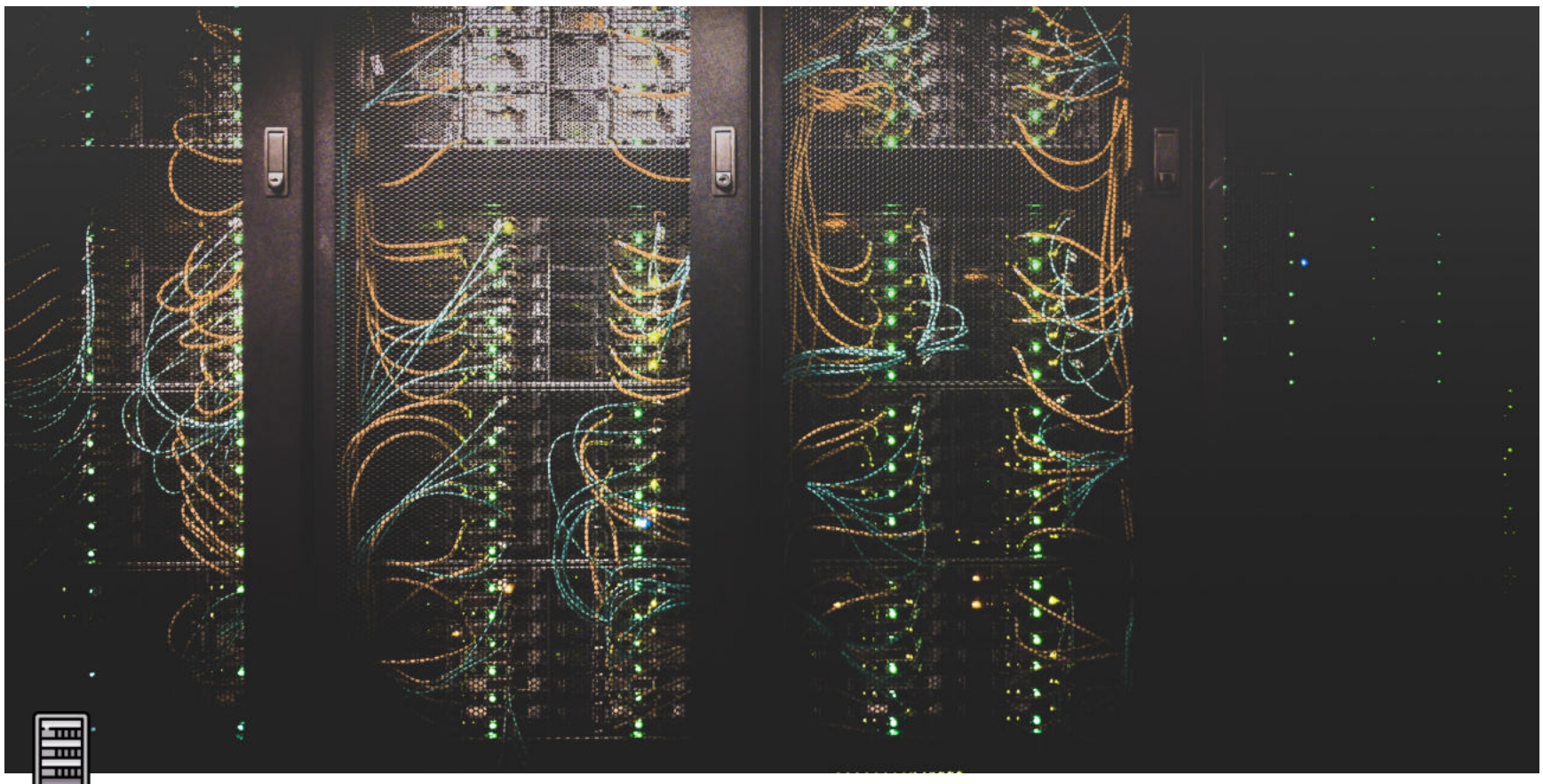
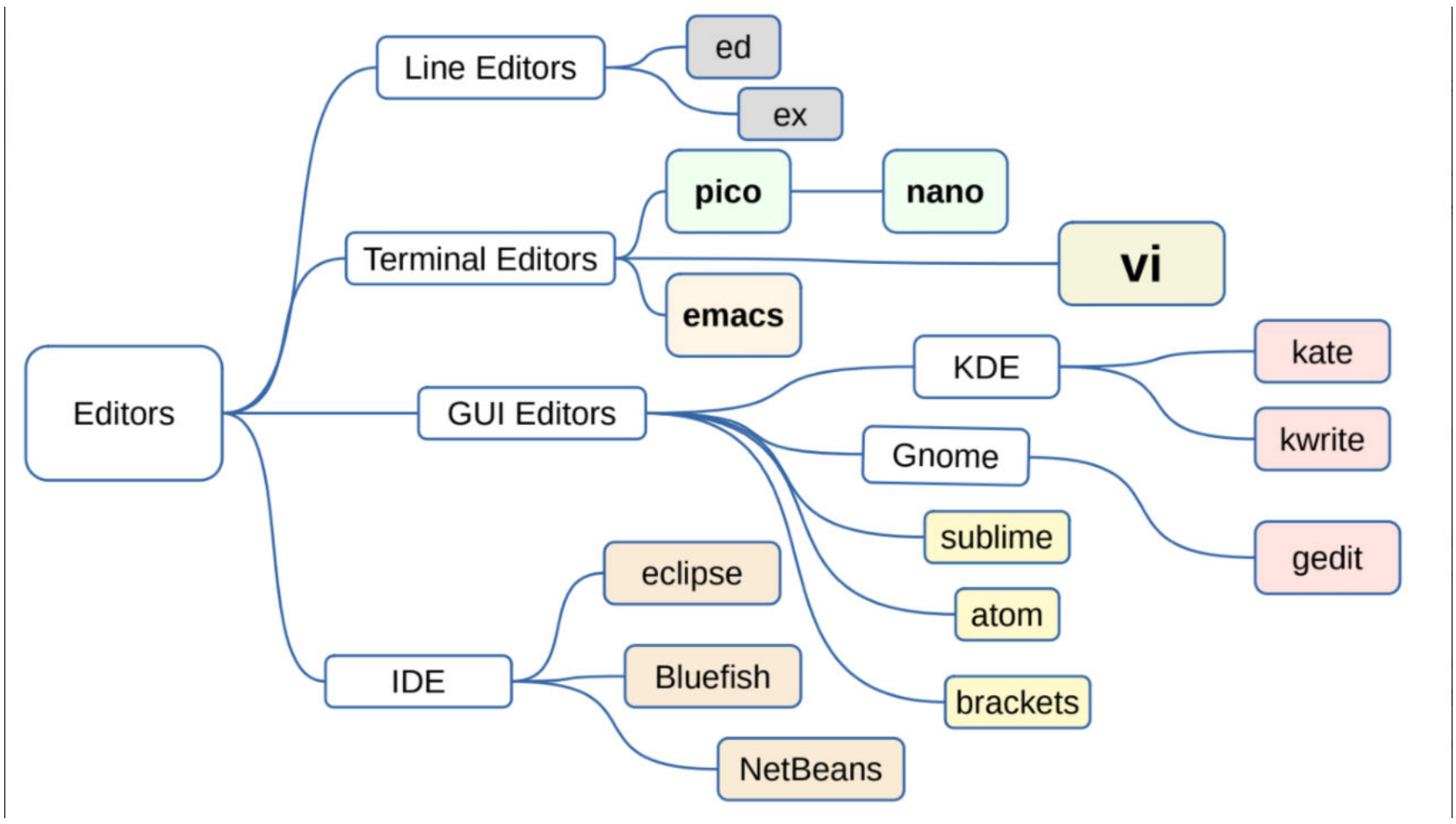


Command Line Editors - pt. 1

Type	Lecture
Date	@February 2, 2022
Lecture #	1
Lecture URL	https://youtu.be/NIIZ1cgrO7g
Notion URL	https://21f1003586.notion.site/Command-Line-Editors-pt-1-d1e3cf98aed64ed78a8e621b73e43636
Week #	5

Command line editors

Working with text files in the terminal



Features

- Scrolling, view modes, current position in the file

- Navigation (char, word, line, pattern)
- Insert, Replace, Delete
- Cut-Copy-Paste
- Search-Replace
- Language-aware syntax highlighting
- Key-maps, init scripts, macros
- Plugins

`ed`

Show the Prompt	P
Command Format	`[addr[,addr]]cmd[params]`
commands for location	*2* . $ % + - , ; /RE/
commands for editing	f p a c d i j s m u
execute a shell *command*	*!command*
edit a file	e *filename*
read file contents into buffer	r *filename*
read *command* output into buffer	r *!command*
write buffer to filename	w *filename*
quit	q

Command Format → `[starting-address[,ending-address]command[command-parameters]]`

To locate the cursor on any particular line of the file ...

We can use the line number itself

- press **2** → in the 2nd line of the text file
- press **. (dot)** → referring to the current line the cursor is
- press **$** → refers to the last line
- press **%** → refers to all the lines, i.e. any action we are doing applies to all the lines
- press **+** → line after the cursor
- press **- (minus sign)** → line before the cursor
- press **, (comma)** → represent the entire buffer, i.e. the whole file
- press **; (semicolon)** → refers the end of the text file from the current position
- `/RE/` → To match a specific Regular Expression in the file

A way to invoke the `ed` editor

```
ed file.txt
```

and it shows the number of bytes in the file, instead of the contents of the file ... *ayo wut*

```
~/Documents/week5 ed test.txt
117
P  This gives a prompt
*1  Move the cursor to line #1
line-1 hello world
*$  Move the cursor to end of the buffer
line-4 end of file
*,p Show all the lines in the file
line-1 hello world
line-2 welcome to line editor
line-3 ed is perhaps the oldest editor out there
line-4 end of file
*2,3p  Show lines from 2 to 3
line-2 welcome to line editor
line-3 ed is perhaps the oldest editor out there
*/hello/  Find the line with the pattern “hello”
line-1 hello world
*/oldest/ Find the line with the pattern “oldest”
line-3 ed is perhaps the oldest editor out there
*1 Move to line #1
line-1 hello world
*+ Move to the next line
line-2 welcome to line editor
*- Move to the previous line
line-1 hello world
*3 Move to line #3
line-3 ed is perhaps the oldest editor out there
*;p Get all the lines from current line to the end of file
line-3 ed is perhaps the oldest editor out there
line-4 end of file
*%p Display all the lines of the file
line-1 hello world
line-2 welcome to line editor
line-3 ed is perhaps the oldest editor out there
line-4 end of file
*. Get the current line
line-4 end of file
```

Run a bash command, read the bash command’s output and write it to the file

```
~/Documents/week5 ed test.txt
117
P
*!date
Tuesday 01 February 2022 09:03:06 PM IST
!
*r !date
41
*w
158
*,p
line-1 hello world
line-2 welcome to line editor
line-3 ed is perhaps the oldest editor out there
line-4 end of file
Tuesday 01 February 2022 09:03:11 PM IST
*
```

Delete the last line of the file and write the changes to the file

```
~/Documents/week5 ed test.txt
158
,p
line-1 hello world
line-2 welcome to line editor
line-3 ed is perhaps the oldest editor out there
line-4 end of file
Tuesday 01 February 2022 09:03:11 PM IST
$
Tuesday 01 February 2022 09:03:11 PM IST
.d
P
*,p
line-1 hello world
line-2 welcome to line editor
line-3 ed is perhaps the oldest editor out there
line-4 end of file
*w
117
*q
```

Append a line to the file (press . then enter to exit)

```
~/Documents/week5 ed test.txt
117
P
*,p
line-1 hello world
line-2 welcome to line editor
line-3 ed is perhaps the oldest editor out there
line-4 end of file
*1
line-1 hello world
*a
appended this line after line-1
.
*,p
line-1 hello world
appended this line after line-1
line-2 welcome to line editor
line-3 ed is perhaps the oldest editor out there
line-4 end of file
*
```

Search and Replace

```
*2
appended this line after the first line
*s/appended/Appended
Appended this line after the first line
*,p
line-1 hello world
Appended this line after the first line
line-2 welcome to line editor
line-3 ed is perhaps the oldest editor out there
line-4 end of file
```

File name

```
*f
test.txt
```

Print the current line

```
*p
line-4 end of file
```

Append to the current line

```
*a
This line is appended at the end of the file
.
```

Join line 5 and 6 (The command is `5,6j`)

```
*5
line-4 end of file
*5,6j
*p
line-4 end of fileThis line is appended at the end of the file
*.
line-4 end of fileThis line is appended at the end of the file
```

Move the current line after the given line # (*We are moving the current line below line 1 here*)

```
*m1
*,p
line-1 hello world
line-4 end of fileThis line is appended at the end of the file
Appended this line after the first line
line-2 welcome to line editor
line-3 ed is perhaps the oldest editor out there
```

Undo the previous operation

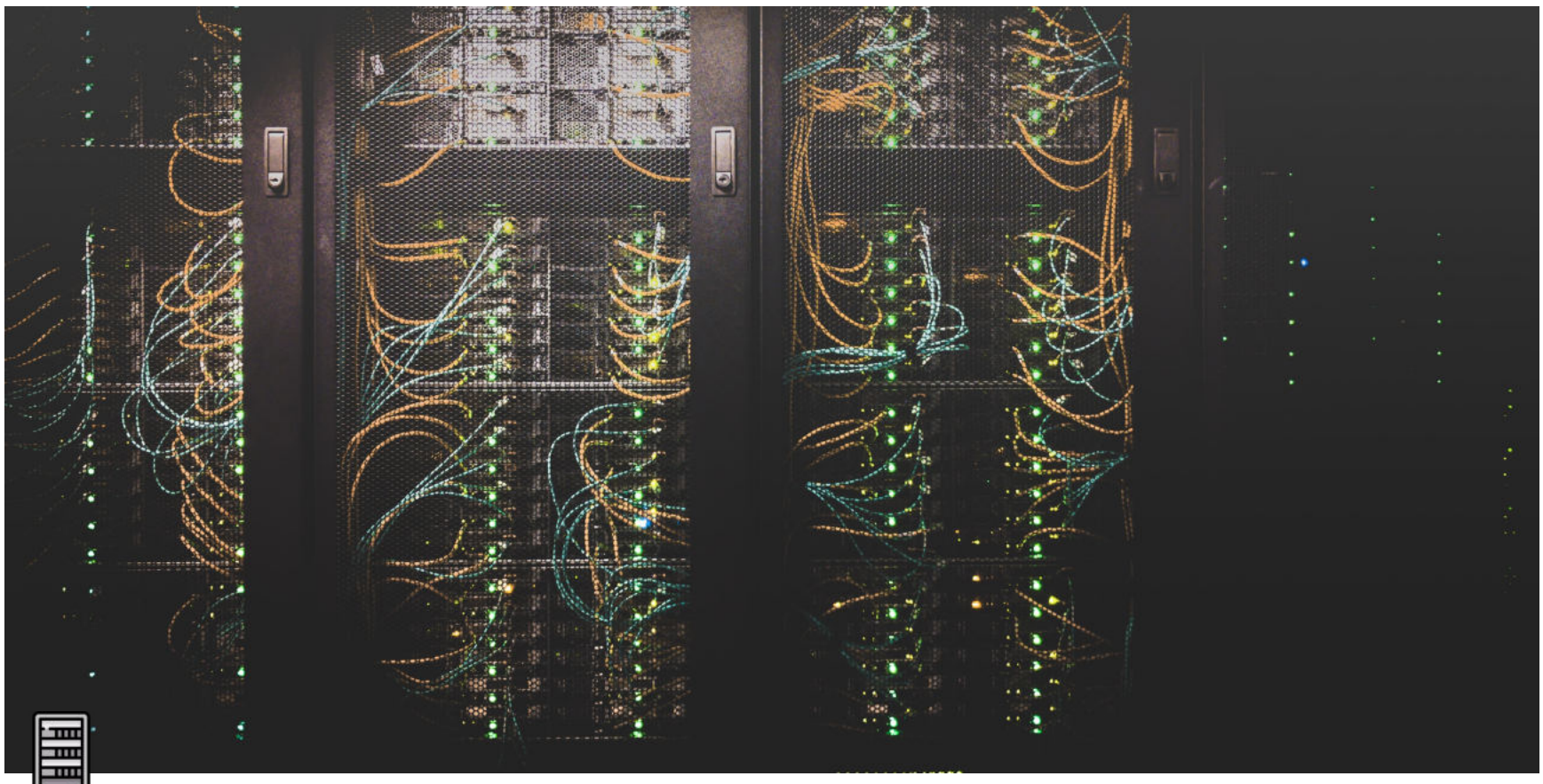
```
*u
*,p
line-1 hello world
Appended this line after the first line
line-2 welcome to line editor
line-3 ed is perhaps the oldest editor out there
line-4 end of fileThis line is appended at the end of the file
```

Add a prefix to all the lines and then modify the prefix on some lines

```
~/Documents/week5 ed test.txt
201
P
*,p
line-1 hello world
Appended this line after the first line
line-2 welcome to line editor
line-3 ed is perhaps the oldest editor out there
line-4 end of fileThis line is appended at the end of the file
*%s/\(.*\)/PREFIX \1/
*,p
PREFIX line-1 hello world
PREFIX Appended this line after the first line
PREFIX line-2 welcome to line editor
PREFIX line-3 ed is perhaps the oldest editor out there
PREFIX line-4 end of fileThis line is appended at the end of the file
*3,5s/PREFIX/prefix/
*,p
PREFIX line-1 hello world
PREFIX Appended this line after the first line
prefix line-2 welcome to line editor
prefix line-3 ed is perhaps the oldest editor out there
prefix line-4 end of fileThis line is appended at the end of the file
```

`ed/ex` commands

f	show name of file being edited
p	print the current line
a	append at the current line
c	change the line
d	delete the current line
i	insert line at the current position
j	join lines
s	search for regex pattern
m	move current line to position
u	undo latest change



Command Line Editors - pt. 2

Type	📒 Lecture
Date	@February 2, 2022
Lecture #	2
Lecture URL	https://youtu.be/HLhza4vZTsI
Notion URL	https://21f1003586.notion.site/Command-Line-Editors-pt-2-258a6a2baeaa4ec59dee72011a8382b6
Week #	5

`readlink`

Prints the resolved symbolic links or canonical file names, *but what does that mean?*

If we have file which is a symbolic link to file which in turn is another symbolic link to a file and so on ...

The `readlink` command will display the actual file the initial symbolic link is referring to

Example usage

```
readlink -f /usr/bin/pico
```

Output

```
/usr/bin/nano
```

`.bashrc`

It is a config file that is read by the bash shell everytime it opens

`nano`

`nano` is a text editor, example syntax is `nano filename`

It also does syntax highlighting

File handling

Ctrl+S	Save current file
Ctrl+O	Offer to write file ("Save as")
Ctrl+R	Insert a file into current one
Ctrl+X	Close buffer, exit from nano

Editing

Ctrl+K	Cut current line into cutbuffer
Alt+6	Copy current line into cutbuffer
Ctrl+U	Paste contents of cutbuffer
Alt+T	Cut until end of buffer
Ctrl+]	Complete current word
Alt+3	Comment/uncomment line/region
Alt+U	Undo last action
Alt+E	Redo last undone action

Search and replace

Ctrl+Q	Start backward search
Ctrl+W	Start forward search
Alt+Q	Find next occurrence backward
Alt+W	Find next occurrence forward
Alt+R	Start a replacing session

Deletion

Ctrl+H	Delete character before cursor
Ctrl+D	Delete character under cursor
Alt+Bsp	Delete word to the left
Ctrl+Del	Delete word to the right
Alt+Del	Delete current line

Operations

Ctrl+T	Execute some command
Ctrl+J	Justify paragraph or region
Alt+J	Justify entire buffer
Alt+B	Run a syntax check
Alt+F	Run a formatter/fixer/arranger
Alt+:	Start/stop recording of macro
Alt+;	Replay macro

Moving around

Ctrl+B	One character backward
Ctrl+F	One character forward
Ctrl+←	One word backward
Ctrl+→	One word forward
Ctrl+A	To start of line
Ctrl+E	To end of line
Ctrl+P	One line up
Ctrl+N	One line down
Ctrl+↑	To previous block
Ctrl+↓	To next block
Ctrl+Y	One page up
Ctrl+V	One page down
Alt+\	To top of buffer
Alt+/	To end of buffer

Special movement

Alt+G	Go to specified line
Alt+]	Go to complementary bracket
Alt+↑	Scroll viewport up
Alt+↓	Scroll viewport down
Alt+<	Switch to preceding buffer
Alt+>	Switch to succeeding buffer

Information

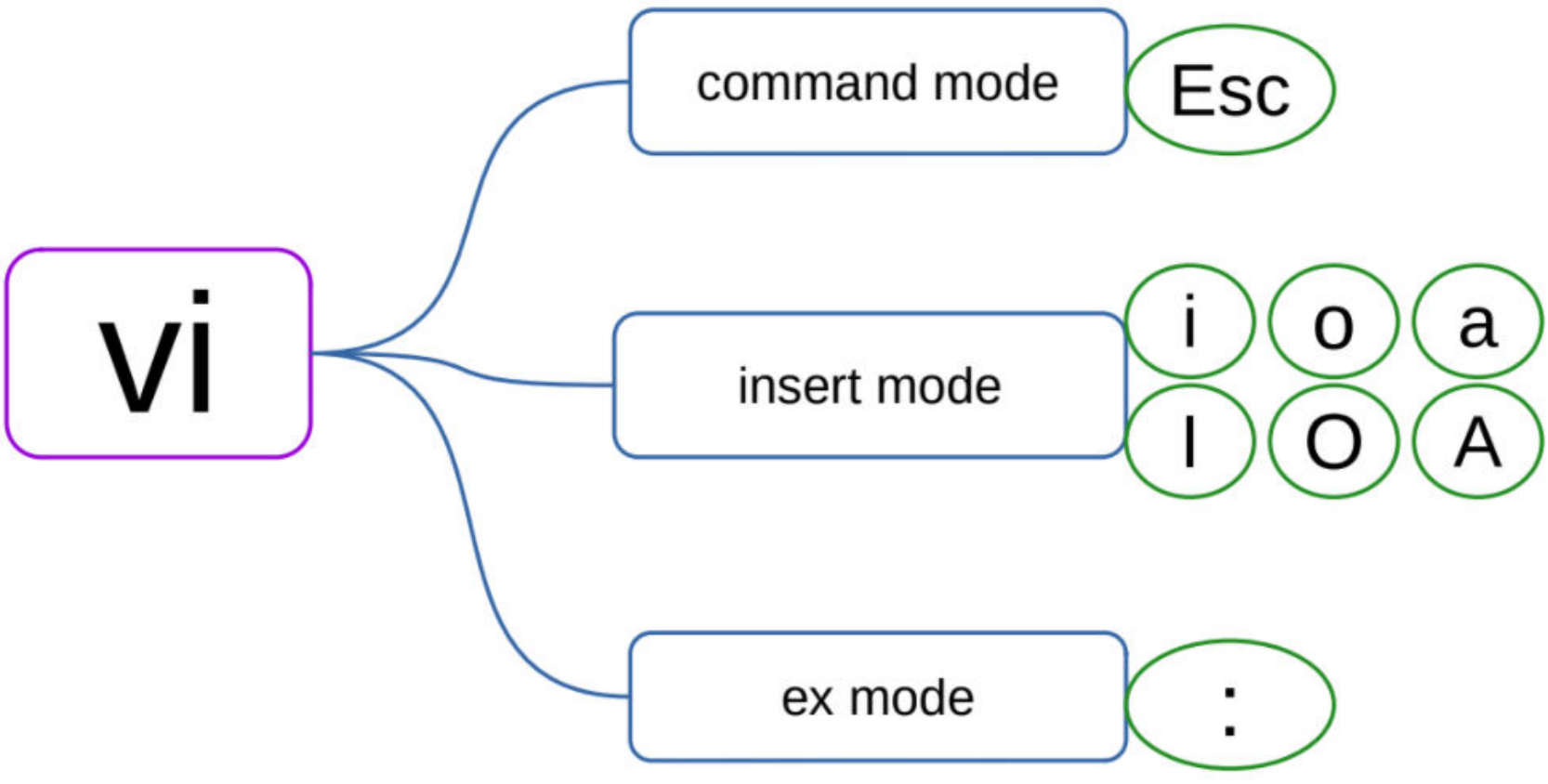
Ctrl+C	Report cursor position
Alt+D	Report line/word/character count
Ctrl+G	Display help text

Various

Alt+A	Turn the mark on/off
Tab	Indent marked region
Shift+Tab	Unindent marked region
Alt+V	Enter next keystroke verbatim
Alt+N	Turn line numbers on/off
Alt+P	Turn visible whitespace on/off
Alt+X	Hide or unhide the help lines
Ctrl+L	Refresh the screen

Source: https://www.nano-editor.org/dist/latest/cheatsheet.html

Modes in `vi` editor



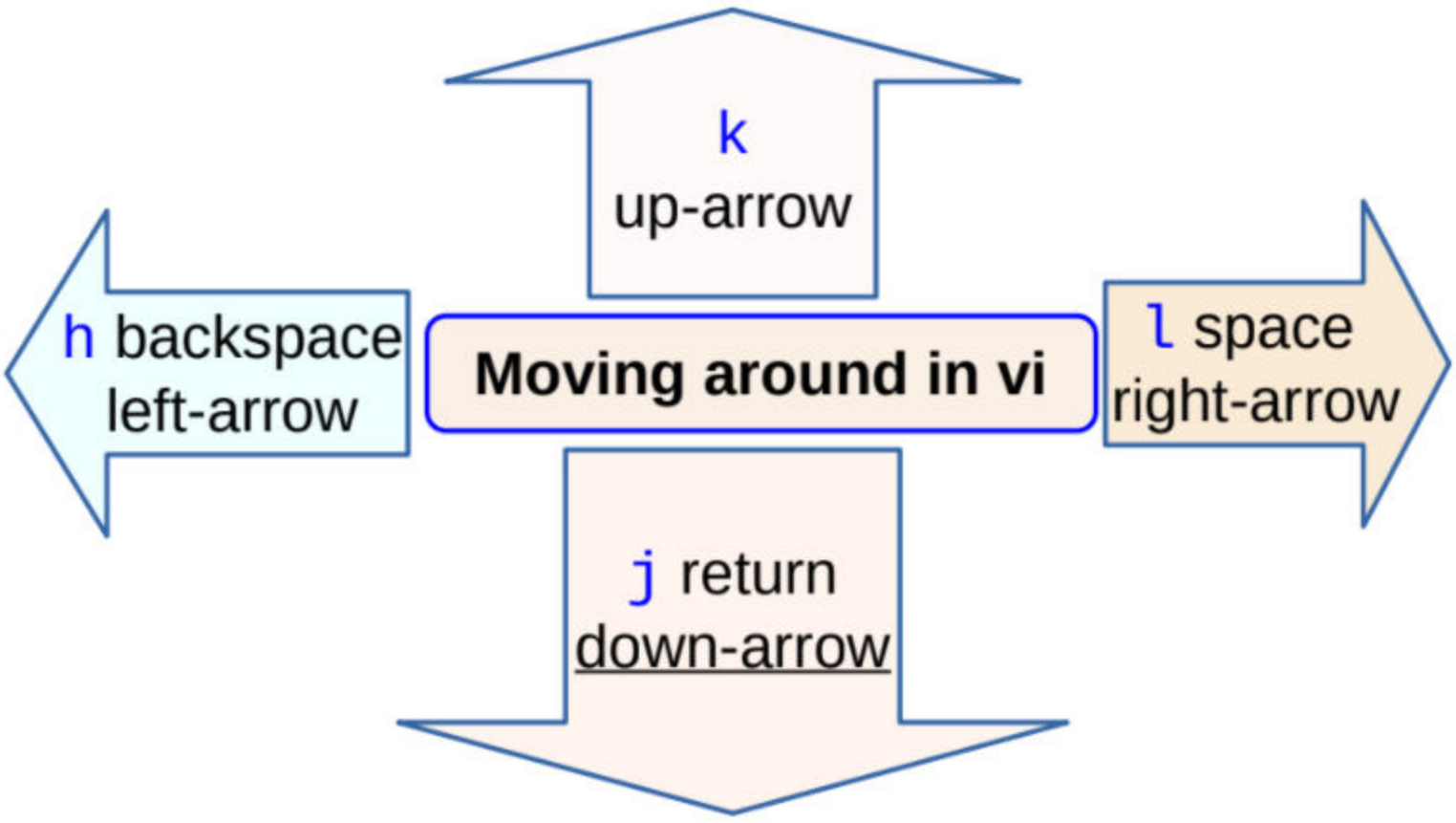
`i` will insert the characters from the current position of the cursor

`o` will insert a new line

`a` will append the text

`vi` help

- **These work only in the Command mode**
  - Press `Esc` to enter this mode
- To exit out of `vi`
  - `:w` → write out
  - `:x` → write out and quit
  - `:wq` → write out and quit
  - `:q` → quit (if write out is over)
  - `:q!` → ignore changes and quit
  - *If nothing else works, cry and smash your keyboard, and then rage quit*



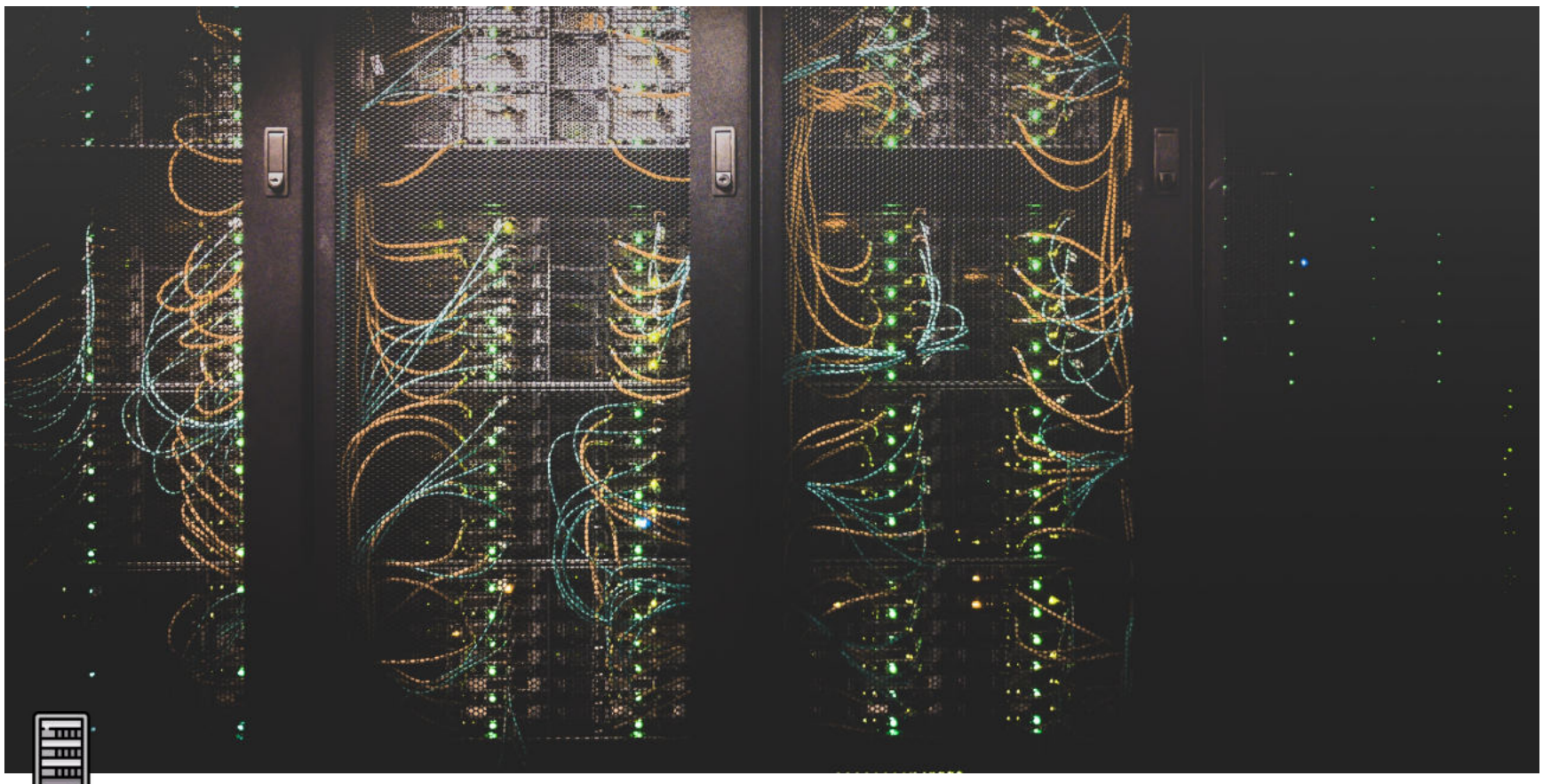
`vi` command mode

- Screen manipulation
  - `Ctrl + F` → Scroll forward one screen
  - `Ctrl + B` → Scroll backward one screen
  - `Ctrl + D` → Scroll down half screen
  - `Ctrl + U` → Scroll up half screen
  - `Ctrl + L` → Redraw screen
  - `Ctrl + R` → Redraw screen removing deleted stuff
- Moving around
  - `0` → Start of the current line
  - `$` → End of the current line
  - `w` → Beginning of the next word
  - `b` → Beginning of the preceding word
  - `:0` → First line in the file
  - `1G` → First line in the file
  - `:n` → n-th line in the file
  - `nG` → n-th line in the file
  - `:$` → Last line in the file
  - `G` → Last line in the file
- Changing text
  - `r` → Replace a single character under the cursor
  - `R` → Replace characters from the cursor till `Esc`
  - `cw` → Change word under the cursor, from the current character till `Esc`
  - `cNw` → Change **N** words, from the current character till `Esc`
  - `C` (CAPITAL C) → Change characters in the current line till `Esc`
  - `cc` (small c) → Change the line till `Esc`
  - `Ncc` → Change the next **N** lines, starting from the current till `Esc`
- Deleting text

    - `x` → Delete a single character under the cursor
    - `Nx` → Delete **N** characters from the cursor
    - `dw` → Delete one word, from the character under the cursor
    - `dNw` → Delete **N** words, from the character under the cursor
    - `D` → Delete the rest of the line, from the character under the cursor
    - `dd` → Delete the current line
    - `Ndd` → Delete the next N lines, starting from the current one
- Copy and Paste text
    - `yy` (small y) → Copy the current line to the buffer
    - `Nyy` → Copy the next N lines, including the current, into the buffer
    - `p` → Paste buffer into the file after the current line
    - `u` → Undo the previous action
- Searching text
    - `/string` → Search forward for the given `string`
    - `?string` → Search backward for the given `string`
    - `n` → Move the cursor to the next occurance of the `string`
    - `N` → Move the cursor to the previous occurrance of the `string`

`:se nu` → Set line numbers

`:se nonu` → Unset the line numbers



Command Line Editors - pt. 3

Type	📒 Lecture
Date	@February 2, 2022
Lecture #	3
Lecture URL	https://youtu.be/w25zlMXshHw
Notion URL	https://21f1003586.notion.site/Command-Line-Editors-pt-3-5452a0b89e32403184ce432b0bfd3b5a
Week #	5

To copy a file using secure copy `scp`

Syntax → `scp <username>@<IP_ADDRESS>:<path/to/file/on/that/machine> <where/to/save>`

Example → `scp gphani@10.17.0.167:Documents/code3d.tar .`

To untar (extract) a `.tar` file

Syntax → `tar -xvf filename.tar`

`emacs`

C-x means `Ctrl + x`

M-x means `Alt + x`

- Moving around
  - `C-p` → Move up by one line
  - `C-b` → Move left by one character
  - `C-f` → Move right by one character
  - `C-n` → Move down by one line
  - `C-a` → Go to the beginning of the current line
  - `C-e` → Go to the end of the current line
  - `C-v` → Move forward one screen
  - `M-<` → Move to the first line of the file
  - `M-b` → Move left to the previous word
  - `M-f` → Move right to the next word
  - `M->` → Move to the last line of the file

- `M-a` → Move to the beginning of the current sentence
- `M-e` → Move to the end of the current sentence
- `M-v` → Move back one screen

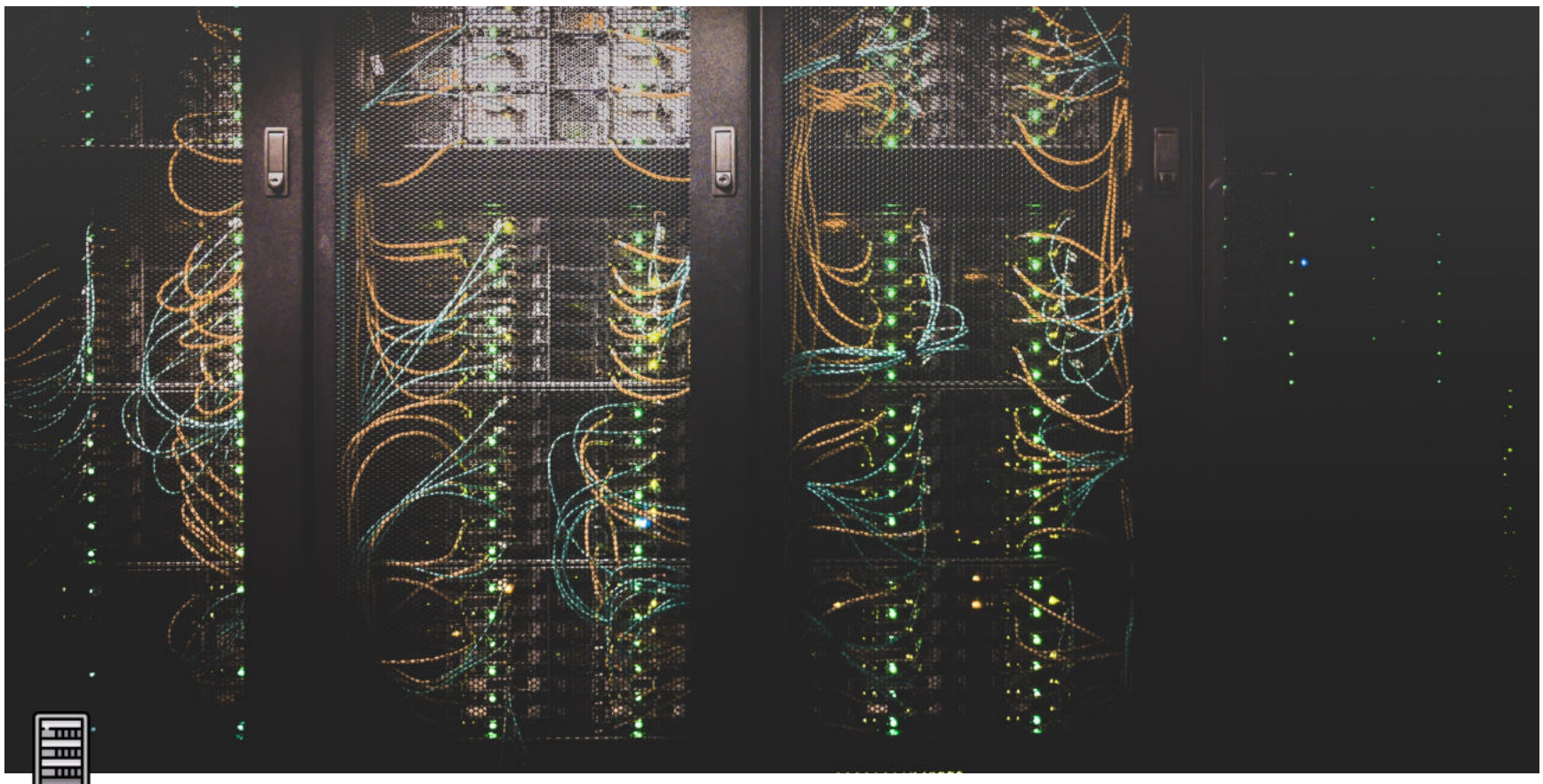
Source: https://www.gnu.org/software/emacs/refcards/pdf/refcard.pdf

`emacs` commands

- Exiting `emacs`
  - `C-x C-s` → Save buffer to the file
  - `C-z` → Exit emacs but keep it running
  - `C-x C-c` → Exit emacs and stop it
- Searching a text
  - `C-s` → Search forward
  - `C-r` → Search backward
  - `M-x` → Replace string
- Copy and Paste
  - `M-backspace` → Cut the word before the cursor
  - `M-d` → Cut the word after the cursor
  - `C-k` → Cut from the cursor to the end of the line
  - `M-k` → Cut from the cursor to the end of the sentence
  - `C-y` → Paste the content at the cursor

Syntax for emacs → `emacs -nw filename`

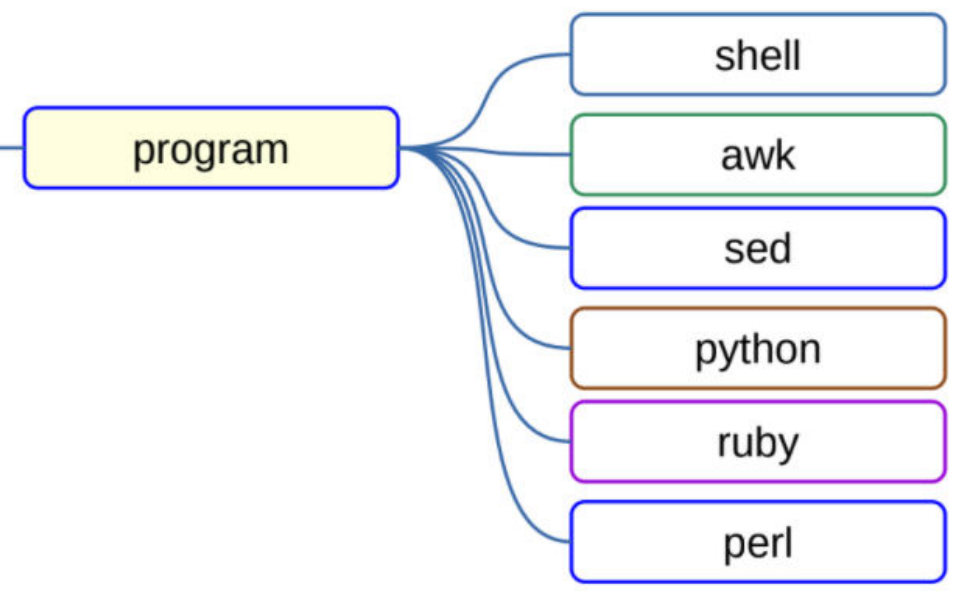
Make sure to pass the `-nw` flag to make it open in terminal mode instead of the GUI

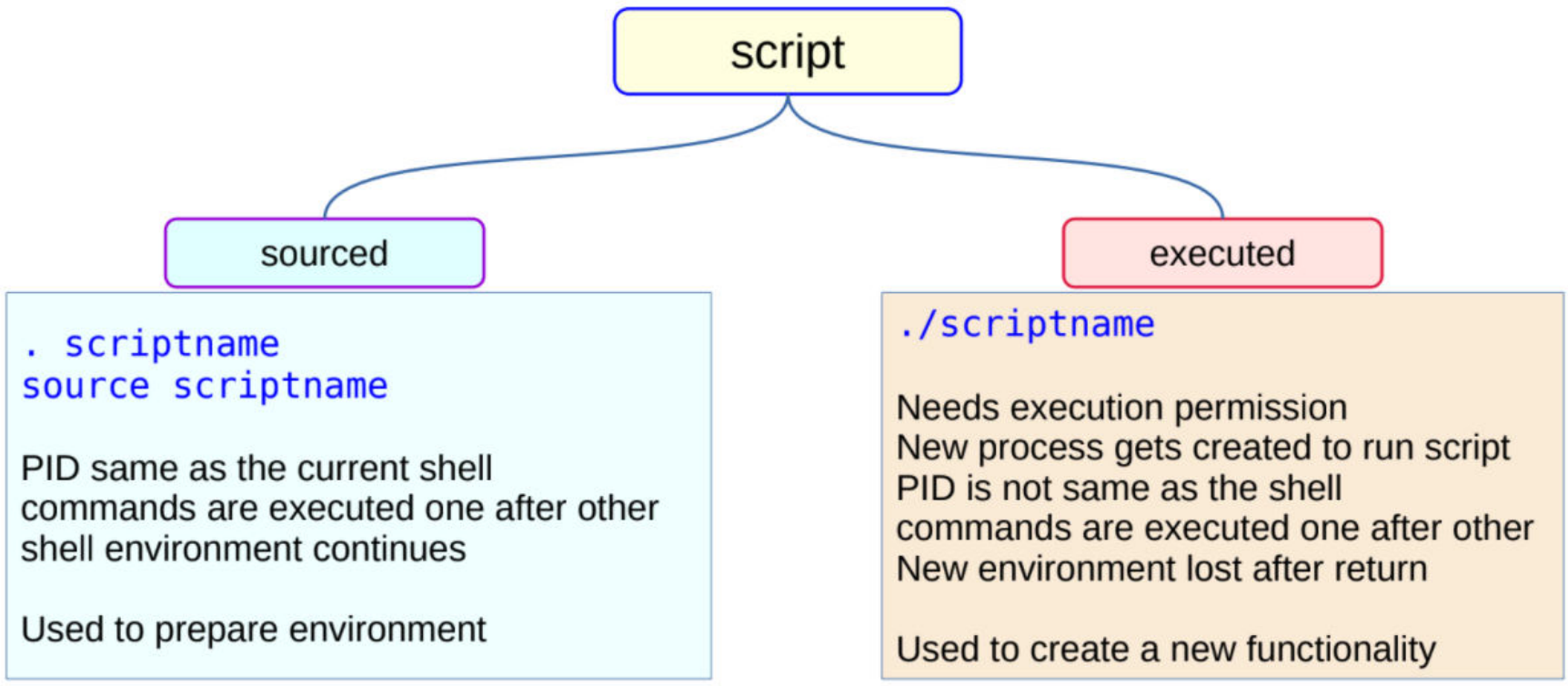


Overview of shell scripts

Type	📒 Lecture
Date	@February 2, 2022
Lecture #	4
Lecture URL	https://youtu.be/YAddHeVpG7I
Notion URL	https://21f1003586.notion.site/Overview-of-shell-scripts-2d85d6d2542f4122b49db652e3913de8
Week #	5

```
#! interpreter
# comments
commands
loops
variables
case statements
functions
```



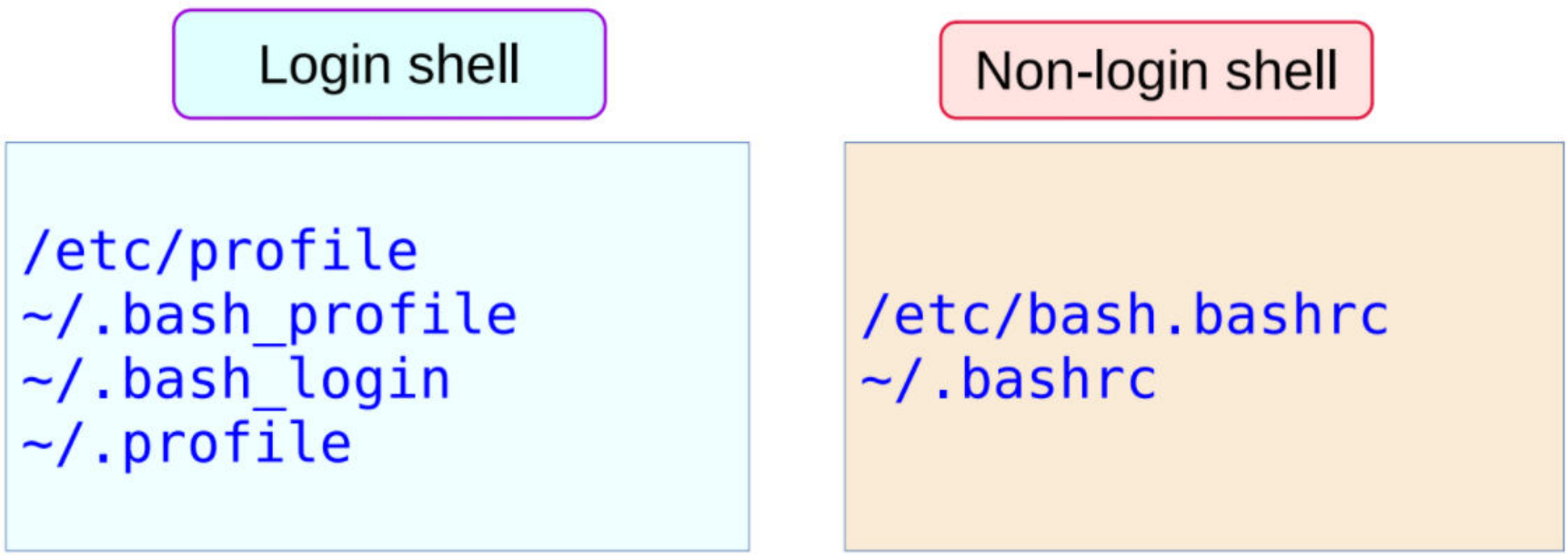


Script location

- Use absolute path or relative path while executing the script
- Keep the script in the folder listed in `$PATH`
- Watch out for the sequence of directories in `$PATH`

bash environment

- When you are already logged in to a system and using the GNOME environment to open a terminal, then the shell we are opening is not asking for a login and is called a **Non-login shell**
- When we ssh into a remote machine, then the shell we are logging into would have checked for the login credentials and then opened up the command line environment for us, therefore that shell is called a **Login shell**



Output from the shell scripts

- `echo`
  - Terminates with a newline if `-n` flag is not given
  - `echo My home is $HOME`
- `printf`
  - Supports format specifiers like in C
  - `printf “My home is %s\n” $HOME`

Input to shell scripts

- `read var`
  - String read from command line is stored in `$var`

Shell script arguments

`./myscript.sh -l arg2 -v arg4`

- `$0` → name of the shell program
- `$#` → number of arguments passed
- `$1` or `${1}` → first argument
- `${11}` → eleventh argument
- `$*` or `$@` → all arguments at once
- `"$*"` → all arguments as a single string
- `"$@"` → all arguments as separate strings

Command substitution

```
var=`command`
var=$(command)
```

`command` is executed and the output is substituted

Here, the variable `var` will be assigned with that output

`for do` loop

```
for var in list
do
  commands
done
```

`commands` are executed once for each item in the `list`

space is the field delimiters

set IFS if required

`case` statement

```
case var in
pattern1)
  commands
  ;;
pattern2)
  commands
  ;;
esac
```

`commands` are executed

each pattern matched

for var in the options

`if` loop

```
if condition
then
  commands
fi
```

```
if condition; then
  commands
fi
```

`commands` are executed only if `condition` returns true

`conditions` in `if`

- `test -e file` or any `text expression`

    - True, if the file exists in the current directory of the shell script
    - else False
- `[ -e file ]` or any `[ expression ]`
    - Same as the above one, just a different syntax
- `[[ expression ]]` like `[[ $ver == 5.* ]]`
    - If we intend to use regex or other complex operations
- `(( expression ))` like `(( $v ** 2 > 10 ))`
    - To perform complex arithmetic operations for the test conditions
- Or just use a `command` like `wc -l file`
    - If the command returns True, i.e. upon successful execution, this means the condition is satisfied
    - else False
- `pipeline` like `who | grep "joy" > /dev/null`
    - A set of commands which can be combined with other commands and redirection of output and combination of logical expressions using the `&&` or `||`
    - These can be put as the condition

For negation, use `!` before the condition

test numeric comparisons

- `$n1 -eq $n2`
    - Check if n1 is equal to n2
- `$n1 -ge $n2`
    - Check if n1 is greater than or equal to n2
- `$n1 -gt $n2`
    - Check if n1 is greater than n2
- `$n1 -le $n2`
    - Check if n1 is less than or equal to n2
- `$n1 -lt $n2`
    - Check if n1 is less than n2
- `$n1 -ne $n2`
    - Check if n1 is not equal to n2

test string comparisons

- `$str1 = $str2`
    - Check if str1 is the same as str2
- `$str1 != $str2`
    - Check if str1 is not the same as str2
- `$str1 < $str2`
    - Check if str1 is less than str2
    - Compares based on lexicographical (alphabetical) order
- `$str1 > $str2`
    - Check if str1 is greater than str2
    - Compares based on lexicographical (alphabetical) order
- `-n $str1`
    - Check if str1 has length greater than zero

- `-z $str1`
  - Check if str1 has the length zero

Unary file comparisons

- `-e file`
  - Check if the file exists
- `-d file`
  - Check if the file exists and is a directory
- `-f file`
  - Check if the file exists and is a file
- `-r file`
  - Check if the file exists and is readable
- `-s file`
  - Check if the file exists and is not empty
- `-w file`
  - Check if the file exists and is writable
- `-x file`
  - Check if the file exists and is executable
- `-O file`
  - Check if the file exists and is owned by the current user
- `-G file`
  - Check if the file exists and the default group is the same as that of the current user

Binary file comparisons

- `file1 -nt file2`
  - Check if file1 is newer than file2
- `file1 -ot file2`
  - Check if file1 is older than file2

`while do` loop

```
while condition
do
  commands
done
```

`commands` are executed only if the `condition` returns `true`

`until do` loop

```
until condition
do
  commands
done
```

`commands` are executed only if the `condition` returns `false`

functions

definition

```
myfunc()
{
```

```
  commands
}
```

call

```
myfunc
```

`commands` are executed each time `myfunc` is called

Definitions must be before the calls