**Shell help**

* Delete word: CTRL – W
* Delete line : CTRL – U

**Bash Commands**

* Ssh : login to remote network
  + ssh -IPADDRESS
* ps : displays what shell is running/ what jobs are running
  + kill command.: kills jobs by using PID
    - kill 2458. kills jobs 2458
* --help: helps with command
  + cat –help
* man: system manual
  + man man
  + man write
  + info: newer and more detailed than man
* info: information command
  + h: go through interactive tutorial
  + ? to list info commands
  + SPACE to scroll
  + m followed by name of menu item
  + q to quit
* HOWTO: how to do something
* passwd: change password
* ls: lists directory contents
  + -a: lists all files
  + -l lists detailed
* cat: displays a text file
* rm: deletes a file
  + -i: asks if you want to delte
  + -r: force remove
* hostname: displays name of system
* cp: copies a file
  + cp sourcefile destinatoinfile
  + -i: asks if you want to overwrite
* mv: moves or renames a file
  + mv sourcesfile destionation/new name
  + -i: asks if you want to overwrite
* lpr: print a file
  + -P print a file on a specific printer
    - lpr -Pprinter1 file
  + lpq: print queue
  + lprm: remove from queue
* grep: searches for a string
  + ps -e | grep deluge
* pwd: list current directory
* mkdir: create a directory
* cd: change directory
  + ../ : previous directory
  + ~: home directory
* rmdir: remove directory
* head: wrtie first 10 lines of text in file
* tail: writes last 10 lines of text in file
* find: search for a file
  + find directory qualifier qualifierstring
  + find ~ -name Documents
* Ctrl+C: kill a process
* sudo: substitute user do
  + must use this everytime need root
  + -u: select which user to sub as
* su: super user
  + only use once for super user
* echo: repeats input to output
  + >: overrites all info and replaces with output
  + >>: adds output to last line of file
* cat: prints contents of text files as output
  + >: overrides all info and replaces with output
  + >>: adds output to last line of file
  + <: input into cat
* less: displays only some output allows scroll
* | - pipe. Sends output of one command to input of other
* sort: displays a file in order
  + -u: generates a list where each line is unqie
* uniq: outputs text with duplicate lines removed
* file: displays info on contents of file
* script: records a linux session
* who am I: user info
* unix2dos: converts linux file to windows format
* dos2unix: converts windows file to linux format
* gzip: gunzip. Compress file.
  + Gunzip: extract gzipped file
  + zcat: output gunzipped file
* tar: creates a tar file
  + -c: create
  + -v: verbose: show what is being done
  + -f: write to or read from a file
  + -t: table of contents of tar file
  + -x: extract
* which: locates default command and their path names
* whereis: locates all locations
* apropos: search commands by keywordcd
* w: lists users on the system
* who: lists users on system
  + after login name is file name for screen. Standard output
* write: send a message to user on system
* touch: creates an empty file
* chmod: changes permissions.
  + (+) means add. (-) means remove.
  + o: other than owner
  + a: all
  + u: user (owner)
  + g: group
  + r: read. w: write. x: execute
* noclobber: doesn't allow overwrites
  + set -o noclobber. Enable
  + set +o noclobber: disable
* tr – translate
  + tr abc ABC → abc becomes ABC and dgabc becomes dgABC
* tee – takes standard input and prints it to file and std output
  + who | tee who.out
* fg – moves job to foreground.
  + Fg 1

? - outputs file plus any character

-memo? → memo6 memo3 memoa

\* - wildcard. String plus anthing following.

-memo\* → memorex memo9 memorandum memo.out

[] - limits and specifies mutiple variations of string to show in output

-memo[17a] → memo1 memo7 memoa

-[a-zA-Z] – all lowercase and uppercase letters

-[6-9] – 6,7, 8, 9

Locale – specifies the way lcale-aware programs display certain kinds of data such as times, dates, money and other numeric values, telephone numbers, and measurements.

Locale – diplays the locale variables

-a: all

-v: verbose

-LANG – specifies the local category for categories not specified by an LC\_ variable.

-LC\_ALL – overrides he value of LANG and all other LC\_ variables

-LC\_COLLATE – specifies the collating sequences for the sort utility and for sorting the results of pathname expansion

-LC-CTYPE – specifies how characters are interprested

History – maintains a list of recently issued command lines, called **events**. Quick way to reexecute any events in the list.

-HISTSIZE – determines number of events preserved in the history list during a session.

-HISTFILE – location of history filename

-HISTFILESIZE – max number of events saved between sessions.

-history: the command that lists the history.

-fc -l: lists the 16 most recent commands in a list that includes event numbers

-fc -l 1030 1035: displays events 1030 through 1035

-fc -l view echo: displays event list that begins with most recent command for echo

**File Structure:**

**–--------** drwx-----x -rwxrwxrwx

d: directory user groups other

-: file

* **/: root.**
  + **/boot**: Static files of boot loader. Contains files needed to boot the system
  + **/dev:** Device files. Contains fils that represent peripheral devices. Disk drives, terminals, printers.
  + **/etc:** Machine local system configuration. Holds adminstrative, config, and other system files. /etc/passwd, contains list of all users who have permissoin to use system
    - **/etc/X11:** Machine local config for X Window System.
    - **/etc/opt**: config files for add-on software packages kept in /opt
  + **/home:** Home directory for each user.
    - David
    - John
    - etc
  + **/lib:** Shared libraries and kernel modules.
    - **/lib/modules**: loadable kernel modules
  + **/mnt:** Mount point for temporary mounting filesystems
  + **/opt:** Add on software packages
  + **/proc:** Kernel and process info virtual filesystems
  + **/root**: Home directory for root
  + **/sbin**: Essential system binaries. Utilities used for system admin are stroed in /sbin and /usr/sbin. Utilities needed during the booting process.
  + **/tmp:** Temporary files
  + **/usr:** Second major hierarchy. Traditoinally includes subdirectories that contain info used by the system. Files do not change often and may be shared by multple systems.
    - **/usr/bin:** Most user commands. Contains standard Linux utility programs.
    - **/usr/bin/X11:** Symbolic link to /usr/X11R6/bin
    - **/usr/games:** Games and educatoinal programs
    - **/usr/include:** header files included by C programs
    - **/usr/include/X11:** Symbolic link to /usr/X11R6/include/X11
    - **/usr/lib:** Libraries
    - **/usr/lib/X11**: Symbolic link to /usr/X11R6/lib/X11
    - **/usr/local:** Local hierarchy. Holds locally important files and directories that are added to the system.
    - **/usr/man:** onlines manuals
    - **/usr/sbin:** nonvital system administation binaries. Holds utilities used after the system is up and running
    - **/usr/share:** Architecture independent data.Subdirectories can include dict, doc, games, info, locale, man, misc, terminfo, and zoneinfo.
    - **/usr/share/doc:** Miscellaneous documentation
    - **/usr/share/info:** GNU info system's primary directory
    - **/usr/source:** Source code
    - **/usr/X11R6:** X window system. Revision 6
    - **/var:** Variable data. Files with contents that vary as the system runs. Most common examples are temporary files, system log files, spooled files, and user mailbox files. Can include cache, lib, lock, log, opt, run, spool, tmp, and yp.
      * **/var/log:** Log files. Contains lastlog (record of last login by each user). Messages(System messages from syslogd), and wtmp (a record of all logins/logouts).
      * **/var/spool:** Spooled application data.

**VIM**

ed – line by line text editor for linux

ex – evolution of ed which showed full screen of text

vi – visual mode of ex. Became so popular it got its own command. Q bring ex mode. Quit.

Vim – improved vi. Not meant for im p formatting, such as LibreOffice. Meant for code, short notes.

**:** - moves cursor to bottom of vim

**:q! -**  quits without saving

Command (Normal) Mode – gives vim commands. Can delete text or exit vim.

Input Mode – accepts anything entered as text and prints it to the screen. Esc returns to command mode

Esc – returns to command mode.

I – go to input mode.

A – append after cursor

:set number – numbers each line.

:set nonumber – turns off numbered lines.

:help -

-same shortcuts for shell work in vim. Ctrl-w

-hjkl – work as commands to move the cursor.

u – undo. Keep doing it as much as you want

X – deletes char on cursor.

:redo or ctrl – r – redo.

o – opens a new line, underneath the cursor

O – open a new line over the cursor. Goes to instert mode.

Ctrl-w s – opens a new window editing same filesystems

Ctrl-w n – opens new window for new filesystems

Ctrl-w w – moves cursor to different windows

q – closes windows

ZZ – saves and quits. Also, :wq.

:w *filename* – write to this filename. When vim doesn't specify filename.

:w! - overwrites exisitng file.

Vim -r memo – shows which files were saved in swap before a crash.

Nowrap – tells vim not to wrap text wheh9hhkjhhln lines are too long.

5h or 5l – moves cursor 5 places left or right

fa – find char a. moves cursor to next instance of char a.

Fa – looks backwards around line after last instance is found

w – moves cursor to first letter or next word. W moves by blank delimited words.

b- moves cursor backward to first letter or previous word. B moves by blank delimited words.

( - moves to beginning of previous sentence.

) - moves to beginning of next sentence.

{ - moves to beginning of previous paragraph

} - moves to beginning of next paragraph

H – moves cursor to first line

M – moves to middle line

L – moves to last line

**BASH – Borne Again Shell**

chsh – change shell

BASH and TCSH are command interpreters and high-level programming languages. They process commands you enter on the command line in response to a prompt. When you use shell as a programming language, it processes commands stored in files called **shell scripts**. Have variables and control flow commands.

Sh shell – original Borne shell created by Steve Bourne at AT&T Bell Labs, used in many commercial UNIX systems.

dash shell – Debian Almquist. Bash is 900 kb, has many features. Dash is 100 Kb and offers compatibility with sh scripts, and because of its size can load and execute shell scripts quicker.

Korn shell – ksh. David Korn, added features to sh. Many features of bash, such as aliases and command line editing, are based on ksh features.

POSIX – portable operating system interface. Family of standards developed by PASC (IEEE's Portable Application Standards Committee). Bash is trying to match POSIX compatibility. If use --posix command, then bash will behave close to POSIX requirements.

Shell runs files to initialize itself when it starts.

login shell – first shell that displays a prompt when you log in on a system from the system console or a virtual console, remotely using ssh or another program, or by another means. Terminal is not a login shell cause you are not logging in on a system, but interactive nonlogin shell.

**/etc/profile** – First executed by shell. Su can set up this file. Establishes system wide default characteristics for users running bash. In addition to executing commands it holds, some versions execute the commands within each of the files with a .sh filename extension in **/etc/profile.d** directory. So root user can modify commands profile runs without modifying profile file. Profile can also be changed when system is updates, so profile.d saves changes.

Environmental variables can be set and exported in **/etc/profile** or in .sh file in **/etc/profile.d.**

{**~/.bash\_profile, ~/.bash\_login, ~/.profile**} – shell looks for each in order, executing commands from first ones it finds. Can put commands in these to override commands set in **profile.** By default, a typical linux distribution sets up new accounts with **~/.bash\_profile** and **~/.bashrc** files, which calls **/etc/bashrc**

**~/.bash\_logout** – shell executes commands upon logout.

interactive non login shell – bash. Do not execute the preceding startup files, but inherit login shell variables declared and exported in those files.

**~/.bashrc** can call **/etc/bashrc** – Su can establish systemwide defaults

noninteractive shell – commands in previously described startup files are not executed by noninteractive shells, such as those that run shell scripts. If shells are forked by login shell, they inherit variables that are declared and exported in these startup files.

Noninteractive shells look for environment variable **BASH\_ENV or ENV** and execute commands in file named by this variable.

If [ -f ~/.bashrc ] ; then . ~/.bashrc; fi → if .bashrc exists, execute it.

Example:

$ cat ~/.bash\_profile

if [ -f ~/.bashrc ]; then

. ~/.bashrc

fi

PATH = $PATH:/usr/local/bin

export PS1 = ' [ \h \W \!]\$

First executes commands in ~/.bashrc, second sets environmental PATH variable , and third exports PS1 which controls user's prompt.

**. (dot) or source** – to run shell script or login file or update login file. Changes forever. If you run it wish bash I guess, changes only apply in subshell.

**Symbols**

() - subshell

$() - command subtitution

(()) arithmetic evaluation. Synonym for let.

$(()) - arithmeticexpansion

[] - test commands

[[]] - conditional expression, similar to [] but adds string comparisons

**Redirecting Standard Error**

File Descriptor is the place a program sends its output to and gets its input from.

When you execute a program, the process running the program opens three file descriptors: 0 (standard input), 1 (standard output), and 2 (standard error). > is short for 1> which means redirect standard output. < is short for 0<, which redirects standard input.

If cat is used and file does not exist, it sends output to standard error which is shown on screen.

$ cat x

cat: x: No such file or directory

$ cat y

this is y

$ cat x y

this is y

cat: x: No such file or directory

$ cat x y > hold

cat: x: No such file or directory

$ cat hold

this is y //standard error isn't affected when you redirect standard output

$ cat x y | tr “[a-z]” “[A-Z]”

cat: x: No such file or directory

THIS IS Y //standard output directed through pipe to tr which translates from lowercase to //uppercase

$ cat x y 1> hold 2> hold2

$ cat hold

this is u

$ cat hold2

cat: x: No such file or directory //standard output sent to hold. Standard error sent to hold2

&> - combines standard output and standard error

$ cat x y &> hold

$ cat hold

cat: x: No such file or directory

This is y

2>&1 – file descriptor 2 is a duplicate of file descriptor 1. Put duplicate after redirect to make both go to same place

$ cat x y 1> hold 2>&1

$ cat hold

cat: x: No such file or directory

This is y

$ cat x y 2>&1 | tr “[a-z]” “[A-Z]”

CAT: X: NO SUCH FILE OR DIRECTORY

THIS IS Y

|& - is shorthand for 2>&1 |

$ cat x y |& tr “[a-z]” “[A-Z]”

CAT: X: NO SUCH FILE OR DIRECTORY

THIS IS Y

< filename- redirects standard input from filename

> filename – redirects standard output to filename unless filename xists and noclobber is set.

>| filename – redirects standard output to filename even if file exists and noclobber is set.

>> filename – redirects and appends standard output to filename unless filename exists and noclobber set.

<&m – duplicates standard input from file descriptor m.

[n]>&m – duplicates standard output or file descriptor n I fspecified from file descriptor m

[n]<&- - closes standard input for file descriptor n if specified

[n]>&- - closes standard output for file descriptor n if specified

Pipes also take standard output not standard error.

Shell script – a file that contains commands that the shell can execute. Might run a utility, a compiled program or another shell script. Like commands on command line, they can use ambiguous file references, can have input or output redirected from or to a file or sent through a pipe. Enables you to quickly and simply initiate a complex series of tasks.

Control flow commands find most of their use in shell scripts. Enables you to alter the order or execution of commands the same way you would in a structured programming language.

Must make script executable with chmod.

Then use ./ to execute. If you have read access, can use bash (. or source) to execute as bash is executed and script is an argument.

**ARCH**

**-cfdisk**

**-mkfs.ext4 /dev/sda**

**-mkswap /dev/sda**

**-swapon /dev/sda**

**-mount /dev/sda /mnt**

**-mkdir /mnt/home**

**-mount /dev/sda /mnt/home**

**-pacstram -i /mnt base base-devel**