Moving Around

h	move cursor left
j	move cursor down
k	mover cursor up
l	move cursor right
O	move cursor beginning line
^	move cursor to first non-whitespace of line
$	move cursor to end of line
gg	move cursor to beginning of file
G	move cursor to end of file
5G	move cursor to line number 5

Search and replace

/	search forward for pattern
?	search backward for pattern
n	repeat the last search in the same direction
N	repeat the last search in opposite direction
:%s/pattern/replacement/g	replace all occurrences of ‘pattern’ with ‘replacement’

Splits and tabs

:sp	split horizontally
:vsp	split vertically
:tabe	create new tab
:tabc	close current tab
:tabn	go to next tab
:tabp	go to previous tab

File Management

:e filename	open file for edit
:w	save file
:w filename	save to filename
:q	close current file
:wq	save and close
:q!	close without saving
:wa	save all buffers to disk

Visual Mode Commands

v	enter visual mode and select char by char
V	enter visual mode and select line by line
y	yank (copy) selection
d	delete selection
c	change selection and enter insert mode

Macros and registers

qa	start recording macro in register a
q	stop recording macro
@a	execute macro stored in ‘a’
“	access a specific register
“ay	yank into register ‘a’

https://www.linkedin.com/in/ankeorum/

Editing commands

i	enter insert mode before cursor	a	enter insert mode after the cursor
I	enter inser mode at the beginning of line	A	enter insert mode at end of line
o	insert a new line below current and go insert mode	O	insert new line above and go inser mode
dd	delete current line	D	delete from cursor to the end of the line
C	change from the cursor to the end of the line and enter edit mode	u	undo last change
cw	change from curso to end of word and enter insert mode	cb	change from cursor to beginning of word and enter insert mode

Bash cheat sheet: (), {}, $(()), $(), ${}, [], [[]]

```
( ls /home/user; whoami )
```

Executes a list of commands in a separate subshell. The commands inside the parentheses run in a child process, isolated from the main shell.

```
{ cd /var/log; ls }
```

Executes a group of commands in the same shell process. Curly braces group commands together to be executed sequentially in the current environment.

```
files=(log.txt log2.txt log.txt)
```

Creates an array of values. Parentheses are used to define an array, allowing multiple elements to be stored in one variable.

```
result=$((5 * 3 + 1))
```

Performs arithmetic calculations. The double parentheses are used for math operations, such as addition, multiplication, etc.

```
output=$(grep "error" /var/log/syslog)
```

Executes a command and captures its output. Command substitution allows the result of a command (in this case, `grep`) to be stored in a variable.

```
if [ -f /etc/passwd ]
```

Tests a condition using single brackets. The `[` and `]` denote a test command that checks conditions, such as whether a file exists.

```
if [[ $USER == "root" ]]
```

Tests a condition using double brackets. Double brackets are more flexible in bash, supporting advanced pattern matching and logical operators.

```
backup_{1..4}.tar.gz
```

Expands to multiple strings. Brace expansion is a powerful way to generate sequences or multiple strings, useful for batch operations.

```
${username}
```

Accesses a variable's value. This is another way to reference a variable, commonly used when you need to follow it with additional characters or text.

```
${filename%.txt}.bak
```

Modifies variable content. Parameter expansion allows you to alter a variable's value, such as changing a file extension from `.txt` to `.bak`.

sysxplore.com