Linux Cheatsheet

General

The basic syntax for commands is `COMMAND [OPTIONS] [INPUTS] [OUTPUT]`.

Options are generally marked with a hyphen (short form) or double hyphen (long form). Short form options can be combined, e.g. `ls -l -a` is the same as `ls -la`.

Type `man [COMMAND]` to open the manual page for the specified command. It contains a general description of the command and all available options. Press Q to quit the manual.

Most commands have a `--help` parameter which displays a summary of the manual page.

In case you accidentally typed `vi` or `vim`, you can exit the program by typing `:q!` followed by a return or by rebooting the computer.

Useful shortcuts

`Ctrl + C`

Abort running program.

`Ctrl + Shift + C` or `Ctrl + Alt + C`

Copy selected text.

`Ctrl + V` or `Ctrl + Shift + V` or `Ctrl + Alt + V`

Paste text from clipboard.

`Ctrl + Z`

Minimize running program.

`fg`

Bring minimized program back to the foreground.

`TAB`

Auto complete command or path.

`Arrow up / down`

Navigate through recently used commands.

`Crtl + R`

Search in command history. Begin typing a command to filter the history. Press `Ctrl + R` again to cycle through matching commands.

Navigation

pwd

'print working directory'

Outputs the absolute path to the directory you are currently in.

---

ls

'list directory contents'

Syntax: `ls [OPTIONS] [DIRECTORY]`

Outputs a list of files in the specified directory. If no directory is specified, the current working directory is used.

Options:

`-a / --all`

also list hidden files (starting with '.')

`-l`

vertical list format (including permissions, owner, file size and change date)

`-1`

one entry per line (only file names)

---

cd

'change directory'

Go to another directory. Absolute or relative paths can be specified.

Syntax: `cd [DIRECTORY]`

Usage:

`cd ..`

go back to the parent directory

`cd ~`

Go to the home directory. All paths starting with ~ are relative to the current user's home directory.

Example: `cd ~/Pictures` is the same as `cd /home/username/Pictures`.

`cd /`

Go to the root directory. All paths starting with / are absolute.

File Management

mkdir

'make directories'

Syntax: `mkdir [OPTIONS] DIRECTORIES`

Examples:

`mkdir test`

Creates a directory called 'test' in the current working directory.

`mkdir test1 test2`

Creates two directories called 'test1' and 'test2' in the current working directory.

`mkdir ~/Pictures/photos`

Creates a directory called 'photos' in */home/username/Pictures*.

---

rmdir

'remove directories'

Removes specified directories if they are empty.

Syntax: `rmdir [OPTIONS] DIRECTORIES`

Rarely used command, most of the time `rm -r` is preferred to `rmdir`.

---

touch

Update access and modification date of a file. If the file does not exist, it is created.

Syntax: `touch [OPTIONS] FILES`

---

mv

'move or rename files'

Syntax: `mv [OPTIONS] SOURCE DESTINATION`

The source file is moved to the specified destination. If the destination is in the same directory as the source, the file is renamed in place. There can be an arbitrary amout of source files - the last parameter ist always the destination.

Usage:

`mv myfile.txt bettername.txt`

renames 'myfile.txt' to 'bettername.txt'

`mv myfile.txt somewhere/bettername.txt`

moves 'myfile.txt' to a directory called 'somewhere' and renames it to 'bettername.txt'

`mv myfile.txt somewhere/`

moves 'myfile.txt' to a directory called 'somewhere'

`mv myfile.txt otherfile.txt thirdfile.txt somewhere/`

moves three files to a directory

`mv *.txt somewhere/`

moves all files ending with '.txt' to a directory

Options:

`-b / --backup`

make a backup of all specified files

`-f / --force`

overwrite existing files without asking

`-i / --interactive`

interactive mode, prompts before overwriting

`-n / --no-clobber`

do not overwrite existing files

`-S / --suffix=SUFFIX`

specify backup suffix

`-u / --update`

move only if source is newer than the destination or destination is missing

`-v / --verbose`

print progress to stdout, useful when moving multiple large files

---

cp

'copy files'

Syntax: `cp [OPTIONS] SOURCE DESTINATION`

The source files are copied to the specified destination. Like with `mv`, there can be an arbitrary amout of source files - the last parameter ist always the destination.

Usage:

Basically behaves just like `mv`.

Options:

All options described in the section for `mv` can be used in the same way.

`-R / -r / --recursive`

has to be used if a copied directory contains subdirectories

---

rm

'remove files'

Syntax: `rm [OPTIONS] FILES`

Deletes all specified files. Does not remove directories unless the `-r` option is used. If a file can't be removed (e.g. because of lacking permissions), the program will prompt the user.

Usage:

`rm file.txt`

deletes the file 'file.txt'

`rm *.txt`

deletes all files with the suffix '.txt'

`rm -r my_directory`

deletes the directory 'my_directory' including its content

`rm -rf /`

Deletes all files on the system, including the operating system (if permissions are sufficient). Don't do this!

Options:

`-R / -r / --recursive`

required to remove directories

Reading Files

cat

'concatenate files'

Syntax: `cat [OPTIONS] FILES`

Reads all specified files, concatenates them and prints their content to standard output. Is often used as a simple tool to display the content of a file in the terminal.

Options:

`-n / --number`

display line numbers

`-s / --squeeze-blank`

get rid of repeated empty lines

---

head, tail

'output the first (head) or last (tail) lines of files'

Syntax:

`head [OPTIONS] [FILES]`

`tail [OPTIONS] [FILES]`

Similar to cat, but only a part (10 lines by default) of each file is displayed. Very useful for reading log files and filtering sorted lists.

Options:

`-n / --lines=NUM`

specify the number of lines to be printed

`-c / --bytes=NUM`

limit the output to a number of bytes instead of a number of lines

---

less

'open an interactive read-only view of a file'

Syntax: `less [OPTIONS] FILE`

This is the way to go if you just want to take a quick peak into a (text) file.

Navigation:

`Arrow Up/Down or J/K`

scroll up/down a line

`Ctrl + U, Ctrl + D`

scroll up/down 10 lines

`gg / Shift + G`

go to the top (gg) or bottom (G) of the file

`q`

quit

`/`

Open search bar. Enter a search term an press return to find the first occurence. Press `n` to jump to the next occurence, press `Shift + N` to jump to the previous occurence.

---

sort

'sort lines of text files'

Syntax: `sort [OPTIONS] FILES`

Keep in mind that *lines* are sorted. If you want e.g. a sorted list of all words in a file, you have to split all words into separate lines before.

Options:

`-R / --random-sort`

shuffle the output

`-r / --reverse`

reverse the output

---

uniq

'report or omit repeated lines'

Syntax: `uniq [OPTIONS] [INPUT [OUTPUT]]`

By default, consecutive matching lines of the input are merged. Very useful in combination with `sort`.

Options:

`-c / --count`

show number of occurences for each line

`-d / --repeated`

only print lines occuring more than once

`-u / --unique`

only print unique lines

`-i / --ignore-case`

ignore differences in case

`-w / --check-chars=N`

only compare the first N characters of each line

---

wc

'word count'

Syntax: `wc [OPTIONS] [FILES]`

Prints number of lines, words, and bytes for each specified file.

Options:

`-c / --bytes`

print the byte count

`-l / --lines`

print the line count

`-w / --words`

print the word count

Redirecting Input and Output

Most commands get some type of input (e.g. standard input or a file), process it and output a result (to standard output). If nothing else is specified, standard output is printed in the terminal. But the true power of command line tools lies in their capability to be combined via pipes - this means the output of one program is used as the input of another program.

Creating a Pipeline

The `|` operator (pipe) can be used to chain together programs. The first programm passes its input to the second program and so on. The result of the final program is passed to *STDOUT* (standard output) and thus printed in the terminal.

Examples:

```
ls | wc -l
```

prints the number of files in the current directory

```
cat *.txt | sort | uniq
```

prints unique lines in all *.txt* files in the current directory

Pipes are especially useful for filtering streams of text by for example removing or replacing characters or only including lines which contain a certain string. Such commands are discussed in a later section.

Redirecting Output to a File

Oftentimes, we want to store the result of a program. This can be done by redirecting standard output to a file with the `>` operator.

Examples:

```
ls > filelist.txt
```

creates a list of files in the current directory

```
cat list.txt | sort -r > reverse.txt
```

creates a reversed copy of a list

The `>` operator will overwrite the specified file. Use the `>>` operator to append output to an existing file.

Some programs will print error messages to the virtual *STDERR* file (standard error). Those are normally also printed in the terminal. To redirect error messages to a file, the `2>` operator can be used. To redirect both *STDOUT* and *STDERR* to a file, use the `&>` operator.

To suppress output of a program, redirect all output to the special file */dev/null*: `command &> /dev/null`.

---

echo

'*output a line of text to standard output*'

Syntax: `echo [OPTIONS] [STRING]`

Similar to *print* functions of many programming languages, `echo` can be used to display a line of text. As its output is passed to *STDOUT*, `echo` can be used to pipe arbitrary input into other commands. Another important use of this command is printing text output from shell scripts (described later).

Options:

`-e`

interpret backslash-escaped sequences (for example `\n` for new line)

Searching

grep

'global-regex-print: print lines that match patterns'

Syntax:

```
grep [OPTIONS] PATTERNS [FILES]
```

```
command | grep [OPTIONS] PATTERNS
```

`grep` searches its input for strings matching a pattern and outputs lines containing such strings. It can either be used to search files by passing them as arguments, or to search within a stream piped into `grep`. Therefore, it is one of the most useful and versatile commands. Patterns can be (but don't have to be) written in quotes.

Pattern Syntax:

`grep` supports different types of regular expressions that can be used as matching patterns:

`-E / --extended-regexp`

interpret patterns as extended regular expressions

`-F / --fixed-strings`

interpret patterns as fixed strings, not regular expressions

`-G / --basic-regexp`

interpret patterns as basic regular expressions *(default)*

`-P / --extended-regexp`

interpret patterns as Perl-compatible regular expressions

Options:

`-e / --regexp=PATTERN`

Specify a pattern to be searched for. Can be used multiple times so `grep` can search for multiple patterns at once: `grep -e PATTERN1 -e PATTERN2 file.txt`. If only searching for one pattern, the `-e` option does not have to be used.

`-f / --file=FILE`

Read search patterns line by line from a file. Useful for more complex operations.

`-i / --ignore-case`

Makes `grep` case-insensitive. `grep -i 'hello'` matches 'hello', 'Hello', 'HELLO' and so on.

`-v / --invert-match`

invert the result so only non-matching lines are selected

`-x / --line-regexp`

select only those lines that entirely match the pattern

`-c / --count`

instead of printing normal output, print the number of found matches

`-l / --files-with-matches, -L / --files-without-matches`

instead of printing normal output, print the name of the first input file with a match (`-l`) or without a match (`-L`)

`-m / -max-count=NUM`

stop reading a file after the specified number of matches

`-o / --only-matching`

do not print the whole line but only the matching string

`-h / --no-filename, -H / --with-filename`

Print or suppress printing file names for each match when multiple files are searched. By default, file names are only shown, when more than one file is passed to `grep`.

`n / --line-number`

prefix each match with its line number (starting from 1) in the original file

`-r / --recursive`

also search within subdirectories
