Unix Commands Cheat Sheet: All the Commands You Need

February 14, 2023 / By Cassandra Lee

Listen to the article

To make full use of Unix operating systems such as macOS's Darwin and Linux's GNU, you need to learn how to operate Unix from the command line. Committing Unix commands and their usage to memory can be a burden. It's also hard to tell from the official documentation which commands are important and which less so.

This Unix commands cheat sheet aims to help you pick up and brush up high-priority Unix command-line operations easily. It covers essential commands, the in-built **text editor vi**, and **basic shell scripting**. A **shell script** is a computer program designed to run in Unix command-line terminals, and it's a key building block of programming in Unix.

Download this Unix command cheat sheet **here**. If you're ready, let's dive in below.

Search cheats here

Essential Commands

With these commands, you can obtain critical information about your Unix machine and perform key operations.

System Information

These provide information about your Unix machine.

COMMAND	DESCRIPTION
`uname`	Show the Unix system information.
`uname -a`	Detailed Unix system information
`uname -r`	Kernel release information, such as kernel version
`uptime`	Show how long the system is running and load information.
`who`	Display who is logged in.
`w`	Display what users are online and what they are doing.
`users`	List current users.
`whoami`	Display what user you are logged in as.
`su`	Superuser; use this before a command that requires root access e.g. `su shutdown`
`cal`	Show calendar where the current date is highlighted.
`date`	Show the current date and time of the machine.
`halt`	Stop the system immediately.
`shutdown`	Shut down the system.
`reboot`	Restart the system.
`last reboot`	Show reboot history.
`man COMMAND`	Shows the manual for a given `COMMAND`. To exit the manual, press “q”.

Input/Output Redirection

These are helpful for logging program output and error messages.

COMMAND	DESCRIPTION
`echo TEXT`	Display a line of `TEXT` or the contents of a variable.
`echo -e TEXT`	Also interprets escape characters in `TEXT`, e.g. `\n` → new line, `\b` → backslash, `\t` → tab.
`echo -n TEXT`	Omits trailing newline of `TEXT`.
`cmd1 | cmd2`	`|` is the pipe character; feeds the output of the command `cmd1` and sends it to the command `cmd2`, e.g. `ps aux | grep python3`.
`cmd > file`	Output of `cmd` is redirected to `file`. Overwrites pre-existing content of `file`.
`cmd > /dev/null`	Suppress the output of `cmd`.
`cmd >> file`	Output of `cmd` is appended to `file`.
`cmd < file`	Input of `cmd` is read from `file`.
`cmd << delim`	Input of `cmd` is read from the standard input with the delimiter character `delim` to tell the system where to terminate the input. Example for counting the number of lines of ad-hoc input: `wc -l << EOF` `> I like` `> apples` `> and` `> oranges.` `> EOF` `       4` Hence there are only 4 lines in the standard input delimited by `EOF`.

File Management

In the following commands: `X` may refer to a single file, a string containing a wildcard symbol referring to a set of multiple files e.g. `file*.txt`, or the stream output of a piped command (in which case the syntax would be `X | command` instead of `command X`); `Y` is a single directory; `A` and `B` are path strings of files/directories.

COMMAND	DESCRIPTION
`*`	Wildcard symbol for variable length, e.g. `*.txt` refers to all files with the TXT extension.
`?`	Wildcard symbol referring to a single character, e.g. `Doc?.docx` can refer to `Doc1.docx`, `DocA.docx`, etc.
`ls`	List the names of files and subfolders in the current directory. Options include `-l`, `-a`, `-t` which may be combined e.g. `-alt`.
`ls -l`	Also show details of each item displayed, such as user permissions and the time/date when the item was last modified.
`ls -a`	Also display hidden files/folders. May be combined with `ls -l` to form `ls -al`.
`ls -t`	Sort the files/folders according to the last modified time/date, starting with the most recently modified item.
`ls X`	List the files
`cd Y`	Change directory to `Y`. Special instances of `Y`: `.` — current directory `..` — parent directory
`cd`	To the `$HOME` directory
`cd ..`	Up one level to enclosing folder / parent directory
`cd /etc`	To the `/etc` directory
`cmp A B`	Compare two files `A` and `B` for sameness. No output if `A` and `B` are identical, outputs character and line number otherwise.
`diff A B`	Compare two files `A` and `B` for differences. Outputs the difference.
`pwd`	Display the path of the current working directory.
`mkdir X`	Make a new directory named `X` inside the current directory.
`mv A B`	Move a file from path `A` to path `B`. Also used for renaming files. Examples: Moving between directories `folder1` and `folder2`:

COMMAND	DESCRIPTION
	`mv ./folder1/file.txt ./folder2` The file name will remain unchanged and its new path will be `./folder2/file.txt`. Renaming a file: `mv new_doc.txt expenses.txt` The new file name is `expenses.txt`.
`cp A B`	Copy a file from path `A` to path `B`. Usage similar to `mv` both in moving to a new directory and simultaneously renaming the file in its new location. Example: `cp ./f1/file.txt ./f2/expenses.txt` simultaneously copies the file `file.txt` to the new location with a new name `expenses.txt`.
`cp -r Y Z`	Recursively copy a directory `Y` and its contents to `Z`. If `Z` exists, copy source `Y` into it; otherwise, create `Z` and `Y` becomes its subdirectory with `Y`'s contents
`rm X`	Remove (delete) `X` permanently.
`rm -r Y`	Recursively delete a directory `Y` and its contents
`rm -f X`	Forcibly remove file `X` without prompts or confirmation
`rm -rf Y`	Forcibly remove directory `Y` and its contents recursively
`rmdir Y`	Remove a directory `Y` permanently, provided `Y` is empty.
`du`	Show file/folder sizes on disk.
`du -ah`	Disk usage in human readable format (KB, MB etc.)
`du -sh`	Total disk usage of the current directory
`df`	Display free disk space.
`du -h`	Free and used space on mounted filesystems
`du -i`	Free and used inodes on mounted filesystems
`open X`	Open `X` in its default program.
`open -e X`	Opens `X` in the default text editor (macOS: TextEdit)
`touch X`	Create an empty file `X` or update the access and modification times of `X`.

COMMAND	DESCRIPTION
`cat X`	View contents of `X`.
`cat -b X`	Also display line numbers as well.
`wc X`	Display word count of `X`.
`head X`	Display the first lines of `X`. If more than a single file is specified, each file is preceded by a header consisting of the string "`==> X <==`" where "`X`" is the name of the file.
`head -n 4 X`	Show the first 4 lines of `X`.
`ls *.c | head -n 5`	Display the first 5 items of a list of `*.c` files in the current directory.
`tail X`	Display the last part of `X`. If more than a single file is specified, each file is preceded by a header consisting of the string "`==> X <==`" where "`X`" is the name of the file.
`tail -n +1 X`	Display entire contents of the file(s) `X` specified, with header of respective file names
`less`	Read a file with forward and backward navigation. Often used with pipe e.g. `cat file.txt | less`
`ln -s A S`	Create symbolic link of path `A` to link name `S`.

Search and Filter

COMMAND	DESCRIPTION
`grep patt X`	Search for a text pattern `patt` in `X`. Commonly used with pipe e.g. `ps aux | grep python3` filters out the processes containing `python3` from all running processes of all users.
`grep -v patt X`	Return lines not matching the specified `patt`.
`grep -l patt X`	Only the names of files containing `patt` are written to standard output.
`grep -i patt X`	Perform case-insensitive matching. Ignore the case of `patt`.
`find`	Find files.
`find /path/to/src -name "*.sh"`	Find all files in `/path/to/src` matching the pattern “`*.sh`” in the file name.
`find .. -size +2M`	Find all files in the parent directory larger than 2MB.
`locate name`	Find files and directories by `name`.
`sort X`	Arrange lines of text in `X` alphabetically or numerically.

Archives

COMMAND	DESCRIPTION
`tar`	Manipulate archives with TAR extension.
`tar -cf archive.tar Y`	Create a TAR archive named `archive.tar` containing `Y`.
`tar -xf archive.tar`	Extract the TAR archive named `archive.tar`.
`tar -tf archive.tar`	List contents of the TAR archive named `archive.tar`.
`tar -czf archive.tar.gz Y`	Create a gzip-compressed TAR archive named `archive.tar.gz` containing `Y`.
`tar -xzf archive.tar.gz`	Extract the gzip-compressed TAR archive named `archive.tar.gz`.

COMMAND	DESCRIPTION
`tar -cjf archive.tar.bz2 Y`	Create a bzip2-compressed TAR archive named `archive.tar.bz2` containing `Y`.
`tar -xjf archive.tar.bz2`	Extract the bzip2-compressed TAR archive named `archive.tar.bz2`.
`zip -r Z.zip Y`	Zip `Y` to the ZIP archive `Z.zip`.
`unzip Z.zip`	Unzip `Z.zip` to the current directory.

File Transfer

These are for uploading and downloading files.

COMMAND	DESCRIPTION
`ssh user@access`	Connect to `access` as `user`.
`ssh access`	Connect to `access` as your local username.
`ssh -p port user@access`	Connect to `access` as `user` using `port`.
`scp [user1@]host1:[path1] [user2@]host2:[path2]`	Login to `hostN` as `userN` via secure copy protocol for `N=1,2`. `path1` and `path2` may be local or remote. If `user1` and `user2` are not specified, your local username will be used.
`scp -P port [user1@]host1:[path1] [user2@]host2:[path2]`	Connect to `hostN` as `userN` using port for `N=1,2`.
`scp -r [user1@]host1:[path1] [user2@]host2:[path2]`	Recursively copy all files and directories from `path1` to `path2`.
`sftp [user@]access`	Login to `access` as `user` via secure file transfer protocol. If `user` is not specified, your local username will be used.
`sftp access`	Connect to `access` as your local username.
`sftp -P port user@access`	Connect to `access` as `user` using `port`.

File Permissions

Not all files are equally accessible. To prevent unwanted tampering, some files on your device may be read-only. For more information about file permissions on Unix, refer to our **Linux File Permissions Cheat Sheet**, as the same content applies to Unix.

File permissions on Unix

COMMAND	DESCRIPTION
`chmod permission file`	Change permissions of a file or directory. Permissions may be of the form `[u/g/o/a][+/-/=][r/w/x]` (see examples below) or a three-digit octal number.
`chown user2 file`	Change the owner of a file to `user2`.
`chgrp group2 file`	Change the group of a file to `group2`.

Usage examples:

- `chmod +x testfile` → allow all users to execute the file
- `chmod u-w testfile` → forbid the current user from writing or changing the file
- `chmod u+wx,g-x,o=rx testfile` → simultaneously add write & execute permissions to user, remove execute permission from group, and set the permissions of other users to only read and write.

Numeric Representation

OCTAL	PERMISSION(S)	EQUIVALENT TO APPLICATION OF
0	No permissions	`-rwx`

OCTAL	PERMISSION(S)	EQUIVALENT TO APPLICATION OF
1	Execute permission only	`=x`
2	Write permission only	`=w`
3	Write and execute permissions only: 2 + 1 = 3	`=wx`
4	Read permission only	`=r`
5	Read and execute permissions only: 4 + 1 = 5	`=rx`
6	Read and write permissions only: 4 + 2 = 6	`=rw`
7	All permissions: 4 + 2 + 1 = 7	`=rwx`

Examples

- `chmod 777 testfile` → allow all users to execute the file
- `chmod 177 testfile` → restrict current user (`u`) to execute-only, while the group (`g`) and other users (`o`) have read, write and execute permissions
- `chmod 365 testfile` → user (`u`) gets to write and execute only; group (`g`), read and write only; others (`o`), read and execute only.

Process Management

The following is redolent of functions in Windows' Task Manager, but on the command line.

COMMAND	DESCRIPTION
`&`	Add this character to the end of a command/process to run it in the background.
`ps`	Show process status. Often used with grep e.g. `ps aux | grep python3` displays information on processes involving `python3`. Meaning of `aux`: `a` = show processes for all users `u` = show user or owner column in output `x` = show processes not attached to a terminal

`ps -e`
`ps -A`
	Either of these two commands prints all running processes in the system.

COMMAND	DESCRIPTION
`ps -ef`	Print detailed overview.
`ps -U root -u root`	Display all processes running under the account `root`.
`ps -eo pid,user,command`	Display only the columns PID, USER and COMMAND in `ps` output.
`top`	Display sorted information about processes.
`kill PID`	Kill a process specified by its process ID `PID`, which you may obtain using the `ps` command.
`lsof`	List all open files on the system. (This command helps you pinpoint what files and processes are preventing you from successfully ejecting an external drive.)

Networking

These commands regulate how your Unix machine communicates with other computers, such as the local area network (LAN) router or external websites.

COMMAND	DESCRIPTION
`ifconfig`	Display all network interfaces with IP addresses
`netstat`	Print open sockets of network connections, routing tables, interface statistics, masquerade connections, and multicast memberships.
`netstat -a`	Show both listening and non-listening sockets.
`netstat -l`	Show only listening sockets, which are omitted by default.
`ping host`	Send ICMP echo request to `host`, which may be a symbolic name, domain name or IP address.
`whois domain`	Display whois information for `domain`.

COMMAND	DESCRIPTION
`dig domain`	Display DNS information for `domain`.
`host domain`	Display DNS IP address for `domain`.
`wget LINK`	Download from location `LINK`.
`curl LINK`	Display the HTML source of `LINK`.

Vi Editor – Basic Commands

Built into Unix systems, vi (or vim) is a command-line visual editor. For simple text file manipulation, the following commands will suffice.

In the Unix terminal:

COMMAND	DESCRIPTION
`vi X`	Create a new file `X` in the vi editor, or open `X` if `X` already exists.
`vi -R X` `view X`	Open an existing file `X` in read-only mode.

While using vi editor (command mode):

COMMAND	DESCRIPTION
`:w`	Save changes.
`:w filename`	Save the file as filename.
`:wq`	Save changes and quit vi editor.
`i`	Enter insert mode and amend the opened file. To return to command mode and use the other commands in this table, press the ESC key.

00:00 00:00 1

COMMAND	DESCRIPTION
`o`	Enter insert mode and add a new line underneath the cursor.
`x`	Delete the character under the cursor location.
`dd`	Delete the line where the cursor is located.
`r`	Replace the character under the cursor location with the key the user presses next.
`yy`	Copy the current line.
`p`	Paste the line that was copied beneath the cursor.
`0`	Go to the beginning of the line.
`$`	Go to the end of the line.
`h,j,k,l`	Move the cursor left, down, up, right respectively.
`G`	Jump to the first character of the last line of the file.
`gg`	Jump to the first character of the first line of the file.
`/foo`	Search for instances of “foo” in the open file.
`:%s/foo/bar`	Replace every instance of “foo” with “bar” in the open file.

Shell Programming – Basic Commands

The file extension for shell scripts is `.sh`.

00:00 00:00 1

`echo $VAR`	Display the contents of a variable.
`read VAR`	Get standard input and save it to variable `VAR`.
`#`	Designates all text after `#` on the same line to be comments (not executed).
`#!/bin/sh`	Alert the system that a shell script is being executed. Used as the first line of the shell script.

Other Links You Might Like:

- Linux File Permissions Cheat Sheet
- Linux Command Line Cheat Sheet
- Bash Cheat Sheet
- Tmux Cheat Sheet
- Tcpdump Cheat Sheet
- Our Linux Courses Collection

Variables

Valid Shell variable names contain alphanumeric [A-Z, a-z, 0-9] characters and/or underscore (_). The variable must begin an alphabetical character and is usually uppercase.

COMMAND	DESCRIPTION
`VAR_NAME=VALUE`	Define a variable `VAR_NAME` and give it a `VALUE`. The value may be a number or string enclosed by double quotation marks ("). Examples: `PRICE=100` `PERSON="John Smith"`
`readonly VAR_NAME`	Make the variable `VAR_NAME` read-only.
`unset VAR_NAME`	Delete the variable `VAR_NAME`.
`$VAR1$VAR2`	Concatenate the values of the variables `$VAR1` and `$VAR2`.

Reserved Variables

00:00 00:00 1

VARIABLE	DESCRIPTION
`$0`	File name of the current shell script.
`$1, $2, $3, …, ${10}, ${11}, …`	References to the arguments supplied to the script: `$1` is the first argument, `$2` is the second argument, and so on.
`$#`	The number of arguments supplied to a script.

VARIABLE	DESCRIPTION
`$*`	Refer to arguments separated by spaces. Here, `"a b c" d e` are considered 5 separate arguments.
`"$@"`	Refer to arguments grouped by the double quotes enclosing them. Here, `"a b c" d e` are considered 3 arguments.
`$?`	The exit status of the last command executed: 0 for success and 1 or other numbers for various errors.
`$$`	Process ID of the shell script.
`$!`	Process number of the last background command.

Arrays

In ksh shell: `set -A ARRAY_NAME value1 value2 ... valueN`

In bash shell: `ARRAY_NAME=(value1 ... valueN)`

Accessing array values (zero-indexed, i.e. first element is at [0] not [1]):

ARRAY VARIABLE	DESCRIPTION
`${ARRAY_NAME[index]}`	Display the value at `[index]` of `ARRAY_NAME`.
`${ARRAY_NAME[*]}`	Display all values of the array `ARRAY_NAME`.

00:00 00:00 1

Basic Operators

These are used in the `expressions` in **decision making** and **loop control**.

For arithmetic and relational operators, the arguments are applied to both sides of each operator, separated by spaces, e.g. `2 + 2` (not `2+2`).

ARITHMETIC OPERATOR	DESCRIPTION
`+`	Addition
`-`	Subtraction
`/`	Division
`%`	Modulus
`=`	Assignment
`==`	Equality
`!=`	Inequality

RELATIONAL OPERATOR	DESCRIPTION
`-eq`	Equal to
`-ne`	Not equal to
`-gt`	Greater than
`-lt`	Less than
`-ge`	Greater than or equal to
`-le`	Less than or equal to

BOOLEAN OPERATOR	DESCRIPTION
`!`	Logical negation / not: inverts true/false condition
`-o`	Logical OR (inclusive): returns true if any one of the operands is true
`-a`	Logical AND: returns true if all operands are true

STRING OPERATOR	DESCRIPTION
`=`	Returns true if the two operands on both sides of = are equal.
`!=`	Returns true if the two operands on both sides of != are not equal.
`-z $STRING_VAR`	Returns true if `$STRING_VAR` is zero in length.

00:00 00:00 1

In the following, `FILE` is a variable containing a string to a file/directory path.

FILE OPERATOR	DESCRIPTION
`-d $FILE`	Returns true if `FILE` is a directory.
`-f $FILE`	Returns true if `FILE` is an ordinary file as opposed to a directory or special file.

FILE OPERATOR	DESCRIPTION
`-r $FILE`	Returns true if `FILE` is readable.
`-w $FILE`	Returns true if `FILE` is writable.
`-x $FILE`	Returns true if `FILE` is executable.
`-e $FILE`	Returns true if `FILE` exists, even if `FILE` is a directory.
`-s $FILE`	Returns true if `FILE` size is greater than zero.

Decision Making

TYPES	SYNTAX

`if…fi`	`if [ expression ]`
	`then`
	    Statement(s) to be executed if `expression` is true
	`fi`

`if…else…fi`	`if [ expression ]`
	`then`
	    Statement(s) to be executed if `expression` is true
	`else`
	    Statement(s) to be executed if `expression` is false
	`fi`

	`elif [ expression2 ]`
	`then`
	    Statement(s) to be executed if `expression2` is true
	`elif [ expression3 ]`
	`then`
	    Statement(s) to be executed if `expression3` is true
	`else`

TYPES	SYNTAX
	Statement(s) to be executed if none of the given `expressions` is true `fi`

`case…esac`	`case word in`
	`pattern1)`
	Statement(s) to be executed if `pattern1` matches `word`
	`;;`
	`pattern2)`
	Statement(s) to be executed if `pattern2` matches `word`
	`;;`
	`pattern3)`
	Statement(s) to be executed if `pattern3` matches `word`
	`;;`
	`*)`
	Default condition to be executed
	`;;`
	`esac`

Loop Control

LOOP TYPE	SYNTAX

	`for VAR in word1 word2 … wordN`
	`do`
	Note: `word1 word2 … wordN` may be a list of numbers (e.g. `1 2 3 4 5`) or a set of paths (e.g. `/home/folder*/app/`).

`while`	`while command`
	`do`
	Statement(s) to be executed if command is true
	`done`

LOOP TYPE	SYNTAX
	Infinite loop: use `:` as the command, i.e. `while :`.

`until`	`until command`
	`do`
	Statement(s) to be executed until `command` is true
	`done`

`select`	Available in `ksh` and `bash` but not `sh`. Behaves like a `for`-loop with the numbers replaced by the `words`.
	
	`select VAR in word1 word2 ... wordN`
	`do`
	Statement(s) to be executed for every `word`
	`done`

FLOW CONTROL	SYNTAX
`break`	Exit a loop.
`continue`	Exit the current iteration of the loop and proceed with the next iteration.
Ctrl+C	Key combination to abort a running process
Ctrl+L	Key combination to remove the previous command and its output (macOS: command+L)

00:00 00:00 1

This article covers all the basic commands you need to know when learning to operate Unix from the command line. We hope this Unix command cheat sheet is an excellent addition to your programming and cybersecurity toolkit. See Unix commands in action with our **Complete Cyber Security Course** available with a **StationX VIP membership**.

Frequently Asked Questions

Why use Unix commands?

To operate individual and batch processes on Unix using the command-line interface, such as administrative and troubleshooting tasks.

How many commands does Unix have?

How do you write a command in Unix?

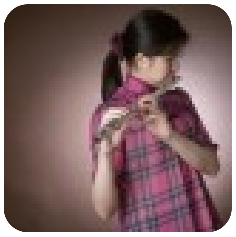
What are five Linux commands?

Grow your Cyber Security Skills

00:00 00:00 1

FIND OUT MORE

CATEGORIES CHEAT SHEETS LINUX



Cassandra Lee

I make connections across disciplines: cyber security, writing/journalism, art/design, music, mathematics, technology, education, psychology, and more. I've been advocating for girls and women in STEM since the 2010s, having written for Huffington Post, International Mathematical Olympiad 2016, and Ada Lovelace Day, and I'm honored to join StationX. You can find me on **LinkedIn** and **Linktree**.

Related Articles

Nmap Cheat Sheet 2023: All the Commands, Flags & Switches

Linux Command Line Cheat Sheet: All the Commands You Need

00:00 00:00 1

Wireshark Cheat Sheet: All the Commands, Filters & Syntax

Common Ports Cheat Sheet: The Ultimate Ports & Protocols List

Read More »

Read More »

INFO

Affiliates

Legal Notices

Privacy Policy

Site Map

SECURITY ASSESSMENT

Penetration Testing

Vulnerability Scanning

Build Reviews

Source Code Review

Social Engineering

CONSULTING

Audit & Compliance

Incident Response

Security Architecture

Risk Assessment

Security Training

00:00 00:00 1