# 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<br><br>Flags:<br>\`-a\`: all (including hidden files)<br>\`-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<br><br>Commands:<br>\`q\`: quit<br>\`up\`, \`down\`, \`left\`, \`right\`: move up, down, left, and or right<br>\`g\`: move to the beginning of the file<br>\`G\`: move to the end of the file<br>\`/search\`: search for a text in the file<br>\`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>`<br>`$ mv <filename> <new filename>` | Moves file to another directory or rename them. You can also move or rename directories. |
| **MKDIR** | `$ mkdir <dirname>`<br>`$ mkdir <dirname> <dirname>`<br>`$ 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>` | Removes a file (or directory) |

| | | |
|---|---|---|
| | $ rm <flag> <filename><br>$ rm -r <dirname> | Flags:<br>`-f`: Forcefully remove write-protected files<br>`-i`: Prompts a confirmation before deleting<br>`-r`: Remove recursively, commonly used to delete directories |
| **RMDIR** | $ rmdir <dirname> | Removes a directory |
| **FIND** | $ find <path> -name <filename><br>$ find <path> -type d -name <dirname> | Finds files (or directories) given path<br><br>Flags:<br>`-name`: Name of the item being searched<br>`-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<br><br>Add `$` as a prefix to access environment variables, e.g:<br>$ echo $HOME | `env` Prints all environment variables you currently have set<br><br>The prefix `$` 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<br>$ echo Hello World >> file.txt<br><br>With file descriptor: `1` (OPTIONAL):<br>$ echo Hello World 1> file.txt<br>$ echo Hello World 1>> file.txt | “>” and “>>” are stdout redirections.<br><br>The “>” operator performs a write to a file.<br>The “>>” operator performs an append.<br><br>You can do this with any other command that prints something, not just `echo`. |
| **STDIN Redirection** | $ cat < file1.txt > file2.txt<br><br>With file descriptor: `0` (OPTIONAL):<br>$ cat 0< file1.txt > file2.txt | “<” is a stdin redirection. It redirects the output of the latter to the former command.<br><br>This particular example copies the |

| | | |
|---|---|---|
| | | contents of file1 to file2. |
| **STDERR Redirection** | $ ls /nonexistent/directory 2> file.txt<br><br>With file descriptor: `2` (OPTIONAL):<br>$ 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><br><br>Example (edit printed text in vim):<br>$ echo Hello World \| vim | Uses the `stdout` of a command as a `stdin` to another command |
| **TEE** | $ <command1> \| tee <command2><br><br>Example (prints and also uses printed text in vim):<br>$ 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<br>$ cut -c <index> <file><br>$ cut -c <index>-<another_index> <file><br><br>Cut text by delimiter<br>$ cut -f <index> -d <delimiter> <file><br>$ cut -f <index>-<another_index> -d "<delimiter>" <file> | Cuts text/get portions of text.<br><br>Flags:<br>`-c`: Cut by characters<br>`-f`: Cut by field<br>`-d`: Specify the type of delimiter (OPTIONAL). Default is TAB. |
| **PASTE** | $ paste <file1> <file2><br>$ paste -s <filename> -d "<delimiter>" | Merges lines from multiple files side-by-side by a delimiter (default: TAB)<br><br>Flags:<br>`-s`: Merges lines in a single line.<br>`-d`: Specify the type of delimiter (OPTIONAL). Default is TAB. |
| **HEAD** | $ head <file><br>$ head -n <num of lines> <file> | Print the first 10 lines in a file<br><br>Flags:<br>`-n`: Sets number of lines to display (DEFAULT: 10) |
| **TAIL** | $ tail <file><br>$ tail -n <num of lines> <file> | Prints the last 10 lines in a file<br><br>Flags:<br>`-n`: Sets number of lines to display (DEFAULT: 10) |
| **JOIN** | $ join <file1> <file2><br>$ 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.<br><br><u>file1.txt:</u><br>1 The<br>2 quick<br>3 brown<br>4 fox |
| **SPLIT** | $ split <file> | Split a file into different files |
| **SORT** | $ sort <file><br>$ sort -r <file> | Sorts a file containing numerical or alphabetical data.<br><br>Flags:<br>`-r`: Reverse sort |
| **TR** | $ tr <characters> <translation><br>$ tr -d <chars_to_delete><br><br>EXAMPLE (uppercase all letters):<br>$ tr a-z A-Z | Translates a set of characters into another set of characters<br><br>Flags:<br>`-d`: Delete a set of characters from a set of characters |

| | | |
|---|---|---|
| **UNIQ** | $ uniq <file><br><br>RECOMMENDED SYNTAX:<br>$ 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.<br><br>Flags:<br>`-l`: Display number of lines only.<br>`-w`: Display word count only.<br>`-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><br><br>CASE INSENSITIVE:<br>$ grep -i <pattern> <file><br><br>Useful example (get all “.txt” files):<br>$ ls \| grep “.txt$”<br><br>Useful example 2 (search in all files):<br>$ grep <pattern> * | Finds all parts of a file that includes the given pattern<br><br>Flags:<br>`-i`: Make <pattern> case-insensitive |

---

## Regex and Wildcards

| **Concept** | **Examples** | **What it does** |
|---|---|---|
| *** (ALL)** | Search in all files in directory:<br>$ grep <pattern> /path/to/dir/* | A wildcard for getting all elements in a collection (such as a directory) |
| **^ (BEGINNING OF LINE)** | Given file.txt:<br>*sally sells seashells*<br>*by the seashore*<br><br>`^by` would match: `by the seashore` | Get lines beginning with the given string prefixed by `^` |
| **$ (END OF LINE)** | Given file.txt:<br>*sally sells seashells*<br>*by the seashore*<br><br>`ore$` would match: `by the seashore` | Get lines ending with the given string postfixed by `$` |
| **. (CONTAINING CHARACTER)** | Given file.txt:<br>*sally sells seashells*<br>*by the seashore*<br><br>`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<br><br>`d[^i]g` would match: dog, dug but not dig<br><br>`d[a-c]g` will match patterns like dag, dbg, and dcg<br><br>`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<br>$ vim &lt;file&gt; | Opens vim |
| **EXIT VIM** | :w (writes and save file)<br>:q (quits file)<br>:wq (write then quit)<br>:q! (quit without warning of unsaved changes) | \`w\` writes to a file and saves.<br>\`q\` quits file.<br>\`!\` does something forcefully without showing any warnings |
| **VIM NAVIGATION** | h, j, k, l | h: go left<br>j: go up<br>k: go down<br>l: go right |
| **INSERT MODE** | i | Enter insert mode |
| **CUT, DELETE** | x (cut whatever is highlighted)<br>dd (delete line) | \`x\` cuts text, \`dd\` deletes line |
| **COPY/YANK** | y (copy whatever is highlighted)<br>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.<br><br>Each entry display information about a user in the following order (separated by a colon):<br><br>1. Username<br>2. User's password (stored in /etc/shadow)<br>3. User ID<br>4. Group ID<br>5. GECOS field (comma delimited, used for storing info like phone number, etc)<br>6. User's home directory<br>7. User's shell |
| **/etc/shadow** | $ sudo cat /etc/shadow | A list of information about users' authentication<br><br>Each entry display information in the following order (separated by a colon):<br><br>1. Username<br>2. Encrypted password<br>3. Date of last password change<br>4. Minimum password age<br>5. Maximum password age<br>6. Password warning period<br>7. Password inactivity period<br>8. Account expiration date<br>9. Reserved field for future use |

| | | |
|---|---|---|
| **/etc/group** | `$ cat /etc/group` | A list of information about groups<br><br>Each entry display information about a group in the following order (separated by a colon):<br>1. Group name<br>2. Group password<br>3. Group ID<br>List of users |

---

| Permissions | | |
|---|---|---|
| **Concept** | **Prompt** | **Explanation** |
| **FILE PERMISSION** | `$ ls -l <path/to/dir>` | File permissions should look something like:<br><br>**drwxr-xr-x**<br><br>The 1st letter represents the filetype.<br>\`d\`: directory<br>\`-\`: file<br><br>The 2nd section represents user's permissions, the 3rd section represents group permissions, the 4th section represents others' permissions<br>\`r\`: read permission<br>\`w\`: write permission<br>\`x\`: execute permission<br>\`-\`: no permission |
| **CHMOD** | $ chmod <user>+<permission> <file><br>$ chmod <user>-<permission> <file><br><br>Example 1:<br>$ chmod u+x file.txt<br><br>Example 2 (multiple user sets):<br>$ chmod ugo-w file.txt<br><br>Example 3 (numerical permission set):<br>$ chmod 755 file.txt<br><br>// 7 = 4+2+1, so \`user\` can read, write, execute<br>// 5 = 4+1 so \`group\` can read, execute<br>// 5 = 4+1 so \`others\` can read, execute | \`chmod\` modifies permission.<br><br>Users: \`u\` (user), \`g\` (group), \`o\`(other)<br>Permissions: \`r\` (read), \`w\` (write), \`x\` (execute)<br>Operators: \`+\` (add permission), \`-\` (remove permission)<br><br>Numerical representations:<br>\`4\`: read<br>\`2\`: write<br>\`1\`: execute |
| **MODIFYING FILE OWNERSHIP** | Change user owner to <user>:<br>$ sudo chown <user> file.txt<br><br>Change group owner to <group>:<br>$ sudo chgrp <group> file.txt<br><br>Change user and group owner simultaneously:<br>$ sudo chown <user>:<group> file.txt | \`chown\`: change user owner<br>\`chgrp\`: change group owner |
| **UMASK** | $ umask <u_perm><g_perm><o_perm)<br><br>Example:<br>$ umask 021<br>// Users have all permissions<br>// Groups cannot do a write | Changes the default state of file permissions by removing instead of adding.<br><br>Numerical representations:<br>\`4\`: remove read<br>\`2\`: remove write |

| | | |
|---|---|---|
| | // Others cannot do an execution | `1`: remove execute<br>`0`: remove none |
| **SUID** | Adding/removing SUID permission for user:<br>$ chmod u+s file.txt<br>$ chmod u-s file.txt<br><br>Numerically, prepend `4` to the permission set:<br>$ chmod 4<permission set> file.txt<br><br>Example:<br>$ 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:<br>$ chmod g+s file.txt<br>$ chmod g-s file.txt<br><br>Numerically, prepend `2` to the permission set:<br>$ chmod 2<permission set> file.txt<br><br>Example:<br>$ 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):<br>$ chmod +t file.txt<br>$ chmod -t file.txt<br><br>Numerically, prepend `1` to the permission set:<br>$ chmod 1<permission set> file.txt<br><br>Example:<br>$ 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<br><br>`PID`: Process ID<br>`TTY`: Controlling terminal associated with the process (we'll go in detail about this later)<br>`STAT`: Process status code<br>`TIME`: Total CPU usage time<br>`CMD`: Name of executable/command |
| **PS AUX** | $ ps aux | `a`: Display all active processes including the ones being ran by other users.<br>`u`: Display more details about the processes.<br>`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.<br><br>`USER`: The effective user (the one whose access we are using)<br>`PID`: Process ID<br>`%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<br>`VSZ`: Virtual memory usage of the entire process<br>`RSS`: Resident set size, the non-swapped physical memory that a task has used<br>`TTY`: Controlling terminal associated with the process<br>`STAT`: Process status code<br>`START`: Start time of the process<br>`TIME`: Total CPU usage time<br>`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 &lt;PID&gt; | SIGKILL or 9 - Kill the process, kill it with fire, doesn't do any cleanup<br><br>`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 &lt;PID&gt; | SIGTERM or 15 - Kill the process, but allow it to do some cleanup first (Linux Journey)<br><br>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 &lt;priority&gt; &lt;command&gt; | Runs a command with a level or priority the user can set.<br><br>The higher the priority (nicer), the less it will be prioritised for CPU consumption.<br><br>The lower the priority (less nice), the more it will be prioritised for CPU consumption. |
| **RENICE** | $ renice &lt;priority&gt; -p &lt;PID&gt;<br><br>Use `$ top` to see niceness of existing processes under the `NI` column | Changes the niceness of an existing process<br><br>The higher the priority (nicer), the less it will be prioritised for CPU consumption.<br><br>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`:<br><br>`R`: running or runnable, it is just waiting for the CPU to process it<br>`S`: Interruptible sleep, waiting for an event to complete, such as input from the |

| | | |
|---|---|---|
| | | terminal<br>`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<br>`Z`: Zombie, we discussed in a previous lesson that zombies are terminated processes that are waiting to have their statuses collected<br>`T`: Stopped, a process that has been suspended/stopped<br><br>Source: Linux Journey |
| **/proc FILESYSTEM** | $ ls /proc<br>$ cat /proc/&lt;PID&gt;/status | All processes in linux is a file, and information about these processes are store in a special file system called `/proc`<br><br>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 ` &`):**<br>$ &lt;command&gt; &<br><br>**Sending a command that has been ran to the background:**<br>$ &lt;command&gt;<br>^Z (CTRL+Z)<br>[JOB NUMBER]+ Stopped &lt;command&gt;<br><br>$ bg<br>[JOB NUMBER]+ &lt;command&gt; &<br><br><br>Example 1:<br>$ sleep 1000 &<br><br>Example 2:<br>$ sleep 1000<br>^Z (CTRL+Z)<br><br>$ 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 %&lt;JOB NUMBER&gt;<br><br>Example:<br>$ sleep 1000 &<br>$ jobs<br>[1] Running sleep 1000 &<br><br>$ fg %1<br>sleep 1000 | `fg` sends a background job to the foreground based on the &lt;JOB NUMBER&gt; you passed.<br><br>Use `$ jobs` to see list of background jobs and their job numbers. |