Linux Cheatsheet

Abyan Majid, 2023

GitHub

Basic Commands

Concept	Syntax/Example	What it does
ECHO	$ echo <text>	Print to the terminal
PWD	$ pwd	Print working directory (get path)
CD	$ cd <path>	Change directory
LS	$ ls <optional: path>	\`ls\` lists all items in current directory Flags: \`-a\`: all (including hidden files) \`-l\`: long format
TOUCH	$ touch <filename>	Create a new file
FILE	$ file <filename>	Print file type
CAT	$ cat <filename>	Print contents of a file
LESS	$ less <filename>	View text files with the ability to navigate Commands: \`q\`: quit \`up\`, \`down\`, \`left\`, \`right\`: move up, down, left, and or right \`g\`: move to the beginning of the file \`G\`: move to the end of the file \`/search\`: search for a text in the file \`h\`: help
HISTORY	$ history	Prints history of commands you’ve ran
CLEAR	$ clear	Clears the terminal
CP	$ cp <filename> <destination>	Copies file to the given destination
MV	$ mv <filename> <destination> $ mv <filename> <new filename>	Moves file to another directory or rename them. You can also move or rename directories.
MKDIR	$ mkdir <dirname> $ mkdir <dirname> <dirname>	Create a new directory. You can make multiple directories at the same time, and

	$ mkdir -p <dir>/<subdir>	you can make subdirectories at once.

RM	$ rm <filename>
	$ rm <flag> <filename>
	$ rm -r <dirname>
		Removes a file (or directory)
		
		Flags:
		`-f`: Forcefully remove write-protected files
		`-i`: Prompts a confirmation before deleting
		`-r`: Remove recursively, commonly used to delete directories

RMDIR	$ rmdir <dirname>	Removes a directory
FIND	$ find <path> -name <filename> $ find <path> -type d -name <dirname>	Finds files (or directories) given path Flags: `-name`: Name of the item being searched `-type`: Type of the item being searched, use `d` for directory
HELP	$ help <command>	Shows guidance on how to use a command, and lists all available flags
MAN	$ man <command>	Shows the manual for a given command
WHATIS	$ whatis <command>	Shows a very brief description of what a given command does.
ALIAS	$ alias <alias>=<command>	Sets an alias for a given command, such that you can run <command> by running <alias>
EXIT	$ exit	Terminates the shell
ENV	$ env Add `$` as a prefix to access environment variables, e.g: $ echo $HOME	`env` Prints all environment variables you currently have set The prefix `$` allows you to access the value of an environment variable
SUDO	$ sudo <command>	Run a command as a superuser
USERADD	$ sudo useradd <user>	Add a new superuser
USERDEL	$ sudo userdel <user>	Delete a superuser
PASSWD	$ passwd <user>	Change a superuser’s password

Text Manipulation

Concept	Syntax/Example	What it does

STDOUT Redirection	$ echo Hello World > file.txt
	$ echo Hello World >> file.txt
	
	With file descriptor: `1` (OPTIONAL):
	$ echo Hello World 1> file.txt
	$ echo Hello World 1>> file.txt
		“>” and “>>” are stdout redirections.
		
		The “>” operator performs a write to a file.
		The “>>” operator performs an append.
		
		You can do this with any other command that prints something, not just `echo`.

STDIN Redirection	$ cat < file1.txt > file2.txt	“<” is a stdin redirection. It redirects the output of the latter to the former

	With file descriptor: `0` (OPTIONAL): $ cat 0< file1.txt > file2.txt	command. This particular example copies the contents of file1 to file2.
STDERR Redirection	$ ls /nonexistent/directory 2> file.txt With file descriptor: `2` (OPTIONAL): $ ls /nonexistent/directory 2> file.txt	This is an example of writing a stderr to a file. You are required to include the file descriptor `2` when redirecting a stderr input!
PIPE	$ <command1> | <command2> Example (edit printed text in vim): $ echo Hello World | vim	Uses the `stdout` of a command as a `stdin` to another command
TEE	$ <command1> | tee <command2> Example (prints and also uses printed text in vim): $ echo Hello World | tee vim	Write the output of a command to two different streams (1) its own output stream, and (2) as a `stdin` to another command

CUT	Get characters of text by index
	$ cut -c <index> <file>
	$ cut -c <index>-<another_index> <file>
	Cut text by delimiter
	$ cut -f <index> -d <delimiter> <file>
	$ cut -f <index>-<another_index> -d “<delimiter>” <file>
		Cuts text/get portions of text.
		Flags:
		`-c`: Cut by characters
		`-f`: Cut by field
		`-d`: Specify the type of delimiter (OPTIONAL). Default is TAB.

PASTE	$ paste <file1> <file2> $ paste -s <filename> -d “<delimiter>”	Merges lines from multiple files side-by-side by a delimiter (default: TAB) Flags: `-s`: Merges lines in a single line. `-d`: Specify the type of delimiter (OPTIONAL). Default is TAB.
HEAD	$ head <file> $ head -n <num of lines> <file>	Print the first 10 lines in a file Flags: `-n`: Sets number of lines to display (DEFAULT: 10)
TAIL	$ tail <file> $ tail -n <num of lines> <file>	Prints the last 10 lines in a file Flags: `-n`: Sets number of lines to display (DEFAULT: 10)
JOIN	$ join <file1> <file2> $ join -1 <field> -2 <field> <file1> <file2>	Joins multiple files by a common field. Files must be sorted by having a number prefix for each line, e.g. file1.txt: 1 The 2 quick 3 brown 4 fox
SPLIT	$ split <file>	Split a file into different files
SORT	$ sort <file> $ sort -r <file>	Sorts a file containing numerical or alphabetical data. Flags: `-r`: Reverse sort
TR	$ tr <characters> <translation> $ tr -d <chars_to_delete>	Translates a set of characters into another set of characters

	EXAMPLE (uppercase all letters): $ tr a-z A-Z	Flags: `-d`: Delete a set of characters from a set of characters

UNIQ	$ uniq <file>
	
	RECOMMENDED SYNTAX:
	$ sort <file> | uniq
		Removes duplicates only if they are adjacent. To overcome this limitation, use sort first: $ sort <file> | uniq

WC	$ wc <file>	Displays (1) number of lines, (2) number of words, and (3) number of bytes respectively. Flags: `-l`: Display number of lines only. `-w`: Display word count only. `-c`: Display number of bytes only.
NL	$ nl <file>	Print file with number prefixing each line (can be used to count number of lines/find a particular line number)

GREP	$ grep <pattern> <file>
	
	CASE INSENSITIVE:
	$ grep -i <pattern> <file>
	
	Useful example (get all “.txt” files):
	$ ls | grep “.txt$”
	
	Useful example 2 (search in all files):
	$ grep <pattern> *
		Finds all parts of a file that includes the given pattern
		
		Flags:
		`-i`: Make <pattern> case-insensitive

Regex and Wildcards		
Concept	**Examples**	**What it does**
*** (ALL)**	Search in all files in directory: $ grep <pattern> /path/to/dir/*	A wildcard for getting all elements in a collection (such as a directory)
^ (BEGINNING OF LINE)	Given file.txt: *sally sells seashells* *by the seashore* `^by` would match: `by the seashore`	Get lines beginning with the given string prefixed by `^`
$ (END OF LINE)	Given file.txt: *sally sells seashells* *by the seashore* `ore$` would match: `by the seashore`	Get lines ending with the given string postfixed by `$`
. (CONTAINING CHARACTER)	Given file.txt: *sally sells seashells* *by the seashore* `b.` would match: `by the seashore`	Get lines containing the given character postfixed by `.`
[] (CONTAINING MULTIPLE CHARACTERS)	`d[iou]g` would match: dig, dog, dug `d[^i]g` would match: dog, dug but not dig `d[a-c]g` will match patterns like dag, dbg, and dcg	Get lines containing any of the given characters within the brackets `[]`. It is CASE-SENSITIVE.

	`d[A-C]g` will match dAg, dBg and dCg but not dag, dbg and dcg	

Vim

Concept	What to do	What it does
OPEN VIM	$ vim $ vim <file>	Opens vim
EXIT VIM	:w (writes and save file) :q (quits file) :wq (write then quit) :q! (quit without warning of unsaved changes)	`w` writes to a file and saves. `q` quits file. `!` does something forcefully without showing any warnings
VIM NAVIGATION	h, j, k, l	h: go left j: go up k: go down l: go right
INSERT MODE	i	Enter insert mode
CUT, DELETE	x (cut whatever is highlighted) dd (delete line)	`x` cuts text, `dd` deletes line
COPY/YANK	y (copy whatever is highlighted) yy (copy line)	Copy text
PASTE	p	Paste text

User Management Files

File	How to access	What it contains
/etc/passwd	$ cat /etc/passwd	A list of users and detailed information about them. Each entry display information about a user in the following order (separated by a colon): 1. Username 2. User's password (stored in /etc/shadow) 3. User ID 4. Group ID 5. GECOS field (comma delimited, used for storing info like phone number, etc) 6. User's home directory 7. User's shell
/etc/shadow	$ sudo cat /etc/shadow	A list of information about users' authentication Each entry display information in the following order (separated by a colon): 1. Username 2. Encrypted password 3. Date of last password change 4. Minimum password age 5. Maximum password age

		6. Password warning period 7. Password inactivity period 8. Account expiration date 9. Reserved field for future use
/etc/group	$ cat /etc/group	A list of information about groups Each entry display information about a group in the following order (separated by a colon): 1. Group name 2. Group password 3. Group ID List of users

---

Permissions

Concept	**Prompt**	**Explanation**
FILE PERMISSION	$ ls -l <path/to/dir>	File permissions should look something like: **drwxr-xr-x** The 1st letter represents the filetype. `d`: directory `-`: file The 2nd section represents user’s permissions, the 3rd section represents group permissions, the 4th section represents others’ permissions `r`: read permission `w`: write permission `x`: execute permission `-`: no permission

CHMOD	$ chmod <user>+<permission> <file>
	$ chmod <user>-<permission> <file>
	
	Example 1:
	$ chmod u+x file.txt
	
	Example 2 (multiple user sets):
	$ chmod ugo-w file.txt
	
	Example 3 (numerical permission set):
	$ chmod 755 file.txt
	
	// 7 = 4+2+1, so `user` can read, write, execute
	// 5 = 4+1 so `group` can read, execute
	// 5 = 4+1 so `others` can read, execute
		`chmod` modifies permission.
		
		Users: `u` (user), `g` (group), `o`(other)
		Permissions: `r` (read), `w` (write), `x` (execute)
		Operators: `+` (add permission), `-` (remove permission)
		
		Numerical representations:
		`4`: read
		`2`: write
		`1`: execute

MODIFYING FILE OWNERSHIP	Change user owner to <user>:
	$ sudo chown <user> file.txt
	
	Change group owner to <group>:
	$ sudo chgrp <group> file.txt
	
	Change user and group owner simultaneously:
	$ sudo chown <user>:<group> file.txt
		`chown`: change user owner
		`chgrp`: change group owner

UMASK	$ umask <u_perm><g_perm><o_perm>
	
	Example:
	$ umask 021
	// Users have all permissions
	// Groups cannot do a write
	// Others cannot do an execution
		Changes the default state of file permissions by removing instead of adding.
		
		Numerical representations:
		`4`: remove read
		`2`: remove write
		`1`: remove execute
		`0`: remove none

SUID	Adding/removing SUID permission for user:
	$ chmod u+s file.txt
	$ chmod u-s file.txt
	
	Numerically, prepend `4` to the permission set:
	$ chmod 4<permission set> file.txt
	
	Example:
	$ chmod 4755 file.txt
		The SUID (Set User ID) permission bit `s` lets you execute a file as the `root` user. It is represented as a prefix `4` to the permission set

SGID	Adding/removing SGID permission:
	$ chmod g+s file.txt
	$ chmod g-s file.txt
	
	Numerically, prepend `2` to the permission set:
	$ chmod 2<permission set> file.txt
	
	Example:
	$ chmod 2755 file.txt
		The SGID (Set Group ID) permission bit `s` lets the program execute as if it was a member of the group. It is represented as a prefix `2` to the permission set

STICKY BIT (t)	Adding/removing sticky bit (t):
	$ chmod +t file.txt
	$ chmod -t file.txt
	
	Numerically, prepend `1` to the permission set:
	$ chmod 1<permission set> file.txt
	
	Example:
	$ chmod 1755 file.txt
		The sticky bit `t` makes it so that only the owner of the file or the root user can delete/modify the file.

Processes

Concept	Prompt	Explanation
PS	$ ps	Shows a quick snapshot of active processes `PID`: Process ID `TTY`: Controlling terminal associated with the process (we'll go in detail about this later) `STAT`: Process status code `TIME`: Total CPU usage time `CMD`: Name of executable/command
PS AUX	$ ps aux	`a`: Display all active processes including the ones being ran by other users. `u`: Display more details about the processes. `x`: Display all processes that don't have a TTY associated with it, these programs will show a ? in the TTY field, they are most common in daemon processes that launch as part of the system startup.

		`USER`: The effective user (the one whose access we are using) `PID`: Process ID `%CPU`: CPU time used divided by the time the process has been running `%MEM`: Ratio of the process's resident set size to the physical memory on the machine `VSZ`: Virtual memory usage of the entire process `RSS`: Resident set size, the non-swapped physical memory that a task has used `TTY`: Controlling terminal associated with the process `STAT`: Process status code `START`: Start time of the process `TIME`: Total CPU usage time `COMMAND`: Name of executable/command
TOP	$ top	Display real-time information about active processes (refreshes every 10 seconds by default)
SIGHUP (1)	None	SIGHUP or 1 - Hangup, sent to a process when the controlling terminal is closed. For example, if you closed a terminal window that had a process running in it, you would get a SIGHUP signal. So basically you've been hung up on (Linux Journey)
SIGINT (2)	None	SIGINT or 2 - Is an interrupt signal, so you can use Ctrl-C and the system will try to gracefully kill the process (Linux Journey)
SIGKILL (9)	$ kill -9 <PID>	SIGKILL or 9 - Kill the process, kill it with fire, doesn't do any cleanup `kill` by default sends a SIGTERM. To send a SIGKILL instead, you need to specify a `-9` flag.
SIGSEGV (11)	None	SIGSEGV or SEGV or 11 is a common signal for process segmentation fault.
SIGTERM (15)	$ kill <PID>	SIGTERM or 15 - Kill the process, but allow it to do some cleanup first (Linux Journey) You can send a SIGTERM to terminate a process by passing the process id (PID) to a `kill` command
SIGSTOP	None	SIGSTOP - Stop/suspend a process
NICE	$ nice -n <priority> <command>	Runs a command with a level or priority the user can set. The higher the priority (nicer), the less it will be prioritised for CPU consumption. The lower the priority (less nice), the more it will be prioritised for CPU consumption.
RENICE	$ renice <priority> -p <PID> Use `$ top` to see niceness of existing processes under the `NI` column	Changes the niceness of an existing process The higher the priority (nicer), the less it will be prioritised for CPU consumption. The lower the priority (less nice), the more it will be prioritised for CPU consumption.

PROCESS STATES	$ ps aux	You can see the status of processes under the STAT column when running `ps aux`: `R`: running or runnable, it is just waiting for the CPU to process it `S`: Interruptible sleep, waiting for an event to complete, such as input from the terminal `D`: Uninterruptible sleep, processes that cannot be killed or interrupted with a signal, usually to make them go away you have to reboot or fix the issue `Z`: Zombie, we discussed in a previous lesson that zombies are terminated processes that are waiting to have their statuses collected `T`: Stopped, a process that has been suspended/stopped Source: Linux Journey
/proc FILESYSTEM	$ ls /proc $ cat /proc/<PID>/status	All processes in linux is a file, and information about these processes are store in a special file system called `/proc` To print detailed information about a process you run `cat /proc/<PID>/status`
JOBS	$ jobs	Display all jobs/commands running the background

SENDING JOB TO BACKGROUND	**Running a new command in the background (prepend a ` &`):**
	$ <command> &
	
	Sending a command that has been ran to the background:
	$ <command>
	^Z (CTRL+Z)
	[JOB NUMBER]+ Stopped <command>
	
	$ bg
	[JOB NUMBER]+ <command> &
	
	Example 1:
	$ sleep 1000 &
	
	Example 2:
	$ sleep 1000
	^Z (CTRL+Z)
	
	$ bg
		Running commands in the background lets you use your shell without waiting for the commands to finish. This is useful for commands that take a long time to run.

SENDING BACKGROUND JOB TO FOREGROUND	fg %<JOB NUMBER>
	
	Example:
	$ sleep 1000 &
	$ jobs
	[1] Running sleep 1000 &
	
	$ fg %1
	sleep 1000
		`fg` sends a background job to the foreground based on the <JOB NUMBER> you passed.
		
		Use `$ jobs` to see list of background jobs and their job numbers.

Packages

Concept	Prompt	Explanation
GZ (FILE COMPRESSION)	$ gzip <file>	Compress a file
GUNZIP (FILE DECOMPRESSION)	$ gunzip <file>.gz	Decompress a file

TAR (ARCHIVE AND EXTRACT)	Archive files:
	$ tar cvf <archive>.tar <file1> <file2>
	
	Extract files from archive:
	$ tar xvf <archive>.tar
	
	Compressed archive:
	$ tar czf <archive>.tar <file1> <file2>
	
	Extract from compressed archive:
	$ tar xzf <archive>.tar.gz
		Archive and extract files
		
		`c`: create
		`v`: tell the program to be verbose and let us see what it's doing
		`f`: the filename of the tar file has to come after this option
		`x`: extract
		`z`: use the gzip or gunzip utility
		
		(Linux Journey)

DPKG vs. RPM	None	`dpkg` is a Debian family-based package manager - therefore used in distros like Debian, Ubuntu, and LinuxMint `RPM` is a Red Hat family-based package manager - therefore used in distros like Red Hat and CentOS.

(DPKG, RPM) INSTALL PACKAGE	Debian:
	$ dpkg -i package_name.deb
	
	RPM:
	$ rpm -i package_name.rpm
		`i` for install

(DPKG, RPM) UNINSTALL PACKAGE	Debian:
	$ dpkg -r package_name.deb
	
	RPM:
	$ rpm -e package_name.rpm
		Debian:
		`r` for remove
		
		RPM:
		`e` for erase

(DPKG, RPM) LIST INSTALLED PACKAGES	Debian:
	$ dpkg -l
	
	RPM:
	$ rpm -qa
		Debian:
		`l` for list
		
		RPM:
		`q` for query
		`a` for all

(APT, YUM) PACKAGE MANAGEMENT SYSTEMS	None	`APT` is a package management system exclusive to the Debian family of Linux distros. `YUM` is a package management system exclusive to the Debian family of Linux distros.

(APT, YUM) INSTALL PACKAGE	Debian:
	$ apt install package_name
	
	RPM:
	$ yum install package_name
		None

(APT, YUM) UNINSTALL PACKAGE	Debian:
	$ apt remove package_name
	
	RPM:
	$ yum erase package_name
		None

(APT, YUM) UPDATE APT, YUM	Debian: $ apt update or $ apt upgrade	None

	RPM: $ yum update	

(APT, YUM) Get information about an installed package	Debian:
	$ apt show package_name
	RPM:
	$ yum info package_name
		None