

# Bash Cheat Sheet



## Command History

```
!!          # Run the last command
touch foo.sh
chmod +x !$  # !$ is the last argument of the last command i.e. foo.sh
```

## Navigating Directories

```
pwd          # Print current directory path
ls           # List directories
ls -a|--all  # List directories including hidden
ls -l        # List directories in long form
ls -l -h|--human-readable # List directories in long form with human readable sizes
ls -t        # List directories by modification time, newest first
stat foo.txt # List size, created and modified timestamps for a file
stat foo     # List size, created and modified timestamps for a directory
tree         # List directory and file tree
tree -a      # List directory and file tree including hidden
tree -d      # List directory tree
cd foo       # Go to foo sub-directory
cd           # Go to home directory
cd ~         # Go to home directory
cd -         # Go to last directory
pushd foo    # Go to foo sub-directory and add previous directory to stack
popd         # Go back to directory in stack saved by `pushd`
```

## Creating Directories

```
cp -R|--recursive foo bar # Copy directory
mv foo bar # Move directory
rsync -z|--compress -v|--verbose /foo /bar # Copy directory,
overwrites destination
rsync -a|--archive -z|--compress -v|--verbose /foo /bar # Copy directory,
without overwriting destination
rsync -avz /foo username@hostname:/bar # Copy local direc-
tory to remote directory
rsync -avz username@hostname:/foo /bar # Copy remote direc-
tory to local directory
```

## Moving Directories

```
cp -R|--recursive foo bar # Copy directory
mv foo bar # Move directory
rsync -z|--compress -v|--verbose /foo /bar # Copy directory,
overwrites destination
rsync -a|--archive -z|--compress -v|--verbose /foo /bar # Copy directory,
without overwriting destination
rsync -avz /foo username@hostname:/bar # Copy local direc-
tory to remote directory
rsync -avz username@hostname:/foo /bar # Copy remote direc-
tory to local directory
```

## Deleting Directories

```
rmdir foo # Delete non-empty directory
rm -r|--recursive foo # Delete directory including contents
rm -r|--recursive -f|--force foo # Delete directory including contents, ig-
nore nonexistent files and never prompt
```

## Creating Files

```
touch foo.txt | vim foo.txt | nano foo.txt          # Create file or update
existing files modified timestamp
touch foo.txt bar.txt    # Create multiple files
touch {foo,bar}.txt      # Create multiple files
touch test{1..3}         # Create test1, test2 and test3 files
touch test{a..c}         # Create testa, testb and testc files
mktemp                   # Create a temporary file
```

## Standard Output, Standard Error and Standard Input

```
echo "foo" > bar.txt      # Overwrite file with content
echo "foo" >> bar.txt     # Append to file with content
ls exists 1> stdout.txt   # Redirect the standard output to a file
ls noexist 2> stderr.txt  # Redirect the standard error output to a file
ls 2>&1 out.txt           # Redirect standard output and error to a file
ls > /dev/null           # Discard standard output and error
read foo                 # Read from standard input and write to the vari-
able foo
```

## Moving Files

```
cp foo.txt bar.txt      # Copy file
mv foo.txt bar.txt      # Move file
rsync -z|--compress -v|--verbose /foo.txt /bar    # Copy file quickly if not
changed
rsync z|--compress -v|--verbose /foo.txt /bar.txt # Copy and rename file
quickly if not changed
```

## Deleting Files

```
rm foo.txt              # Delete file
rm -f|--force foo.txt   # Delete file, ignore nonexistent files and never
prompt
```

## Reading Files

```
cat foo.txt           # Print all contents
less foo.txt          # Print some contents at a time (g - go to top of
file, SHIFT+g, go to bottom of file, /foo to search for 'foo')
head foo.txt          # Print top 10 lines of file
tail foo.txt          # Print bottom 10 lines of file
open foo.txt          # Open file in the default editor
wc foo.txt            # List number of lines words and characters in the
file
```

## File Permissions

#	PERMISSION	RWX	BINARY
7	Read, write and execute	rwX	111
6	Read and write	rw-	110
5	Read and execute	r-X	101
4	Read only	r--	100
3	Write and execute	-wX	011
2	Write only	-w-	010
1	Execute only	--X	001
0	None	---	000

For a directory, execute means you can enter a directory.

USER	GROUP	OTHERS	DESCRIPTION
6	4	4	User can read and write, everyone else can read (Default file permissions)
7	5	5	User can read, write and execute, everyone else can read and execute (Default directory permissions)

u - User

g - Group

o - Others

a - All of the above

```
ls -l /foo.sh          # List file permissions
chmod +100 foo.sh      # Add 1 to the user permission
chmod -100 foo.sh      # Subtract 1 from the user permission
chmod u+x foo.sh       # Give the user execute permission
chmod g+x foo.sh       # Give the group execute permission
chmod u-x,g-x foo.sh   # Take away the user and group execute permission
chmod u+x,g+x,o+x foo.sh # Give everybody execute permission
chmod a+x foo.sh       # Give everybody execute permission
chmod +x foo.sh        # Give everybody execute permission
```

## Finding Files

Find binary files for a command.

```
type wget              # Find the binary
which wget             # Find the binary
whereis wget           # Find the binary, source, and
manual page files
```

`locate` uses an index and is fast.

```
updatedb              # Update the index
locate foo.txt        # Find a file
locate --ignore-case  # Find a file and ignore case
locate f*.txt         # Find a text file starting with
'f'
```

`find` doesn't use an index and is slow.

```

find /path -name foo.txt           # Find a file
find /path -iname foo.txt          # Find a file with case insensi-
tive search
find /path -name "*.txt"           # Find all text files
find /path -name foo.txt -delete   # Find a file and delete it
find /path -name "*.png" -exec pngquant {} # Find all .png files and execute
pngquant on it
find /path -type f -name foo.txt    # Find a file
find /path -type d -name foo        # Find a directory
find /path -type l -name foo.txt    # Find a symbolic link
find /path -type f -mtime +30       # Find files that haven't been
modified in 30 days
find /path -type f -mtime +30 -delete # Delete files that haven't been
modified in 30 days

```

## Find in Files

```

grep 'foo' /bar.txt                # Search for 'foo' in file
'bar.txt'
grep 'foo' /bar -r|--recursive     # Search for 'foo' in directory
'bar'
grep 'foo' /bar -R|--dereference-recursive # Search for 'foo' in directory
'bar' and follow symbolic links
grep 'foo' /bar -l|--files-with-matches  # Show only files that match
grep 'foo' /bar -L|--files-without-match # Show only files that don't
match
grep 'Foo' /bar -i|--ignore-case        # Case insensitive search
grep 'foo' /bar -x|--line-regexp        # Match the entire line
grep 'foo' /bar -C|--context 1          # Add N line of context above
and below each search result
grep 'foo' /bar -v|--invert-match       # Show only lines that don't
match
grep 'foo' /bar -c|--count              # Count the number lines that
match
grep 'foo' /bar -n|--line-number        # Add line numbers
grep 'foo' /bar --colour                # Add colour to output
grep 'foo\|bar' /baz -R                 # Search for 'foo' or 'bar' in
directory 'baz'
grep --extended-regexp|-E 'foo\|bar' /baz -R # Use regular expressions
egrep 'foo\|bar' /baz -R                # Use regular expressions

```

## Replace in Files

```
sed 's/fox/bear/g' foo.txt          # Replace fox with bear in foo.txt
and output to console
sed 's/fox/bear/gi' foo.txt         # Replace fox (case insensitive)
with bear in foo.txt and output to console
sed 's/red fox/blue bear/g' foo.txt # Replace red with blue and fox
with bear in foo.txt and output to console
sed 's/fox/bear/g' foo.txt > bar.txt # Replace fox with bear in foo.txt
and save in bar.txt
sed 's/fox/bear/g' foo.txt -i|--in-place # Replace fox with bear and over-
write foo.txt
```

## Symbolic Links

```
ln -s|--symbolic foo bar           # Create a link 'bar' to the 'foo'
folder
ln -s|--symbolic -f|--force foo bar # Overwrite an existing symbolic link
'bar'
ls -l                               # Show where symbolic links are pointing
```

## Compressing Files

### zip

Compresses files.one or more files into \*.zip

```
zip foo.zip /bar.txt              # Compress bar.txt into foo.zip
zip foo.zip /bar.txt /baz.txt     # Compress bar.txt and baz.txt into
foo.zip
zip foo.zip /{bar,baz}.txt        # Compress bar.txt and baz.txt into
foo.zip
zip -r|--recurse-paths foo.zip /bar # Compress directory bar into foo.zip
```

### gzip

Compresses a single file into \*.gz files.



```
gzip /bar.txt foo.gz           # Compress bar.txt into foo.gz and then delete bar.txt
gzip -k|--keep /bar.txt foo.gz # Compress bar.txt into foo.gz
```

## tar -c

Compresses (optionally) and combines one or more files into a single `*.tar`, `*.tar.gz`, `*.tpz` or `*.tgz` file.

```
tar -c|--create -z|--gzip -f|--file=foo.tgz /bar.txt /baz.txt # Compress bar.txt and baz.txt into foo.tgz
tar -c|--create -z|--gzip -f|--file=foo.tgz /{bar,baz}.txt    # Compress bar.txt and baz.txt into foo.tgz
tar -c|--create -z|--gzip -f|--file=foo.tgz /bar              # Compress directory bar into foo.tgz
```

## Decompressing Files

### Unzip

```
unzip foo.zip                 # Unzip foo.zip into current directory
```

### Gunzip

```
gunzip foo.gz                # Unzip foo.gz into current directory and delete foo.gz
gunzip -k|--keep foo.gz      # Unzip foo.gz into current directory
```

### tar -x

```
tar -x|--extract -z|--gzip -f|--file=foo.tar.gz # Un-compress foo.tar.gz into current directory
tar -x|--extract -f|--file=foo.tar              # Un-combine foo.tar into current directory
```

## Disk Usage

```
df                                # List disks, size, used and available space
df -h|--human-readable           # List disks, size, used and available space in a hu-
man readable format
du                                # List current directory, subdirectories and file
sizes
du /foo/bar                      # List specified directory, subdirectories and file
sizes
du -h|--human-readable           # List current directory, subdirectories and file
sizes in a human readable format
du -d|--max-depth                # List current directory, subdirectories and file
sizes within the max depth
du -d 0                          # List current directory size
```

## Memory Usage

```
free                             # Show memory usage
free -h|--human                 # Show human readable memory usage
free -h|--human --si            # Show human readable memory usage in power of 1000
instead of 1024
free -s|--seconds 5             # Show memory usage and update continuously every
five seconds
```

## Packages

```
apt update                      # Refreshes repository index
apt search wget                 # Search for a package
apt show wget                   # List information about the wget package
apt list --all-versions wget    # List all versions of the package
apt install wget                # Install the latest version of the wget pack-
age
apt install wget=1.2.3          # Install a specific version of the wget pack-
age
apt remove wget                 # Removes the wget package
apt upgrade                     # Upgrades all upgradable packages
```

## Identifying Processes

```
top                # List all processes interactively
htop               # List all processes interactively
ps all            # List all processes
pidof foo         # Return the PID of all foo processes

CTRL+Z           # Suspend a process running in the foreground
bg               # Resume a suspended process and run in the back-
ground
fg               # Bring the last background process to the foreground
fg 1             # Bring the background process with the PID to the
foreground

sleep 30 &       # Sleep for 30 seconds and move the process into the
background
jobs              # List all background jobs
jobs -p           # List all background jobs with their PID

lsof              # List all open files and the process using them
lsof -itcp:4000   # Return the process listening on port 4000
```

## Process Priority

Process priorities go from -20 (highest) to 19 (lowest).

```
nice -n -20 foo   # Change process priority by name
renice 20 PID      # Change process priority by PID
ps -o ni PID      # Return the process priority of PID
```

## Killing Processes

```
CTRL+C          # Kill a process running in the foreground
kill PID        # Shut down process by PID gracefully. Sends TERM
                # signal.
kill -9 PID     # Force shut down of process by PID. Sends SIGKILL
                # signal.
pkill foo       # Shut down process by name gracefully. Sends TERM
                # signal.
pkill -9 foo    # force shut down process by name. Sends SIGKILL sig-
                # nal.
killall foo     # Kill all process with the specified name grace-
                # fully.
```

## Date & Time

```
date            # Print the date and time
date --iso-8601 # Print the ISO8601 date
date --iso-8601=ns # Print the ISO8601 date and time
time tree       # Time how long the tree command takes to execute
```

## Scheduled Tasks

*	*	*	*	*
Minute	Hour	Day of month	Month	Day of week

```

crontab -l                # List cron tab
crontab -e                # Edit cron tab in Vim
crontab /path/crontab    # Load cron tab from a file
crontab -l > /path/crontab # Save cron tab to a file

* * * * * foo            # Run foo every minute
*/15 * * * * foo         # Run foo every 15 minutes
0 * * * * foo            # Run foo every hour
15 6 * * * foo           # Run foo daily at 6:15 AM
44 4 * * 5 foo           # Run foo every Friday at 4:44 AM
0 0 1 * * foo            # Run foo at midnight on the first of the month
0 0 1 1 * foo            # Run foo at midnight on the first of the year

at -l                    # List scheduled tasks
at -c 1                  # Show task with ID 1
at -r 1                  # Remove task with ID 1
at now + 2 minutes       # Create a task in Vim to execute in 2 minutes
at 12:34 PM next month   # Create a task in Vim to execute at 12:34 PM
next month
at tomorrow              # Create a task in Vim to execute tomorrow

```

## HTTP Requests

```

curl https://example.com # Return response
body
curl -i|--include https://example.com # Include status code
and HTTP headers
curl -L|--location https://example.com # Follow redirects
curl -o|--remote-name foo.txt https://example.com # Output to a text
file
curl -H|--header "User-Agent: Foo" https://example.com # Add a HTTP header
curl -X|--request POST -H "Content-Type: application/json" -d|--data
'{"foo":"bar"}' https://example.com # POST JSON
curl -X POST -H --data-urlencode foo="bar" http://example.com
# POST URL Form Encoded

wget https://example.com/file.txt . # Download a
file to the current directory
wget -O|--output-document foo.txt https://example.com/file.txt # Output to a
file with the specified name

```

## Network Troubleshooting

```
ping example.com          # Send multiple ping requests using the ICMP
protocol
ping -c 10 -i 5 example.com # Make 10 attempts, 5 seconds apart

ip addr                   # List IP addresses on the system
ip route show             # Show IP addresses to router

netstat -i|--interfaces  # List all network interfaces and in/out usage
netstat -l|--listening   # List all open ports

traceroute example.com    # List all servers the network traffic goes
through

mtr -w|--report-wide example.com # Con-
tinually list all servers the network traffic goes through
mtr -r|--report -w|--report-wide -c|--report-cycles 100 example.com # Output
a report that lists network traffic 100 times

nmap 0.0.0.0              # Scan for the 1000 most common open ports on
localhost
nmap 0.0.0.0 -p1-65535    # Scan for open ports on localhost between 1 and
65535
nmap 192.168.4.3          # Scan for the 1000 most common open ports on a
remote IP address
nmap -sP 192.168.1.1/24   # Discover all machines on the network by
ping'ing them
```

## DNS

```
host example.com          # Show the IPv4 and IPv6 addresses
dig example.com           # Show complete DNS information
cat /etc/resolv.conf      # resolv.conf lists nameservers
```

## Hardware

```
lsusb                     # List USB devices
lspci                     # List PCI hardware
lshw                      # List all hardware
```

## Terminal Multiplexers

Start multiple terminal sessions. Active sessions persist reboots. tmux is more modern than `screen`.

```
tmux                # Start a new session (CTRL-b + d to detach)
tmux ls              # List all sessions
tmux attach -t 0     # Reattach to a session

screen              # Start a new session (CTRL-a + d to detach)
screen -ls          # List all sessions
screen -R 31166     # Reattach to a session

exit                # Exit a session
```

## Secure Shell Protocol (SSH)

```
ssh hostname        # Connect to hostname using your current user
                    # name over the default SSH port 22
ssh -i foo.pem hostname # Connect to hostname using the identity file
ssh user@hostname   # Connect to hostname using the user over the
                    # default SSH port 22
ssh user@hostname -p 8765 # Connect to hostname using the user over a
                    # custom port
ssh ssh://user@hostname:8765 # Connect to hostname using the user over a
                    # custom port
```

Set default user and port in `~/.ssh/config`, so you can just enter the name next time:

```
$ cat ~/.ssh/config
Host name
  User foo
  Hostname 127.0.0.1
  Port 8765
$ ssh name
```

## Secure Copy

```
scp foo.txt ubuntu@hostname:/home/ubuntu # Copy foo.txt into the specified
remote directory
```

## Bash Profile

```
bash - .bashrc
zsh - .zshrc
```

```
# Always run ls after cd
function cd {
    builtin cd "$@" && ls
}

# Prompt user before overwriting any files
alias cp='cp --interactive'
alias mv='mv --interactive'
alias rm='rm --interactive'

# Always show disk usage in a human readable format
alias df='df -h'
alias du='du -h'
```

## Bash Script

### Variables

```
#!/bin/bash

foo=123                # Initialize variable foo with 123
declare -i foo=123     # Initialize an integer foo with 123
declare -r foo=123     # Initialize readonly variable foo with 123
echo $foo              # Print variable foo
echo ${foo}_bar        # Print variable foo followed by _bar
echo ${foo:-'default'} # Print variable foo if it exists otherwise print default

export foo             # Make foo available to child processes
unset foo              # Make foo unavailable to child processes
```

### Environment Variables

```
#!/bin/bash

env                    # List all environment variables
echo $PATH             # Print PATH environment variable
export FOO=Bar         # Set an environment variable
```



## Functions

```
#!/bin/bash

greet() {
    local world = "World"
    echo "$1 $world"
    return "$1 $world"
}
greet "Hello"
greeting=$(greet "Hello")
```

## Exit Codes

```
#!/bin/bash

exit 0    # Exit the script successfully
exit 1    # Exit the script unsuccessfully
echo $?   # Print the last exit code
```

## Conditional Statements

### Boolean Operators

`$foo` - Is true

`!$foo` - Is false

### Numeric Operators

`-eq` - Equals

`-ne` - Not equals

`-gt` - Greater than

`-ge` - Greater than or equal to

`-lt` - Less than

`-le` - Less than or equal to

`-e foo.txt` - Check file exists

`-z foo` - Check if variable exists

### String Operators

`=` - Equals

**==** - Equals

**-z** - Is null

**-n** - Is not null

**<** - Is less than in ASCII alphabetical order

**>** - Is greater than in ASCII alphabetical order

## If Statements

```
#!/bin/bash

if [[ $foo = 'bar' ]]; then
    echo 'one'
elif [[ $foo = 'bar' ]] || [[ $foo = 'baz' ]]; then
    echo 'two'
elif [[ $foo = 'ban' ]] && [[ $USER = 'bat' ]]; then
    echo 'three'
else
    echo 'four'
fi
```

## Inline If Statements

```
#!/bin/bash

[[ $USER = 'rehan' ]] && echo 'yes' || echo 'no'
```

## While Loops

```
#!/bin/bash

declare -i counter
counter=10
while [ $counter -gt 2 ]; do
    echo The counter is $counter
    counter=counter-1
done
```

## For Loops

```
#!/bin/bash

for i in {0..10..2}
do
    echo "Index: $i"
done

for filename in file1 file2 file3
do
    echo "Content: " >> $filename
done

for filename in *;
do
    echo "Content: " >> $filename
done
```

## Case Statements

```
#!/bin/bash

echo "What's the weather like tomorrow?"
read weather

case $weather in
    sunny | warm ) echo "Nice weather: " $weather
    ;;
    cloudy | cool ) echo "Not bad weather: " $weather
    ;;
    rainy | cold ) echo "Terrible weather: " $weather
    ;;
    * ) echo "Don't understand"
    ;;
esac
```

## Parallel

```
parallel <command> ::: <list of files> # run a command on multiple files in parallel
parallel <command> <args> ::: <list of arguments> # run a command on multiple arguments in parallel
parallel <command 1> & <command 2> & ... # run multiple commands in parallel
parallel -j <number of processes> <command> <args> ::: <list of arguments> # specify the number of parallel processes to use
parallel -o <output file pattern> <command> <args> ::: <list of arguments> # save the output of the commands to separate files
parallel -e <error file pattern> <command> <args> ::: <list of arguments> # save the error output of the commands to separate files, use the -e option
parallel -env <variable name> <command> <args> ::: <list of arguments> # pass environment variables to the commands, use the -env option
```

## Regular Expressions

### Anchors

<code>^</code>	Start of string, or start of line in multi-line pattern
<code>\$</code>	End of string, or end of line in multi-line pattern
<code>\b</code>	Word boundary
<code>\B</code>	Not word boundary

### Character Classes

<code>[ABC]</code>	Match any character in the set.
<code>[^ABC]</code>	Match any character that is not in the set.
<code>[A-Z]</code>	Matches a character having a character code between the two specified characters inclusive.
<code>.</code>	Matches any character except linebreaks. Equivalent to <code>[\n\r]</code> .
<code>[\s\S]</code>	A character set that can be used to match any character, including line breaks, without the <code>dotall</code> flag ( <code>s</code> ). An alternative is <code>^</code> , but it is not supported in all browsers.
<code>\w</code>	Matches any word character (alphanumeric & underscore). Only matches low-ascii characters (no accented or non-roman characters). Equivalent to <code>[A-Za-z0-9_]</code>
<code>\W</code>	Matches any character that is not a word character (alphanumeric & underscore). Equivalent to <code>^[^A-Za-z0-9_]</code>
<code>\d</code>	Matches any digit character (0-9). Equivalent to <code>[0-9]</code> .
<code>\D</code>	Matches any character that is not a digit character (0-9). Equivalent to <code>^[^0-9]</code> .
<code>\s</code>	Matches any whitespace character (spaces, tabs, line breaks).
<code>\S</code>	Matches any character that is not a whitespace character (spaces, tabs, line breaks).
<code>\p{L}</code>	Matches a character in the specified unicode category. For example, <code>\p{Ll}</code> will match any lowercase letter.
<code>\P{L}</code>	Matches any character that is not in the specified unicode category.
<code>\p{Han}</code>	Matches any character in the specified unicode script. For example, <code>\p{Arabic}</code> will match characters in the Arabic script.
<code>\P{Han}</code>	Matches any character that is not in the specified unicode script.

## Escaped Characters

<code>\+</code>	The following character have special meaning, and should be preceded by a <code>\</code> (backslash) to represent a literal character: <code>+ * ? ^ \$ \ . [ ] { } ( )   /</code> Within a character set, only <code>\</code> , <code>-</code> , and <code>]</code> need to be escaped.
<code>\000</code>	Octal escaped character in the form <code>\000</code> . Value must be less than 255 ( <code>\377</code> ).
<code>\xFF</code>	Hexadecimal escaped character in the form <code>\xFF</code> .
<code>\uFFFF</code>	Unicode escaped character in the form <code>\uFFFF</code>
<code>\u{FFFF}</code>	Unicode escaped character in the form <code>\u{FFFF}</code> . Supports a full range of unicode point escapes with any number of hex digits. Requires the unicode flag ( <code>u</code> ).
<code>\cI</code>	Escaped control character in the form <code>\cZ</code> . This can range from <code>\cA</code> (SOH, char code 1) to <code>\cZ</code> (SUB, char code 26).
<code>\t</code>	Matches a TAB character (char code 9).
<code>\n</code>	Matches a LINE FEED character (char code 10).
<code>\v</code>	Matches a VERTICAL TAB character (char code 11).
<code>\f</code>	Matches a FORM FEED character (char code 12).
<code>\r</code>	Matches a CARRIAGE RETURN character (char code 13).
<code>\0</code>	Matches a NULL character (char code 0).

## Groups & References

<code>(ABC)</code>	Groups multiple tokens together and creates a capture group for extracting a substring or using a backreference.
<code>(?&lt;name&gt;ABC)</code>	Creates a capturing group that can be referenced via the specified name.
<code>\1</code>	Matches the results of a capture group. For example <code>\1</code> matches the results of the first capture group & <code>\3</code> matches the third.
<code>(?:ABC)</code>	Groups multiple tokens together without creating a capture group.

## Lookaround

<code>(?=ABC)</code>	Matches a group after the main expression without including it in the result.
<code>(?!ABC)</code>	Specifies a group that can not match after the main expression (if it matches, the result is discarded).
<code>(?&lt;=ABC)</code>	Matches a group before the main expression without including it in the result.
<code>(?&lt;!ABC)</code>	Specifies a group that can not match before the main expression (if it matches, the result is discarded).

## Quantifiers & Alternation

<code>+</code>	Matches 1 or more of the preceding token.
<code>*</code>	Matches 0 or more of the preceding token.
<code>{1,3}</code>	Matches the specified quantity of the previous token. <code>{1,3}</code> will match 1 to 3. <code>{3}</code> will match exactly 3. <code>{3,}</code> will match 3 or more.
<code>?</code>	Matches 0 or 1 of the preceding token, effectively making it optional.
<code>?</code>	Makes the preceding quantifier lazy, causing it to match as few characters as possible. By default, quantifiers are greedy, and will match as many characters as possible.
<code>/</code>	Acts like a boolean OR. Matches the expression before or after the <code> </code> . It can operate within a group, or on a whole expression. The patterns will be tested in order.

## Substitution

<code>\$&amp;</code>	Inserts the matched text.
<code>\$1</code>	Inserts the results of the specified capture group. For example, <code>\$3</code> would insert the third capture group.
<code>\$`</code>	Inserts the portion of the source string that precedes the match.
<code>\$'</code>	Inserts the portion of the source string that follows the match.
<code>\$\$</code>	Inserts a dollar sign character (\$).
<code> n</code>	For convenience, these escaped characters are supported in the Replace string in RegExr: <code>\n</code> , <code>\r</code> , <code>\t</code> , <code>\</code> , and unicode escapes <code>\uFFFF</code> . This may vary in your deploy environment.

## Flags

<code>i</code>	Makes the whole expression case-insensitive. For example, <code>/aBc/i</code> would match <code>AbC</code> .
<code>g</code>	Retain the index of the last match, allowing subsequent searches to start from the end of the previous match. Without the global flag, subsequent searches will return the same match.
<code>m</code>	When the multiline flag is enabled, beginning and end anchors (^ and \$) will match the start and end of a line, instead of the start and end of the whole string. Note that patterns such as <code>/^[s\S]+\$/m</code> may return matches that span multiple lines because the anchors will match the start/end of any line.
<code>u</code>	When the unicode flag is enabled, you can use extended unicode escapes in the form <code>\x{FFFFF}</code> . It also makes other escapes stricter, causing unrecognized escapes (ex. <code>\j</code> ) to throw an error.
<code>y</code>	The expression will only match from its <code>lastIndex</code> position and ignores the global ( <code>g</code> ) flag if set. Because each search in RegExr is discrete, this flag has no further impact on the displayed results.
<code>s</code>	Dot ( <code>.</code> ) will match any character, including newline.

## AWK command

### Default behavior of Awk

```
awk '{print}' employee.txt # By default Awk prints every line of data from the specified file.
```

### Print the lines which match the given pattern.

```
awk '/manager/ {print}' employee.txt # The awk command prints all the line which matches with the 'manager'.
```

## Splitting a Line Into Fields

```
awk '{print $1,$4}' employee.txt # For each record i.e line, the awk command splits the record delimited by whitespace character by default and stores it in the $n variables. If the line has 4 words, it will be stored in $1, $2, $3 and $4 respectively. Also, $0 represents the whole line.
```

## Built-In Variables In Awk

Awk's built-in variables include the field variables—\$1, \$2, \$3, and so on (\$0 is the entire line) — that break a line of text into individual words or pieces called fields.

- + NR: NR command keeps a current count of the number of input records. Remember that records are usually lines. Awk command performs the pattern/action statements once for each record in a file.
- + NF: NF command keeps a count of the number of fields within the current input record.
- + FS: FS command contains the field separator character which is used to divide fields on the input line. The default is “white space”, meaning space and tab characters. FS can be reassigned to another character (typically in BEGIN) to change the field separator.
- + RS: RS command stores the current record separator character. Since, by default, an input line is the input record, the default record separator character is a newline.
- + OFS: OFS command stores the output field separator, which separates the fields when Awk prints them. The default is a blank space. Whenever print has several parameters separated with commas, it will print the value of OFS in between each parameter.
- + ORS: ORS command stores the output record separator, which separates the output lines when Awk prints them. The default is a newline character. print automatically outputs the contents of ORS at the end of whatever it is given to print.

```

awk '{print NR,$0}' employee.txt # the awk command with NR prints all the
lines along with the line number.
awk '{print $1,$NF}' employee.txt # $NF represents last field.
awk 'NR==3, NR==6 {print NR,$0}' employee.txt # Display Line From 3 to 6
awk '{print NR "- " $1 }' geeksforgeeks.txt # print the first item along
with the row number(NR) separated with " - " from each line in geeksfor-
geeks.txt
awk 'NF == 0 {print NR}' geeksforgeeks.txt # print any empty line if pre-
sent also the line number
awk '{ if (length($0) > max) max = length($0) } END { print max }' geeksfor-
geeks.txt # find the length of the longest line present in the file
awk 'END { print NR }' geeksforgeeks.txt # count the lines in a file
awk 'length($0) > 10' geeksforgeeks.txt # Printing lines with more than 10
characters
awk '{ if($3 == "B6") print $0;}' geeksforgeeks.txt # find/check for any
string in any specific column
awk 'BEGIN { for(i=1;i<=6;i++) print "square of", i, "is",i*i; }' # print
the squares of first numbers from 1 to n say 6

```



## Parameter Expansion

```
${parameter:-word} # This form returns the value of parameter if it is set
and non-empty, otherwise it returns word.
${parameter:=word} # This form sets the value of parameter to word if param-
eter is unset or empty, and returns the value of parameter.
${parameter:?word} # This form returns an error message containing word if
parameter is unset or empty.
${parameter:+word} # This form returns word if parameter is set and non-
empty, otherwise it returns an empty string.
${parameter:offset} # This form returns the substring of parameter starting
from the offset position.
${parameter:offset:length} # This form returns the substring of parameter
starting from the offset position and with a length of length.
${!prefix*} # This form returns all variable names that begin with prefix.
${!prefix@} # This form returns all variable names that begin with prefix in
a list.
${!name[@]} # This form returns all elements of the array name.
${!name[*]} # This form returns all elements of the array name as a single
string.
${#parameter} # This form returns the length of the value of parameter.
${parameter%word} # This form returns the value of parameter with the short-
est matching word removed from the end.
${parameter%%word} # This form returns the value of parameter with the long-
est matching word removed from the end.
${parameter/pattern/string} # This form returns the value of parameter with
the first occurrence of pattern replaced by string.
${parameter^pattern} # This form returns the value of parameter with the
first character of the first word that matches pattern converted to upper-
case.
${parameter^^pattern} # This form returns the value of parameter with all
characters of all words that match pattern converted to uppercase.
${parameter,pattern} # This form returns the value of parameter with the
first character of the first word that matches pattern converted to lower-
case.
${parameter,,pattern} # This form returns the value of parameter with all
characters of all words that match pattern converted to lowercase.
${parameter@operator} # This form expands parameter based on the value of
operator. The available operators are "Q" (quote), "E" (escape), "P" (path-
name), and "A" (array).
```

## References

1. <https://github.com/RehanSaeed/Bash-Cheat-Sheet>
2. <https://regexr.com/>
3. <https://www.geeksforgeeks.org/awk-command-unixlinux-examples/>
4. <https://www.gnu.org/software/bash/manual/bash.html>
5. <http://tldp.org/LDP/Bash-Beginners-Guide/html/index.html>