs
  - List jobs that have been launched from the terminal

Processes

- kill [-*signal*] *PID*
- Kill Signals
  - -INT (Same as Ctrl+C)
  - -KILL (last resort, kernel immediately terminates process)
  - Must be owner of process to kill
- killall

Part 2:
Regular Expression

Wildcards (Filename globbing)

- *
  - Matches any char sequence
- ?
  - Matches any single char
- [*characters*]
  - Matches any char specified in the set of *characters*
- [!*characters*]
  - Any char that is not a member of the set of *characters*
- Character ranges
  - [A-Z]
  - [A-Za-z0-9_]
  - [!ABC]

Wildcards

- [[:*class*:]]
  - Matches any char that is a member of the class
- Common classes
  - [:alpha:]
  - [:alnum:]
  - [:digit:]
  - [:lower:]
  - [:upper:]

Wildcards

- E.g
  - b*.txt
  - [abc]*
  - BACKUP.[0-9][0-9][0-9]
- Exercises: Using ls
  - In aac-frontend/src/pages/
    - List all files in directories that start with a
  - In aac-frontend/src/pages/activities
    - List all files related to Sessions
    - List only ActivitySessionRegistration.tsx and ActivitySessionBatchRegistration.tsx

Wildcards

- In aac-frontend/src/pages/
  - List all files in directories that start with a
    - ls a*
- In aac-frontend/src/pages/activities
  - List all files related to Sessions
    - ls *Session*
  - List only ActivitySessionRegistration.tsx and ActivitySessionBatchRegistration.tsx
    - ls *Session[RB]*n.tsx
    - ls *Session*n.tsx

Regex

- .
  - Matches any char
- [*characters*]
  - Matches any char specified in characters
- [^*characters*]
- Character ranges
  - [A-Z]
- ^
  - Start of line
- $
  - End of line
- <...>
  - Match words

Regex

- Exercise:
  - Match the following patterns:
    - cat.
    - 896.
    - ?=+.
  - Exclude/skip the following patterns:
    - abc1
  - Hint: you may need to escape . with \ to specifically match period

Regex

- Exercise:
  - Match the following patterns:
    - cat.
    - 896.
    - ?=+.
  - Exclude/skip the following patterns:
    - abc1
  - Hint: you may need to escape . with \ to specifically match period
- Solution:
  - ...\.

Regex

- Exercise:
  - Match the following patterns:
    - can
    - fan
    - tan
  - Exclude/skip the following patterns:
    - dan
    - ran
    - pan

Regex

- Exercise:
  - Match the following patterns:
    - can
    - fan
    - tan
  - Exclude/skip the following patterns:
    - dan
    - ran
    - pan
- Solution:
  - [cft]an
  - [^drp]an

Regex

- Special characters
  - \d
    - Any digit
  - \D
    - Any non-digit
  - \s
    - Any whitespace
  - \S
    - Any non-whitespace
  - \w
    - Any alphanumeric

Repetitions

- {m, n}
  - Repeats at least m times, at most n times
  - waz{3,5}up
    - wazzzup
    - wazzzzup
    - wazzzzzup
  - {1,}

Repetitions

- +
  - 1 or more repetition
- *
  - 0 or more repetition
- ?
  - Optional char
- E.g
  - a*
    - Zero or more a
  - [abc]+
    - One or more a/b/c

Repetitions

- Exercise:
  - Match the following patterns:
    - aaaabcc
    - aabbbbc
    - aacc
  - Exclude/skip the following patterns:
    - a

Repetitions

- Exercise:
  - Match the following patterns:
    - aaaabcc
    - aabbbbc
    - aacc
  - Exclude/skip the following patterns:
    - a
- Solution:
  - aa+b*c+
  - a{2,4}b{0,4}c{1,2}

Repetitions

- Exercise:
  - Match the following patterns:
    - 1 file found?
    - 2 files found?
    - 24 files found?
  - Exclude/skip the following patterns:
    - No files found.

Repetitions

- Exercise:
  - Match the following patterns:
    - 1 file found?
    - 2 files found?
    - 24 files found?
  - Exclude/skip the following patterns:
    - No files found.
- Solution:
  - \d+ files? found\?

Repetitions

- Exercise:
  - Match the following patterns:
    - IMG01.png
    - IMG02.png
    - ...
    - IMG99999999999.png
  - Exclude/skip the following patterns:
    - IMG01.jpeg
    - FANCYIMG01.png

Repetitions

- Exercise:
  - Match the following patterns:
    - IMG01.png
    - IMG02.png
    - ...
    - IMG99999999999.png
  - Exclude/skip the following patterns:
    - IMG01.jpeg
    - FANCYIMG01.png
- Solution:
  - ^IMG\d+\.png$

Capture groups/match groups

- Extract info for further processing by defining groups of characters and capturing them within parentheses
  - Back references, can store up to 9
  - Recall the references/stored pattern with *number*
- (...)
  - \([a-z]\)\([a-z]\)[a-z]\2\1
    - Matches a 5 letter palindrome
  - ^(IMG\d+)\.png$
  - ^(file.+)\.pdf$
- Nested subgroups
  - ^(IMG(\d+))\.png$
- Conditionals
  - (cats*|dogs?)

Grep

- grep [*options*] *regex* [*file*...]
  - -i
    - Case insensitive
  - -c
    - Count no of matches
  - -l
    - Print name of each file that contains a match
  - -n
    - Prefix each matching line with line number
  - -v
    - Display lines that match the regex

Grep

- Only 2 lines in test.txt
  - a\{4,8\}
  - aaaaa
- E.g
  - cat test.txt | grep '[a-z]\{4,8\}'
  - cat test.txt | grep '[a-z]\\{4,8\\}'
  - history | grep ssh

Sed

- sed 's/oldtext/newtext/' *filename* > *output_file*
  - /g
    - Replace all occurrences of the pattern in a single line instead of just the first occurrence
  - /p
    - Duplicate the line after replacing
  - /d
    - Delete the line
  - /I or /i
    - Case insensitive
- sed -i 's/oldtext/newtext/' *filename*
    - To override/save to the file we are working on

Sed

- sed -f *sed_file* < *filename* > *output_file*
  - Runs sed commands from a file instead
- sed -e 's/oldtext/newtext/' 's/oldtext/newtext/' < *filename* > *output_file*
  - To run multiple sed commands

Sed

- E.g
  - sed 's/\/usr\/local\/bin/\/common\/bin/' < *old_file* > *new_file*
    - Changes /usr/local/bin to /common/bin
- Using & as the matched pattern
  - echo "123 abc" | sed 's/[0-9]*/& &/'
    - Output: 123 123 abc
  - sed 's/[a-zA-Z]* /test/2g' < *old_file* > *new_file*
    - Skips the first occurrence of the pattern and modifies all occurrence from the 2nd onwards
  - sed 's/ [[:digit:]]\+ / /g'
    - Deletes space-delimited numbers

Sed

- Address and range
  - sed -n '5,10p' *myfile.txt*
    - Views lines 5-10
  - sed '3 s/[0-9][0-9]*//' < old_*file* > *new_file*
    - Deletes the first number on line 3
  - sed -n -e '5,7p' -e '10,13p' myfile.txt
  - sed '20,35d' myfile.txt
  - sed '30,40 s/version/story/g' myfile.txt`
  - sed '/^#/ s/[0-9][0-9]*//'
    - Matches lines with the first pattern /^#/ then performs the sed command

Sed

- Exercise (Work out what each of the following commands do)
  - sed 's/ */ /g'
  - sed '/^#\|^$\| *#/d' *file_name*
  - echo "/home/bryan/temp/myfile.txt" | sed 's/.*\///'
  - echo "apple,banana" | sed 's/^\(.*\),\(.*\)$/\2,\1/g'
  - sed 's/\([0-9]\{2\}\)\/\([0-9]\{2\}\)\/\([0-9]\{4\}\)$/\3-\1-\2/'
  - ip route show | sed -n '/src/p' | sed -e 's/ */ /g' | cut -d ' ' -f 9

Sed

- Exercise (Work out what each of the following commands do)
  - sed 's/ */ /g'
    - Replace multiple spaces with a single space
  - sed '/^#\|^$\| *#/d' *file_name*
    - Delete comments starting with # and empty lines
  - echo "/home/bryan/temp/myfile.txt" | sed 's/.*\///'
    - Remove the path/dir names and return only the file name
  - echo "apple,banana" | sed 's/^\(.*\),\(.*\)$/\2,\1/g'
    - Swap the order for pairs of words
  - sed 's/\([0-9]\{2\}\)\/\([0-9]\{2\}\)\/\([0-9]\{4\}\)$/\3-\1-\2/'
    - Changes date from MM/DD/YYYY to YYYY-MM-DD
  - ip route show | sed -n '/src/p' | sed -e 's/ */ /g' | cut -d ' ' -f 9

Text processing

- Sort, uniq
- Cut, paste, join
- Comm, diff, patch
  - diff -Naur *old_file new_file* > *diff_file*
  - patch < *diff_file*
- Tr
- Sed
- Aspell

Summary

- Wildcards
- Range
- Repetition
- Capture groups
- grep
- sed
    - Address

Part 3:
Bash Scripting

Bash scripting

3 steps to bash scripting:

1) Write the script
2) Make it executable
3) Place it somewhere the shell can find

Permissions

- Recall ls -l
  - Access rights
  - -rwxrwxrwx
  - Owner permission, group permission, world permission
- chmod and chown
- chmod *who=permissions filename*
  - chmod a=wrx *filename*
- chmod *who+permissions filename*
  - chmod a+x *filename*
- chmod *XXX filename*
  - 001 --x
  - 010 -w-
  - 100 r--
  - chmod 744 *filename* (rwx, r--, r--)

Bash scripting

- File format
  - hello_world.sh

```
#!bin/bash


# This is our first script


echo ‘Hello World!’
```

Bash scripting

- Make it executable
  - chmod +x hello_world.sh

```
#!bin/bash


# This is our first script


echo ‘Hello World!’
```

Bash scripting

- Run it
  - ./hello_world.sh

```
#!bin/bash


# This is our first script


echo ‘Hello World!’
```

Bash scripting

- Why the ./
  - For convenience actually, as long as the shell knows where to find the file it’s fine.
  - E.g /home/bryan/hello_world.sh works
- Shebang (#!)
  - Immediately after the shebang, is the path to the interpreter/program used to run the script
  - Typically we use /bin/bash to comply with posix standards

Bash scripting

- Declaring variables
  - *variable=value*
  - foo=“Hello World!”
  - bar=$(ls -l foo.txt)
  - test=“Variables expansion! $foo”
- To use variable
  - $*name*
  - echo $foo
  - export foo
- E.g
  - filename=“myfile”
  - mv $filename ${filename}2

Bash scripting

- Variable/parameter expansion
  - ${*var:-value*}
    - Expands to *value if var is unset/empty*
  - ${*var#pattern}*
    - Chops shortest match for pattern from front of var
  - ${*var##pattern}*
    - Chops longest match
  - ${*var%pattern}*
    - Chops shortest match from end of var
  - ${*var%%pattern}*
    - Chops longest match

Bash scripting

- Variable/parameter expansion
  - E.g
    - this=”Example”
    - THIS=${this:-”NOTSET”}
    - THAT=${that:-”NOTSET”}
  - E.g
    - longfilename=”home/bryan/cvwo/aac-frontend/package.json”
    - file=${longfilename##*/}
      - Returns package.json
    - dir=${longfilename%/*}
      - Returns /home/bryan/cvwo/aac-frontend

Bash scripting

- Variable/parameter expansion
  - E.g
    - MYSTRING="Be liberal in what you accept, and conservative in what you send"
    - ${MYSTRING#*in}
      - what you accept, and conservative in what you send
    - ${MYSTRING##*in}
      - what you send

Bash scripting

- Special Variables
  - $0 - The name of the Bash script
  - $1 - $9 - The first 9 arguments to the Bash script (passed in the command line)
  - $# - How many arguments were passed to the Bash script
  - $@ - All the arguments supplied to the Bash script
  - $? - The exit status of the most recently run process
  - $(date +“%x %r %Z”)
    - 06/30/22 12:06:06 AM +08

Bash scripting

- Special Variables
  - $$ - The process ID of the current script
  - $USER - The username of the user running the script
  - $HOSTNAME - The hostname of the machine the script is running on
  - $SECONDS - The number of seconds since the script was started
  - $RANDOM - Returns a different random number each time is it referred to
  - $LINENO - Returns the current line number in the Bash script

Bash scripting

- Functions

```
function name {
        commands
        return
}
```

```
name() {
        commands
        return
}
```

```
function hello_world {
        echo “Hello World!”
        return
}

hello_world
```

```
foo=0 #global variable
funct1 () {
        local foo
        foo=1
        return
}
```

Bash scripting

- Using if

```
if commands; then
        commands
[elif commands; then
        commands...]
[else
        commands...]
fi
```

```
x=5
if [ $x = 5 ]; then
        echo “x equals 5.”
else
        echo “x does not equal 5.”
fi
```

Bash scripting

- Most commonly used commands are
  - test *expression*
  - [*expression*]
- File expressions
  - file1 -ef file2
    - file1 and file2 have the same inode numbers
  - -d file (file exists and is a dir)
  - -e file (file exists)
  - -f file (file exists and is a file)
  - -r file (file exists and is readable)

Bash scripting

- Most commonly used commands are
  - test *expression*
  - [*expression*]
- String expressions
  - *string* (True if string is not null)
  - -n *string* (True if string length > 0)
  - -z *string* (True if string length is 0)
  - *string1 > string2*
  - *string1 != string2*

Bash scripting

- Most commonly used commands are
  - test *expression*
  - [*expression*]
- Integer expressions
  - *E.g Integer1 -eq Integer2*
  - -ne
  - -le
  - -lt
  - -ge
  - -gt

Bash scripting

- Enhanced replace for test
  - [[*expression*]]
  - Similar to test but supports an additional string expression
    - *String =~ regex*
  - ((*number*))
  - Similar to above but for numbers instead
    - E.g ((*Int* < 0))

Bash scripting

- Logical operators
  - AND
    - -a for test
    - && for [[ ]], (( ))
  - OR
    - -o
    - ||
  - NOT
    - !
    - !

Bash scripting

- read [-*options*] [*variables*...]
  - -d, delimiter
  - -n, no of chars to be read
  - -p, display a prompt message
  - -t *seconds*, timeout after *seconds*
- If no variables present after read, input is read into $REPLY
- E.g
  - read foo
  - read var1 var2 var3 var4 var5
    - read-multiple a b c d e f g

Bash scripting

```
#!bin/bash

# Read Menu

clear
echo “
Please Select:

1. Display System Information
2. Display Disk Space
0. Quit
“


read -p “Enter selection [0-2] > “
```

Bash scripting

```
...
read -p “Enter selection [0-2] > “

if [[ $REPLY =~ ^[0-2]$ ]]; then
        if [[ $REPLY == 0 ]]; then
                echo “Program terminated”
                exit
        fi
        if [[ $REPLY == 1 ]]; then
                commands
                exit
        fi
        if [[ $REPLY == 2 ]]; then
                commands
                exit
        fi
fi
```

Bash scripting

- Branching with case (similar to switch)

```
case word in
        pattern) Commands
esac
```

```
case $REPLY in
        0)      echo “Program terminated”
                exit
                ;;
        1)      commands
                ;;
        2)      commands
                ;;
        *)      echo “Invalid entry” >&2
                exit 1
                ;;
esac
```

Bash scripting

- Looping

```
#!bin/bash

count=1

while [ $count -le 5 ]; do
        echo $count
        count=$((count+1))
done
echo “Finished!”
```

```
#!bin/bash

count=1

until [ $count -gt 5 ]; do
        echo $count
        count=$((count+1))
done
echo “Finished!”
```

Bash scripting

- Looping

```
for variable [in list]; do
        commands
done
```

```
for i in {A..D}; do echo $i; done

for i in sample*.txt; do echo $i; done
```

- Recent versions now have support for C language loops

```
for (( i=0; i<5; i=i+1 )); do
        echo $i
done
```

Bash scripting

- Arrays
  - a[1]=foo
  - echo ${a[1]}
- Using declare
  - declare -a test_array
- Assigning values
  - *name[index]=value*
  - *name=(value1 value2 ...)*
  - *name=([0]=value1 [1]=value2 ...)*

Bash scripting

- Array Operations

```
animals=(“a dog” “a cat” “a fish”)
for i in ${animals[*]}; do echo $i; done

for i in ${animals[@]}; do echo $i; done

for i in “${animals[*]}”; do echo $i; done

for i in “${animals[@]}”; do echo $i; done
```

```
foo=(a b c)
echo ${foo[@]}
foo+=(d e f)
echo ${foo[@]}
```

```
a=(f e d c b a)
a_sorted=($(for i in “${a[@]}”; do echo $i; done | sort))
echo “Sorted array:    ${a_sorted[@]}”
```

Bash scripting

- hours.sh

```
#!bin/bash

usage() {
        echo “usage: $(basename $0) directory” >&2
}

if [[ ! -d $1 ]]; then
        usage
        exit 1
fi

...
```

Bash scripting

```
for i in {0..23}; do hours[i]=0; done

for i in $(stat -c %y “$1”/* | cut -c 12-13); do
        j=${i/#0}
        ((++hours[j]))
        ((++count))
done

echo -e “Hours\tFiles\tHour\tFiles”
echo -e “----\t----\t----\t-----”
for i in {0..11}; do
        j=$((i + 12))
        Printf “%02d\t%d\t%02d\t%d\n” $i ${hours[i]} $j ${hours[j]}
done
Printf “\nTotal files = %d\n” $count
```

Bash scripting

- Summary
  - File format
  - Permissions
  - Variables
  - Functions
  - If else statements, conditionals, case
  - Looping with while, until and for
  - Arrays

Thank you!