Linux Cheatsheet

Abyan Majid, 2023

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> $ mkdir -p <dir>/<subdir>	Create a new directory. You can make multiple directories at the same time, and 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 With file descriptor: `0` (OPTIONAL): $ cat 0< file1.txt > file2.txt	"<" is a stdin redirection. It redirects the output of the latter to the former command. This particular example copies the contents of file1 to file2.
STDERR Redirection	$ ls /nonexistent/directory 2> file.txt With file descriptor: `2` (OPTIONAL):	This is an example of writing a stderr to a file. You are required to include the file descriptor `2` when redirecting a

	`$ ls /nonexistent/directory 2> file.txt`	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>` EXAMPLE (uppercase all letters): `$ tr a-z A-Z`	Translates a set of characters into another set of characters 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 `d[A-C]g` will match dAg, dBg and dCg but not dag, dbg and dcg	Get lines containing any of the given characters within the brackets `[]`. It is CASE-SENSITIVE.

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