Command-Line & Shell Cheatsheet

A comprehensive cheat sheet for navigating and utilizing command-line interfaces and shell environments effectively. This guide covers essential commands, scripting techniques, and environment configurations for improved productivity.

Basic Navigation & File Management

Navigation Commands

`pwd`	Print working directory (shows the current directory).
`cd <directory>`	Change directory to `<directory>`. Use `cd ..` to go up one level.
`ls`	List directory contents (files and subdirectories).
`ls -l`	List directory contents in long format (permissions, size, etc.).
`ls -a`	List all files, including hidden files (starting with `.`).
`ls -t`	List files sorted by modification time (newest first).

File & Directory Manipulation

`mkdir <directory>`	Create a new directory named `<directory>`.
`touch <file>`	Create an empty file named `<file>` or update the timestamp if the file exists.
`cp <source> <destination>`	Copy the file or directory `<source>` to `<destination>`.
`mv <source> <destination>`	Move or rename the file or directory `<source>` to `<destination>`.
`rm <file>`	Remove (delete) the file `<file>`. **Warning: This is permanent!**
`rm -r <directory>`	Remove the directory `<directory>` and its contents recursively. **Use with caution!**

File Viewing

`cat <file>`	Display the entire contents of `<file>` on the terminal.
`less <file>`	View the contents of `<file>` one page at a time, allowing navigation.
`head <file>`	Display the first few lines of `<file>` (default is 10 lines).
`tail <file>`	Display the last few lines of `<file>` (default is 10 lines).
`tail -f <file>`	Display the last few lines of `<file>` and continue to display new lines as they are added (follow mode).
`wc <file>`	Word count - Display number of lines, words, and bytes in file.

Piping, Redirection, and Permissions

Piping and Redirection

`|` (pipe)	Pass the output of one command as input to another command. **Example:** `ls -l | grep 'txt'` (list files and filter for those containing 'txt')
`>` (redirect output)	Redirect the output of a command to a file, overwriting the file if it exists. **Example:** `ls > files.txt` (save the list of files to files.txt)
`>>` (append output)	Append the output of a command to a file without overwriting it. **Example:** `echo 'New line' >> files.txt`
`2>` (redirect error)	Redirect standard error to a file. **Example:** `command 2> error.log`
`&>` (redirect both)	Redirect standard output and standard error to a file. **Example:** `command &> output.log`
`<` (redirect input)	Redirect input from a file to a command. **Example:** `wc < files.txt` (count words in files.txt)

File Permissions

`chmod <permissions> <file>`	Change the permissions of a file or directory. Permissions can be specified numerically (e.g., `755`) or symbolically (e.g., `u+rwx,g+rx,o+rx`).
`chown <user>:<group> <file>`	Change the owner and group of a file or directory.
`ls -l` output	The output shows permissions in the format `-rwxr-xr--`. The first character indicates the file type (e.g., `-` for regular file, `d` for directory). The next three characters are the owner's permissions, followed by the group's permissions, and then others' permissions. `r` = read, `w` = write, `x` = execute.
Numeric Permissions	`4` = read, `2` = write, `1` = execute. Add these values to set permissions. For example, `7` (4+2+1) means read, write, and execute.
Symbolic Permissions	`u` = user/owner, `g` = group, `o` = others, `a` = all. `+` adds a permission, `-` removes a permission, `=` sets a permission. **Example:** `chmod u+x <file>` (add execute permission for the owner)
`umask`	Sets default permissions for newly created files and directories. Common value is `022`.

Process Management

`ps`	Display a snapshot of the current processes.
`ps aux`	Display a comprehensive list of all processes.
`top`	Display a dynamic real-time view of running processes.
`kill <PID>`	Terminate the process with the specified process ID (PID). **Example:** `kill 1234` (kills process with PID 1234)
`kill -9 <PID>`	Forcefully terminate the process (use as a last resort). **Example:** `kill -9 1234`
`bg`	Place a stopped job in the background.
`fg`	Move a background job to the foreground.
`jobs`	List active jobs.

Shell Scripting Basics

Script Structure

A shell script is a text file containing a sequence of commands.

- The first line should specify the interpreter using a shebang (`#!`):

```
#!/bin/bash
```

- Comments start with `#`.
- Make the script executable using `chmod +x <script_name>`.

Variables

Defining a variable	`variable_name="value"` (no spaces around `=`)`
Accessing a variable	`$variable_name` or `${variable_name}`
Environment variables	Accessed like regular variables. Examples: `$HOME`, `$PATH`, `$USER`
Read-only variables	`readonly variable_name`
Unsetting a variable	`unset variable_name`

Control Structures

If statement:

```
if [ condition ]; then
  commands
elif [ condition ]; then
  commands
else
  commands
fi
```

For loop:

```
for variable in word1 word2 ... wordN; do
  commands
done
```

While loop:

```
while [ condition ]; do
  commands
done
```

Until loop:

```
until [ condition ]; do
  commands
done
```

Functions

Defining a function	`function_name() {`
	`  commands`
	`}`
	
	or
	
	`function function_name {`
	`  commands`
	`}`

Calling a function	`function_name`
Passing arguments	Inside the function, access arguments using `$1`, `$2`, etc.
Returning a value	Use `return value` (value must be an integer between 0 and 255). Use `echo` to return other types of data, but capture the output.

Command Substitution

`$(command)`	Execute `command` and substitute the output into the current command line. **Example:** `echo "Today is $(date +%Y-%m-%d)"`
`` `command` ``	(Deprecated) - An older form of command substitution (using backticks).

Advanced Shell Techniques

Regular Expressions (grep)

`grep` is a powerful tool for searching text using regular expressions.

- `grep 'pattern' <file>`: Search for lines containing `pattern` in `file`.
- `grep -i 'pattern' <file>`: Case-insensitive search.
- `grep -r 'pattern' <directory>`: Recursive search in `directory`.
- `grep -v 'pattern' <file>`: Invert the match (show lines that *do not* contain `pattern`).
- `grep -E 'pattern' <file>`: Use extended regular expressions.

sed (Stream Editor)

`sed` is a powerful stream editor for transforming text.

- `sed 's/old/new/g' <file>`: Replace all occurrences of `old` with `new` in `file`.
- `sed -i 's/old/new/g' <file>`: Replace in-place (modifies the file directly).
- `sed '/pattern/d' <file>`: Delete lines containing `pattern`.
- `sed '2d' <file>`: Delete the second line.
- `sed '$d' <file>`: Delete the last line.

awk (Pattern Scanning and Processing Language)

`awk` is a powerful programming language for text processing.

- `awk '{print $1}' <file>`: Print the first field of each line in `file` (fields are separated by spaces by default).
- `awk -F',' '{print $2}' <file>`: Print the second field of each line, using `,` as the field separator.
- `awk '/pattern/ {print $0}' <file>`: Print lines containing `pattern`.
- `awk 'BEGIN {print "Start"} {print $1} END {print "End"}' <file>`: Execute code before and after processing the file.

find

`find . -name "*.txt"`	Find all files with the `.txt` extension in the current directory and its subdirectories.
`find / -type d -name "config"`	Find all directories named `config` in the entire file system.
`find . -size +1M`	Find all files larger than 1MB in the current directory.
`find . -mtime -7`	Find files modified in the last 7 days.
`find . -user <username>`	Find all files owned by `<username>`.
`find . -exec ls -l {} \;`	Execute the `ls -l` command on each file found.