Shell Commands Cheat Sheet

This document provides a quick reference for common shell commands, focusing on file/directory management, searching, permissions, and I/O operations. Each command includes a practical example.

File and Directory Management 📂

Command	Description	Example
pwd	**P**rint **W**orking **D**irectory (shows your current location).	pwd -> /home/user/documents
ls	**Ls**ts the contents of a directory.	ls -l (shows a detailed list)
cd	**C**hange **D**irectory.	cd /home/user/pictures
mkdir	**Mk**e a new **dir**ectory.	mkdir new_project
touch	Creates a new, empty file.	touch report.txt
cp	**Cp**y a file.	cp report.txt report_backup.txt
mv	**Mv**e or rename a file.	mv report.txt final_report.txt
rm	**Rm**ove a file.	rm old_report.txt
rmdir	**Rm**ove an empty **dir**ectory.	rmdir old_project

Searching for Files and Content 🔍

find

Recursively searches for files and directories based on given criteria.

Criteria	Flag	Example
Name (case-sensitive)	-name	find . -name "*.log"
Name (case-insensitive)	-iname	find . -iname "*.LOG"
Path	-path	find ./project -path "*/test/*.js"
Type (file)	-type f	find . -type f
Type (directory)	-type d	find . -type d
Owner	-user	find /home -user jsmith
Modified Time	-mtime	find . -mtime -7 (modified in the last 7 days)
Size	-size	find . -size +1M (larger than 1 Megabyte)
Execute Command	-exec	find . -name "*.tmp" -exec rm {} \;

grep

Searches for patterns inside files.

Function	Flag	Example
Case-insensitive search	-i	grep -i "error" system.log
Show line number	-n	grep -n "main" app.c
Recursive search	-r	grep -r "API_KEY" ./config
Match whole word only	-w	grep -w "user" users.txt
Show count of matches	-c	grep -c "warning" system.log
Invert match (non-matching lines)	-v	grep -v "#" config.conf
Use Extended Regex	-E	grep -E "error|critical" system.log
Show lines **A**fter match	-A<num>	grep -A 3 "error" app.log
Show lines **B**efore match	-B<num>	grep -B 2 "error" app.log
Show lines of **C**ontext	-C<num>	grep -C 2 "error" app.log

Permissions and Execution 🔐

Command	Description	Example
sudo	**S**uper **U**ser **Do**; executes a command with root privileges.	sudo apt-get update
chmod	**Ch**ange **Mod**e; modifies file permissions.	chmod +x script.sh (makes script executable)
sh	Executes a shell script.	sh run_backup.sh

I/O, Piping, and Other Utilities ⚙️

Operator/Command	Description	Example
>	Redirects output to a file, overwriting it.	ls -l > file_list.txt
>>	Appends output to a file.	echo "New log entry" >> system.log
<	Redirects input from a file.	sort < unsorted_names.txt
`	`	**Pipe**; sends one command's output as input to another.
date	Displays the current date and time.	date -> Wed Aug 20 02:45:00 +06 2025
echo	Displays text.	echo "Hello, World!"
curl	**cURL**; transfers data from or to a server.	curl -O https://example.com/file.zip
tee	Reads input and writes to a file and standard output.	`ls -l

Getting Help 🆘

Command	Description	Example
man	Shows the full **man**ual for a command.	man ls
--help	Shows a brief help message for a command.	grep --help
tldr	Shows simplified, example-based help pages.	tldr find

Shell Commands Cheat Sheet

This cheat sheet provides a quick reference for the `find`, `grep`, and shell scripting commands, now with more practical examples.

🔎 `find` Command

The `find` command is used to search for files and directories.

Command	Description
`find . -name '*.txt'`	Find all files ending with `.txt` in the current directory.
`find . -iname '*.Txt'`	Find files ending with `.Txt` (case-insensitive).
`find . -path '*/archive/*.txt'`	Find files with the `.txt` extension inside any directory named `archive`.
`find . -type f`	Find only files.
`find . -type d`	Find only directories.
`find . -user alice`	Find files owned by the user `alice`.
`find . -group staff`	Find files belonging to the group `staff`.
`find . -maxdepth 2 -type f`	Find files only up to 2 directory levels deep.

`find . -name '*.log' -exec rm {} \;`	Find all `.log` files and execute the `rm` command on each.
`find ~ -name '*.txt' -mtime -7 -size +1k`	Find `.txt` files in the home directory modified in the last 7 days that are larger than 1KB.

Practical `find` Example: Archiving Old Logs

This command finds all `.log` files in the `/var/log` directory that are larger than 1 megabyte and haven't been accessed in over 30 days, then compresses and moves them to an `archive` directory.

```
# Create the archive directory if it doesn't exist
mkdir -p /var/log/archive

# Find and process the files
find /var/log -name "*.log" -size +1M -atime +30 -exec gzip {} \; -exec mv {}.gz /var/log/archive \;
```

📜 `grep` Command

The `grep` command searches for specific patterns within files.

Command	Description
`grep "print" cpu.c`	Search for the word "print" in the file `cpu.c`.
`grep -in linux README.md`	Search for "linux" case-insensitively (`-i`) and show line numbers (`-n`).
`grep -w "printf" cpu.c`	Match the whole word "printf".

`grep -c "print" cpu.c`	Count the number of lines containing "print".
`grep -v "include" common.h`	Invert the match; show lines that *do not* contain "include".
`grep -E 'double`	int' common.h`
`grep -rn "common.h" .`	Recursively (`-r`) search for "common.h" in the current directory and show line numbers (`-n`).
`grep -B2 -A4 "main" cpu.c`	Show 2 lines before (`-B`) and 4 lines after (`-A`) the matched line.
`grep -C3 "main" cpu.c`	Show 3 lines of context (`-C`) before and after the matched line.

Practical `grep` Example: Filtering System Messages

You can pipe the output of other commands to `grep` to filter them. This is useful for searching through logs or system information in real-time.

```
# Search for USB-related messages in the kernel's message buffer
dmesg | grep -i "usb"

# Filter the list of running processes to find the ssh service
ps aux | grep "ssh"
```

💻 Shell Scripting Basics

Variables

- **Assignment**: `my_variable="hello"` (No spaces around the `=` sign).
- **Usage**: `echo "$my_variable"`
- **Single vs. Double Quotes**:

    - `echo '$my_variable'` (Literal): Prints `$my_variable`.
    - `echo "$my_variable"` (Expands): Prints `hello`.

Functions

- Define a function and use arguments:

```
my_function() {
  echo "First argument: $1"
  echo "Second argument: $2"
}

# Call the function
my_function "hello" "world"
```

-
- **Special Variables in Functions/Scripts**:
    - `$0`: Name of the script or function.
    - `$1`, `$2`, ... `$9`: Positional arguments.
    - `$*`: All arguments as a single string.
    - `$@`: All arguments as separate words.
    - `$#`: The number of arguments.

Conditional Statements (`if`)

- **String Comparison**:

```
if [[ "$name" == "admin" ]]; then
  echo "Welcome, admin!"
fi
```

-
- **Numeric Comparison**:

```
if [[ "$age" -ge 18 ]]; then
  echo "You are an adult."
fi
```

-
- **File Checks**:
    - `[[ -f "$file" ]]`: True if `$file` exists and is a regular file.
    - `[[ -d "$dir" ]]`: True if `$dir` exists and is a directory.

- `[[ -z "$username" ]]`: True if the variable `username` is empty.

Numeric	String	Logical
`-eq` (equal)	`==` (equal)	`&&` (AND)
`-ne` (not equal)	`!=` (not equal)	`||` (OR)
`-gt` (greater than)	`-n` (not empty)	`!` (NOT)
`-lt` (less than)	`-z` (is empty)	
`-ge` (greater or equal)		
`-le` (less or equal)		

Loops (`for`)

- **Iterate over a list**:

```
for name in Alice Bob Carol; do
  echo "Hello, $name"
done
```

-
- **Iterate over files**:

```
for file in *.txt; do
  echo "Processing $file"
done
```

-

- **C-style numeric loop**:

```
for (( i=1; i<=5; i++ )); do
  echo "Count: $i"
done
```

-

Practical Script Example: Batch Rename Files

This script renames all `.jpeg` files in the current directory to `.jpg` by iterating through them with a `for` loop.

```
#!/bin/bash

# A script to rename all .jpeg files to .jpg in the current directory.

# Loop through each file with the .jpeg extension
for filename in *.jpeg; do
  # Check if the file actually exists to avoid errors
  if [[ -f "$filename" ]]; then
    # Use parameter expansion to create the new name
    # This removes the '.jpeg' part and adds '.jpg'
    new_name="${filename%.jpeg}.jpg"

    # Rename the file and print a confirmation message
    mv "$filename" "$new_name"
    echo "Renamed '$filename' to '$new_name'"
  fi
done

echo "Batch rename complete."
```

🟢 `grep` Cheat Code (searching inside files)

👉 **Syntax:**

```
grep [OPTIONS] "PATTERN" file(s)
```

🔑 Most Useful Options

- `-i` ⟶ ignore case
- `-n` ⟶ show line numbers
- `-r` ⟶ recursive search in directories
- `-w` ⟶ match whole word
- `-E` ⟶ use Extended Regex (so `+`, `{n}`, `|` work)
- `-A NUM` ⟶ show NUM lines after match
- `-B NUM` ⟶ show NUM lines before match
- `-C NUM` ⟶ show NUM lines of context
- `-c` ⟶ count matches only
- `-v` ⟶ invert (show non-matching lines)
- `--color` ⟶ highlight matches

🔑 Regex Patterns (mix + match)

- `.` ⟶ any character
- `^` ⟶ start of line

- `$` → end of line
- `\b` → word boundary
- `[abc]` → any of a, b, or c
- `[a-z0-9]` → range (lowercase + digits)
- `[^xyz]` → NOT x, y, or z
- `.*` → anything (zero or more)
- `x\{5\}` or `x{5}` (with `-E`) → repeat 5 times
- `(a|b)` → a or b (with `-E`)

🚀 Grep “Cheat Moves”

Find word (case-insensitive):

```
grep -i "word" file.txt
```

-

Find regex pattern in all `.txt`:

```
grep -nE "^[A-Z]{2}[0-9]{3}$" *.txt
```

-

Show 3 lines of context:

```
grep -C3 "error" logfile.log
```

-

Highlight word inside code:

```
grep --color=always -n "function" *.c
```

-

🔵 `find` Cheat Code (searching files/directories)

👉 **Syntax:**

```
find [WHERE] [CONDITIONS] [ACTIONS]
```

🔑 Most Useful Conditions

- `-name "*.txt"` → match name (case-sensitive)
- `-iname "*.txt"` → match name (case-insensitive)
- `-type f` → only files
- `-type d` → only directories
- `-size +1M` → larger than 1MB
- `-mtime -7` → modified in last 7 days
- `-user alice` → owned by user
- `-path "*/dir/*"` → match path pattern

🔑 Actions

- Default → print path
- `-exec COMMAND {} \;` → run a command on each result
- `-delete` → delete matching files
- `-ls` → list details

🚀 Find "Cheat Moves"

Find all `.log` files:

```
find . -name "*.log"
```

-

Find and delete all `.tmp`:

```
find . -type f -name "*.tmp" -delete
```

-

Find files > 10MB:

```
find . -type f -size +10M
```

-

Find files modified last 2 days:

```
find . -mtime -2
```

-

Find `.c` files and count lines:

```
find . -name "*.c" -exec wc -l {} \;
```

-

🔥 GREP + FIND Combo Power

Sometimes you need both: *find the file first, then search inside it*.

```
find . -name "*.txt" -exec grep -in "pattern" {} \;
```

Or faster with `xargs`:

```
find . -name "*.txt" | xargs grep -in "pattern"
```

🧠 Mental Model

Whenever you get a problem:

1. **Am I searching inside files?** → Use `grep`
2. **Am I searching for files themselves?** → Use `find`
3. **Do I need both?** → Combine `find ... -exec grep`

✅ Conditions (`if`, `elif`, `else`)

🔑 Syntax

```
if [ condition ]; then
    # code
elif [ other_condition ]; then
    # code
else
    # code
fi
```

🔑 String Conditions

```
[ "$a" = "$b" ]       # equal
[ "$a" != "$b" ]      # not equal
[ -z "$a" ]           # empty string
[ -n "$a" ]           # non-empty string
```

🔑 Numeric Conditions

```
[ $a -eq $b ]   # equal
[ $a -ne $b ]   # not equal
[ $a -lt $b ]   # less than
[ $a -le $b ]   # less or equal
[ $a -gt $b ]   # greater than
[ $a -ge $b ]   # greater or equal
```

🔑 File Conditions

```
[ -e file ]   # exists
[ -f file ]   # regular file
[ -d dir ]    # directory
[ -r file ]   # readable
[ -w file ]   # writable
[ -x file ]   # executable
```

🔄 Loops

🔑 For Loop

```
for i in 1 2 3 4 5; do
    echo $i
done
```

Range:

```
for i in {1..10}; do
    echo $i
done
```

C-style:

```
for ((i=1; i<=5; i++)); do
    echo $i
done
```

🔑 While Loop

```
count=1
while [ $count -le 5 ]; do
    echo $count
    count=$((count+1))
done
```

🔑 Until Loop (opposite of while)

```
count=1
until [ $count -gt 5 ]; do
    echo $count
    count=$((count+1))
done
```

🔀 Case Statement

```
read -p "Enter a choice: " choice
case $choice in
    1) echo "Option 1";;
    2) echo "Option 2";;
    *) echo "Invalid";;
esac
```

⚡ Break & Continue

```
for i in {1..10}; do
    if [ $i -eq 5 ]; then
        break    # stop loop
    fi
    echo $i
done

for i in {1..10}; do
    if [ $i -eq 5 ]; then
        continue  # skip iteration
    fi
    echo $i
done
```

🧠 “Problem Solving Template”

Whenever you see a shell scripting problem:

1. **Is it decision-based?** → Use `if [ condition ] ... fi`
2. **Do I need repetition?** → Use `for`, `while`, or `until`
3. **Do I need multiple choices?** → Use `case ... esac`
4. **Do I need to stop/skip?** → Use `break` / `continue`

🚀 Example Mini-Problems

Check even/odd:

```
read -p "Enter a number: " n
if [ $((n%2)) -eq 0 ]; then
    echo "Even"
else
    echo "Odd"
fi
```

Sum numbers 1–10:

```
sum=0
for i in {1..10}; do
    sum=$((sum+i))
done
echo "Sum = $sum"
```

Print lines of a file:

```
while read line; do
    echo $line
done < file.txt
```