



# Linux

A FREE, STABLE, AND SECURE OPERATING SYSTEM (OS).



**FSWE**

**Lesson 1**



**Created by Tien Nguyen**  
[tiennk@fullstackdatascience.com](mailto:tiennk@fullstackdatascience.com)

# Learn About



**FSDS**  
fullstackdatascience

01/ **GNU/LINUX**

02/ **COMMAND LINE BASICS**

03/ **BASH SCRIPTING**

04/ **CRON JOB**

 [fullstackdatascience.com](http://fullstackdatascience.com)

# GNU/ Linux



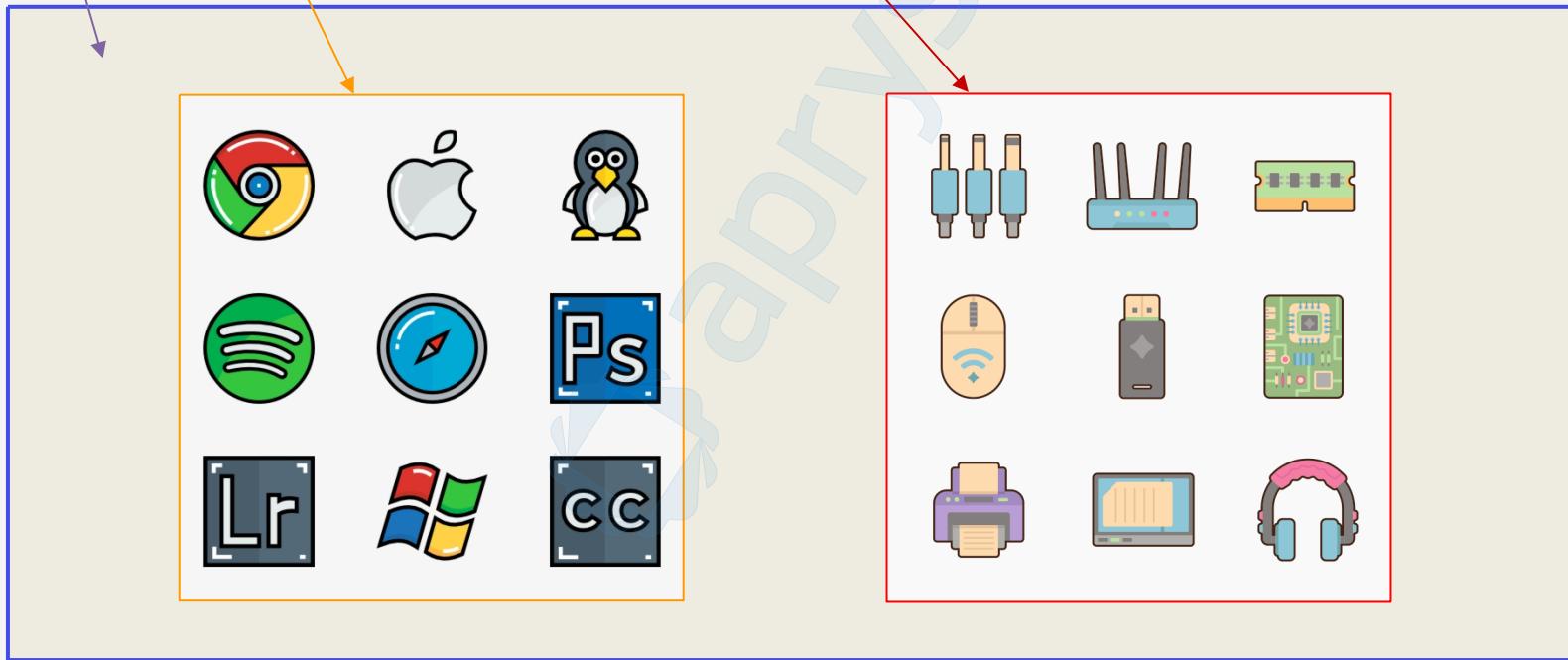
**FSDS**  
fullstackdatascience

## Introduction

- **Open-source operating system**
- **Free to use and modify**
- **Community-developed**
- **Secure and flexible**

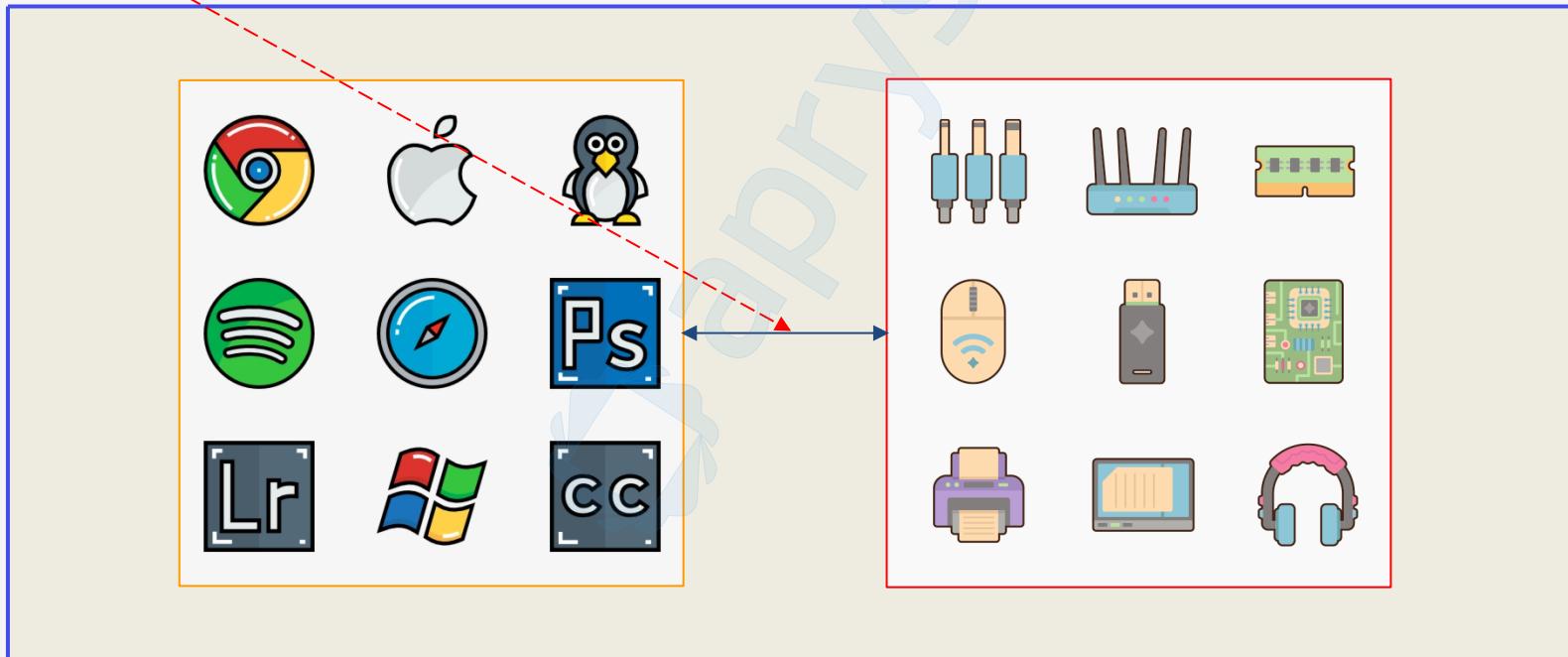
# Operating system (OS)

The **OS** is the **software** that manages all the **hardware** and **software** on your computer/phone



# Kernel

The **kernel** is the *core part* of the OS



Kernel is like the "**bridge**" between your apps and the hardware

# GNU

GNU: A free software project, provides most tools of a **Unix-like OS**

An old OS



**GNU project** made everything: the wheels, seats, steering, engine parts, etc

But they didn't finish the **engine** (the kernel)

**Linus Torvalds** came along and built a working **engine (Linux kernel)**

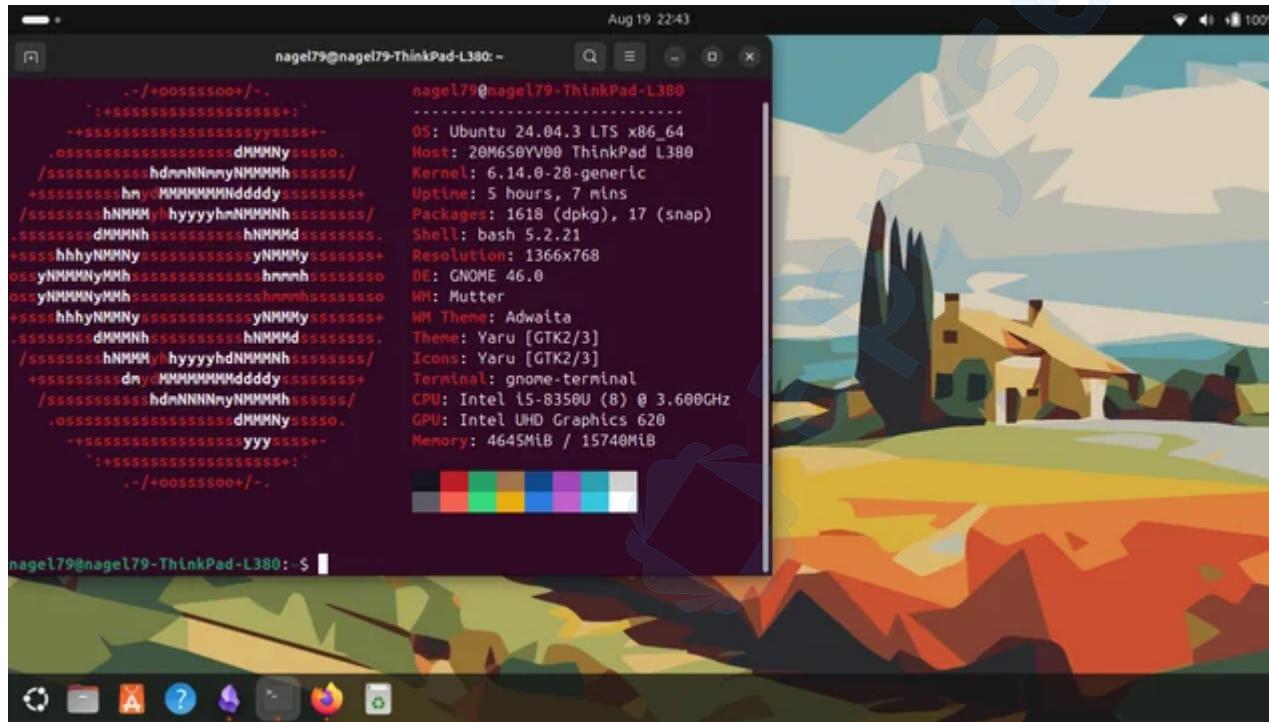
Together, **GNU + Linux kernel = a complete working car (GNU/Linux OS)**

# GNU/Linux distros



These are some common Linux distros

# **Linux Distro: Ubuntu**



We will use this distro in this course

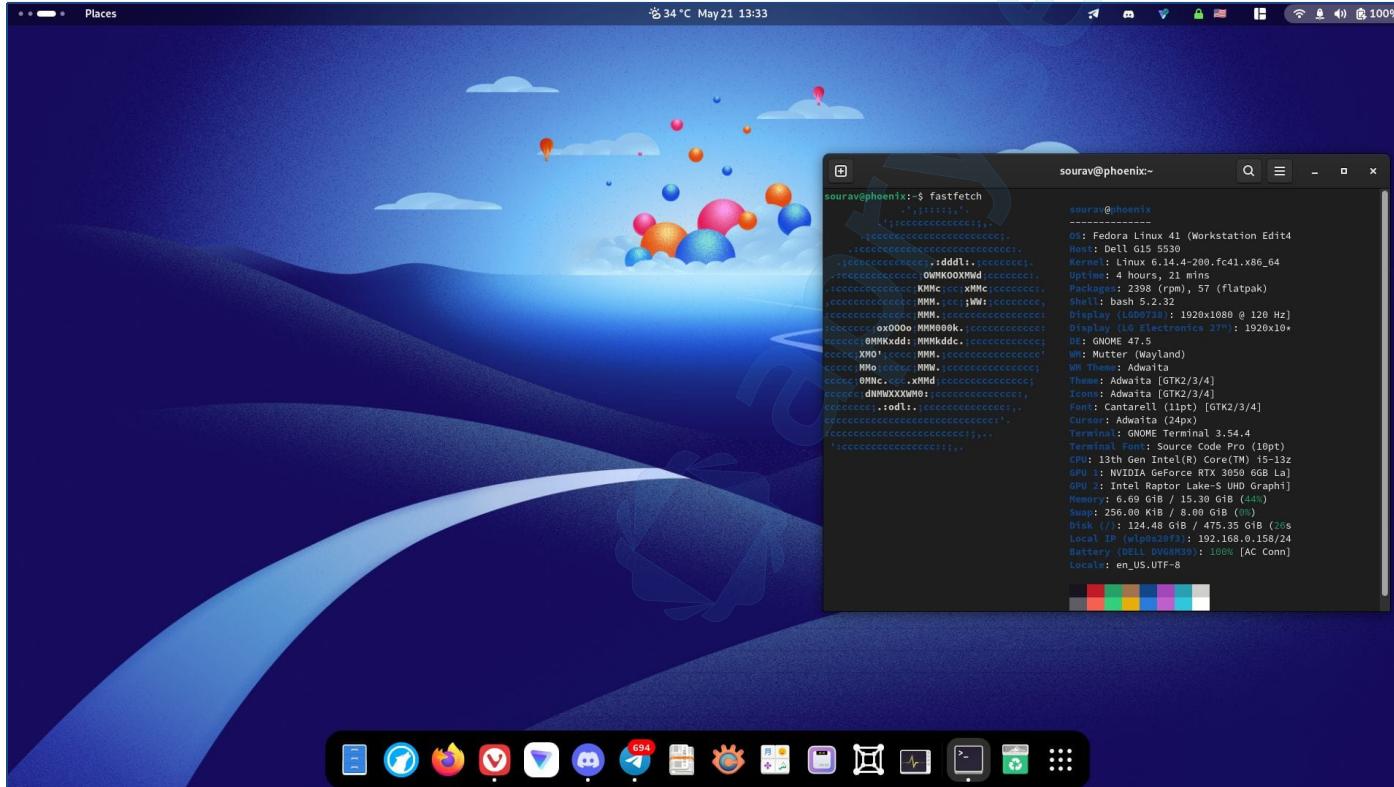
Ubuntu is the easiest for new users

Large community + lots of tutorials

Stable, secure, and user-friendly

Great choice for people  
new to Linux

# Linux Distro: Fedora



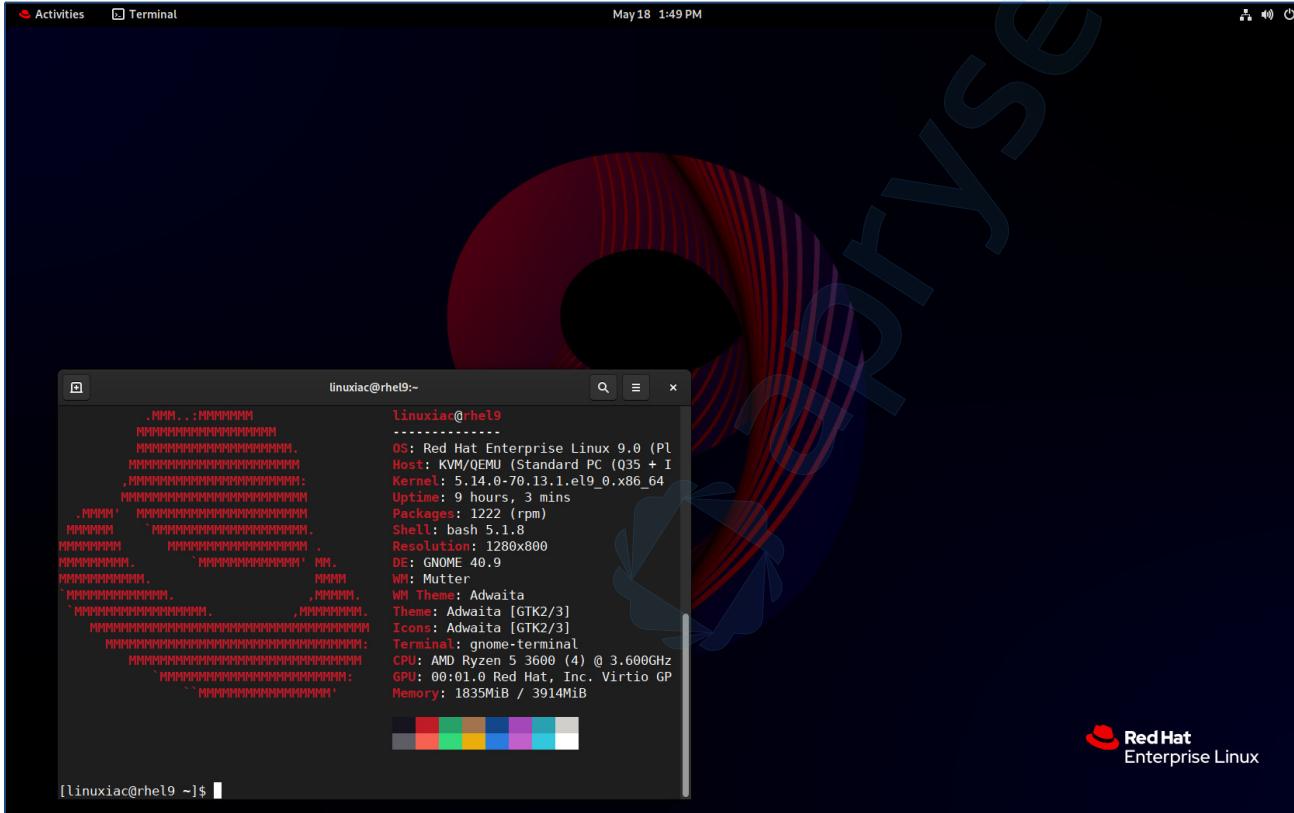
Sponsored by  
Red Hat

Cutting-edge  
software,  
frequent updates

Default GNOME  
desktop  
environment

Used as a  
testbed for Red  
Hat Enterprise  
Linux (RHEL)

# Linux Distro: Red Hat Linux



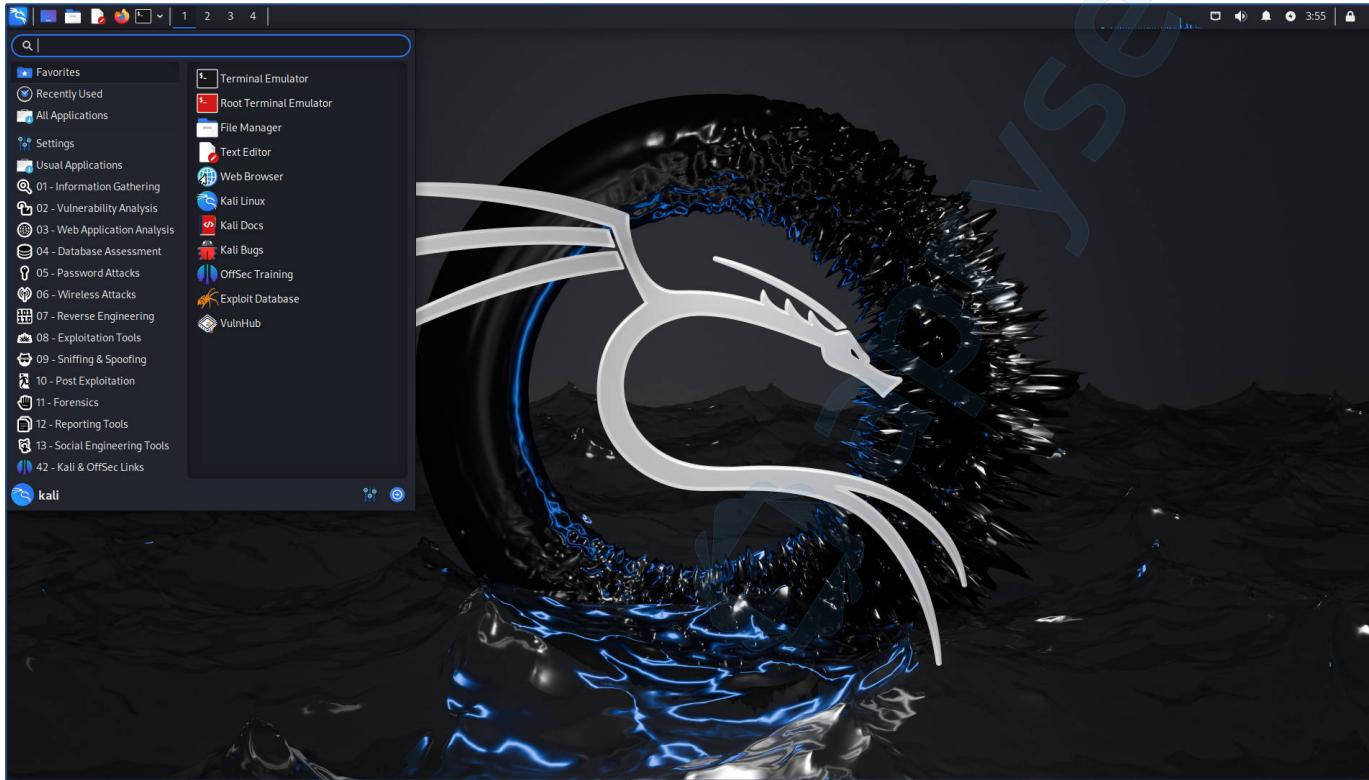
Commercial Linux distribution

Widely used in enterprises and servers

Strong support and stability

Subscription-based model

# Linux Distro: Kali Linux



Based on Debian

Designed for **penetration testing, ethical hacking, and cybersecurity**

Comes with hundreds of pre-installed security tools

Used by security professionals and researchers

Not recommended for beginners or daily desktop use

# Linux Distro: Linux Mint



Very beginner-friendly  
(similar to Windows UI)

Comes with multimedia  
codecs pre-installed

Focus on ease of use  
and stability

Popular among users  
switching from Windows

# Linux Distro: Arch Linux



Arch Linux 5.8.12-arch1-1 (tty1)

archiso login: root (automatic login)

To install [Arch Linux](#) follow the installation guide:

[https://wiki.archlinux.org/index.php/Installation\\_guide](https://wiki.archlinux.org/index.php/Installation_guide)

For Wi-Fi, authenticate to the wireless network using the `iwctl` utility.  
Ethernet and Wi-Fi connections using DHCP should work automatically.

After connecting to the internet, the installation guide can be accessed via the convenience script [Installation\\_guide](#).

root@archiso ~ # \_



You can turn from this dark screen to this crazy UI if you have enough skills!



Lightweight

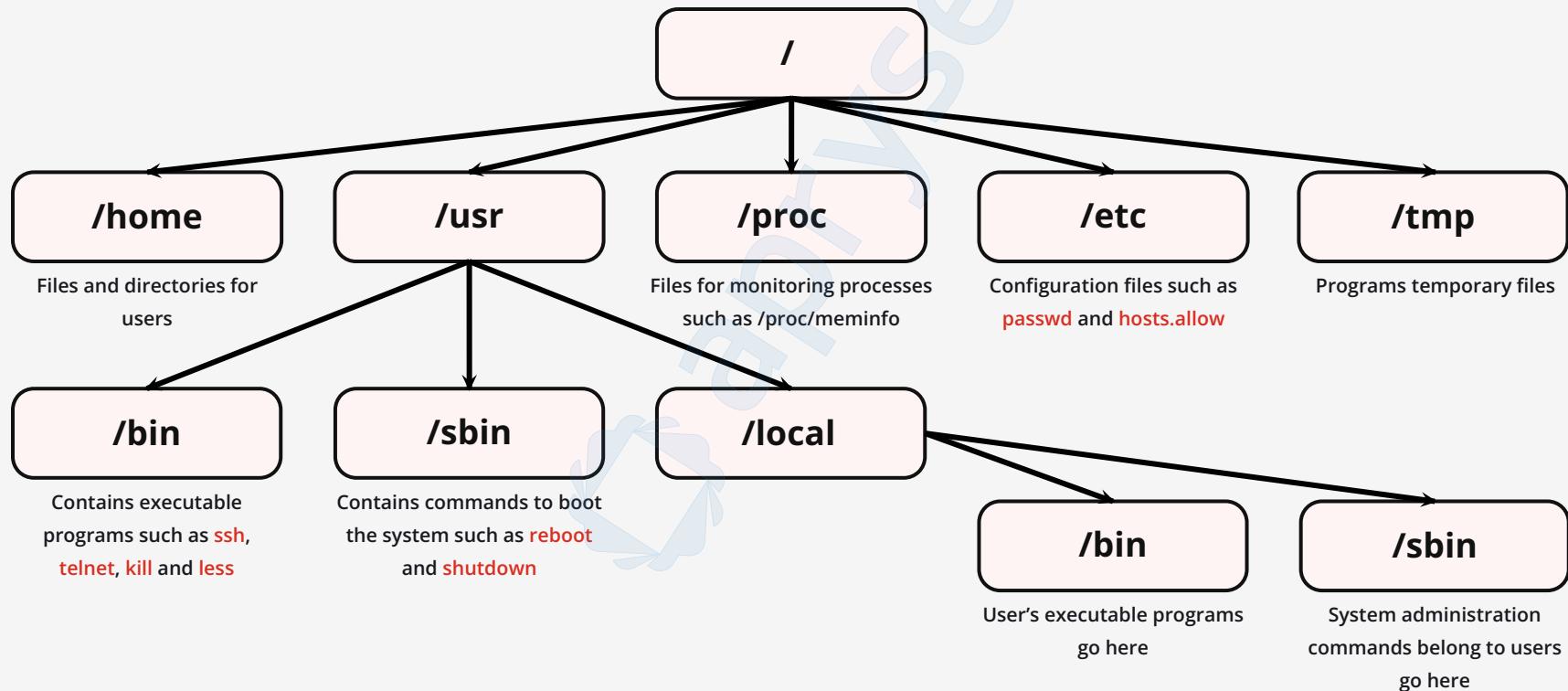
"Do-it-yourself" philosophy → minimal by default

Requires manual setup and configuration

Excellent documentation (Arch Wiki)

Suited for advanced users

# Linux Directory Structure



# TL;DR- GNU/Linux



**GNU** = Free software project, provides most tools of a Unix-like OS.

**Linux** = A kernel that, together with GNU, forms a full OS.

That's why many people say "**GNU/Linux**" instead of just "Linux".

# Command Line Basics

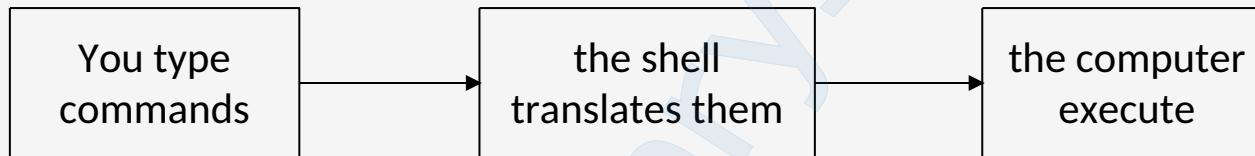
**Typing commands differs from navigating a graphical user interface (GUI) !**



**FSDS**  
fullstackdatascience

# Shell

The **shell is the translator** between you and the computer



**BASH**  
THE BOURNE-AGAIN SHELL

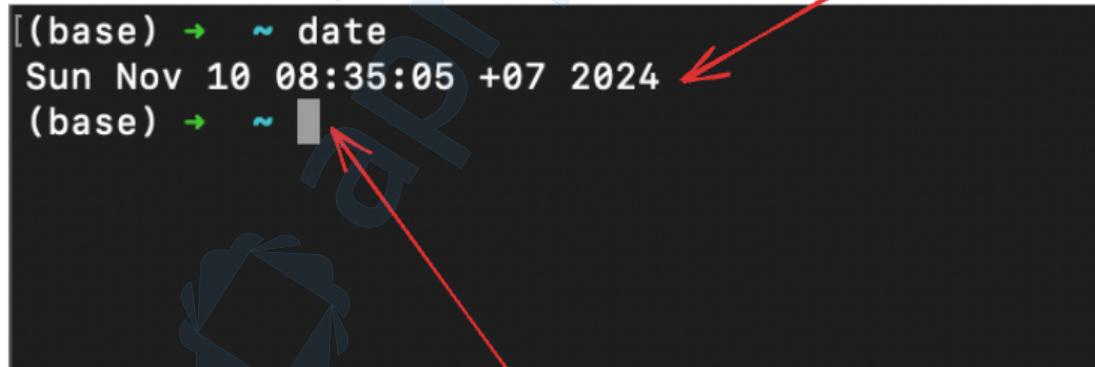


**Bash** is the default shell for most Linux distributions.

# What is Command Line?

A **text-based interface** to interact directly with the computer's operating system

You type commands and get text output instead of clicking buttons, using menus, and dragging things around



The diagram illustrates a terminal window with the following components labeled:

- command line prompt**: The text "(base) ~" followed by a green arrow pointing right.
- command to run**: The command "date" entered at the prompt.
- Computer response**: The output of the "date" command, showing the date and time: "Sun Nov 10 08:35:05 +07 2024".
- newline (can run new command here)**: A red arrow points to the bottom of the terminal window, indicating where a new command can be entered.

# Basic Commands



Functions	Popular commands
Navigate the file system	cd, pwd, ls
Manipulate files	cp, touch, mkdir, mv, rm
View files	cat, head, tail
User manual of commands	man
Search	find . -name “*.log” -size +1G
Others	history, clear, shutdown, reboot, uptime

## Fun fact:

A directory is just a file containing the names of other files (refer to [here](#))

# Navigate the file system



**ls (list):** Lists files and directories in the current directory (or a specified one)

```
magnusdtd@magnusdtd-VivoBook ~/D/test> ls  
folder1/ folder2/  
magnusdtd@magnusdtd-VivoBook ~/D/test> █
```

Use **ls** to list files inside in the current directory

Use **ls** to list files inside the **folder1** directory

```
magnusdtd@magnusdtd-VivoBook ~/D/test> ls folder1  
file.txt folder1.1/ folder1.2/  
magnusdtd@magnusdtd-VivoBook ~/D/test> █
```

# Navigate the file system



**cd (change directory):** Moves you from your current working directory to another one

```
magnusdtd@magnusdtd-VivoBook ~/D/test> ls
folder1/ folder2/
magnusdtd@magnusdtd-VivoBook ~/D/test> cd folder1/
magnusdtd@magnusdtd-VivoBook ~/D/t/folder1> ls
folder1.1/ folder1.2/
magnusdtd@magnusdtd-VivoBook ~/D/t/folder1> █
```

The **test** has 2 folder: **folder1** and **folder2**.  
The **folder1** has 2 subfolder: **folder1.1** and **folder1.2**

# Navigate the file system

**pwd (print working directory):** Shows the **full path of the directory** you're currently in

```
magnusdtd@magnusdtd-VivoBook ~/D/t/folder1> pwd
/home/magnusdtd/Downloads/test/folder1
magnusdtd@magnusdtd-VivoBook ~/D/t/folder1>
```

# Manipulate files

**cp (copy):** Makes a copy of a **file**

New file name

```
magnusdtd@magnusdtd-VivoBook ~/D/test> cp folder1/file.txt folder2/backup.txt
magnusdtd@magnusdtd-VivoBook ~/D/test> ls folder2
backup.txt
magnusdtd@magnusdtd-VivoBook ~/D/test>
```

Copy the **file.txt** from the **folder1** to the **folder2**

# Manipulate files

**cp (copy):** Makes a copy of a **folder**

New folder name

```
magnusdtd@magnusdtd-VivoBook ~/D/test> ls folder2  
backup.txt  
magnusdtd@magnusdtd-VivoBook ~/D/test> cp -r folder2 folder1/folder1.3  
magnusdtd@magnusdtd-VivoBook ~/D/test> ls folder1/folder1.3/  
backup.txt  
magnusdtd@magnusdtd-VivoBook ~/D/test> █
```

Copy all the content of the **folder2** to the new destination **folder1/folder1.3**

# Manipulate files

**touch**: Creates a new empty file

if it doesn't exist

```
magnusdtd@magnusdtd-VivoBook ~/D/test> ls folder1
file.txt  folder1.1/  folder1.2/  folder1.3/
magnusdtd@magnusdtd-VivoBook ~/D/test> touch folder1/newfile.txt
magnusdtd@magnusdtd-VivoBook ~/D/test> ls folder1
file.txt  folder1.1/  folder1.2/  folder1.3/  newfile.txt
magnusdtd@magnusdtd-VivoBook ~/D/test>
```

Create a new file named **newfile.txt**

# Manipulate files

**mkdir (make directory)**: Creates a new folder

```
magnusdtd@magnusdtd-VivoBook ~/D/test> ls
folder1/ folder2/
magnusdtd@magnusdtd-VivoBook ~/D/test> mkdir folder3
magnusdtd@magnusdtd-VivoBook ~/D/test> ls
folder1/ folder2/ folder3/
magnusdtd@magnusdtd-VivoBook ~/D/test>
```

Create a new folder named **folder3**

# Manipulate files

**mv (move)**: Moves files/directories

```
magnusdtd@magnusdtd-VivoBook ~/D/test> ls folder1  
file.txt folder1.1/ folder1.2/ folder1.3/ newfile.txt  
magnusdtd@magnusdtd-VivoBook ~/D/test> mv folder1/newfile.txt folder1/file2.txt  
magnusdtd@magnusdtd-VivoBook ~/D/test> ls folder1  
file2.txt file.txt folder1.1/ folder1.2/ folder1.3/  
magnusdtd@magnusdtd-VivoBook ~/D/test>
```

Create a new folder named **folder3**

# Manipulate files

**mv (move)**: Renames files/directories

```
magnusdtd@magnusdtd-VivoBook ~/D/test> ls
folder1/ folder2/ folder3/
magnusdtd@magnusdtd-VivoBook ~/D/test> mv folder3 folder4
magnusdtd@magnusdtd-VivoBook ~/D/test> ls
folder1/ folder2/ folder4/
magnusdtd@magnusdtd-VivoBook ~/D/test>
```

Rename from **folder3** to **folder4**

# Manipulate files

**rm (remove)**: Deletes files (or directories with -r)

```
magnusdtd@magnusdtd-VivoBook ~/D/test> ls folder1
file2.txt file.txt folder1.1/ folder1.2/ folder1.3/
magnusdtd@magnusdtd-VivoBook ~/D/test> rm folder1/file2.txt
magnusdtd@magnusdtd-VivoBook ~/D/test> ls folder1
file.txt folder1.1/ folder1.2/ folder1.3/
magnusdtd@magnusdtd-VivoBook ~/D/test>
```

Delete file2.txt

Delete folder4

```
magnusdtd@magnusdtd-VivoBook ~/D/test> ls
folder1/ folder2/ folder4/
magnusdtd@magnusdtd-VivoBook ~/D/test> rm -r folder4
magnusdtd@magnusdtd-VivoBook ~/D/test> ls
folder1/ folder2/
magnusdtd@magnusdtd-VivoBook ~/D/test>
```

# View files

**cat (concatenate)**: reads files and writes their contents to the terminal

```
magnusdtd@magnusdtd-VivoBook ~/D/test> cat folder1/file.txt
This is the first lesson of the FSWE course!
```

Show the content of the **file.txt**

Merge the contents of **file.txt** and **file2.txt** into **merge.txt**.

```
magnusdtd@magnusdtd-VivoBook ~/D/test> cat folder1/file.txt folder1/file2.txt \
> folder1/merge.txt
magnusdtd@magnusdtd-VivoBook ~/D/test> cat folder1/merge.txt
This is the first lesson of the FSWE course!
Enjoy the lesson!
magnusdtd@magnusdtd-VivoBook ~/D/test>
```

# View files

**head (head of the file):** prints the first part of a file

Show the content of the file.txt

```
magnusdtd@magnusdtd-VivoBook ~/D/test> head -n 20 folder1/file3.txt
Every day is harder.
Knowing you are but
a mile from where I lay.

In my mind your name
is every other word.

My love for you burdens me
with more time it seems.

You are imprisoned behind your
walls from me but I know
the love that burns beneath
your lovely breast.

We have eight months to pass,
before we can indulge.

I miss your lips of roses,
```

default: 10 lines

shows the first 20 lines

# View files

**tail (tail of the file): prints the first part of a file**

Show the content of the **file.txt**

```
magnusdtd@magnusdtd-VivoBook ~/D/test> tail -n 20 folder1/file3.txt
I'm an over powered anxiety complex
in a box as big as a dime.

My love for you will never cease,
Death is the only ending for this.

Come to me because
I am yet a mere feather
You are my match my lover

My sweet and beautiful Heather.
magnusdtd@magnusdtd-VivoBook ~/D/test>
```

*default: 10 lines*

# Others



**man (manual)**: Displays the documentation for a command.

```
magnusdtd@magnusdtd-VivoBook ~/D/test> [man ls]
magnusdtd@magnusdtd-VivoBook ~/D/test>
```

```
LS(1)                               User Commands                               LS(1)

NAME
    ls - list directory contents

SYNOPSIS
    ls [OPTION]... [FILE]...

DESCRIPTION
    List information about the FILEs (the current directory by default).
    Sort entries alphabetically if none of -cftuvSUX nor --sort is specified.

    Mandatory arguments to long options are mandatory for short options too.

    -a, --all
        do not ignore entries starting with .

    -A, --almost-all
        do not list implied . and ..

    --author
        with -l, print the author of each file

Manual page ls(1) line 1 (press h for help or q to quit)
```

Show the documentation for the **ls** command

# Others

**clear**: Clears the terminal screen

```
magnusdtd@magnusdtd-VivoBook ~/D/test> ls ./folder1/  
file2.txt file3.txt file.txt folder1.1/ folder1.2/ folder1.3/ merge.txt  
magnusdtd@magnusdtd-VivoBook ~/D/test> ls  
folder1/ folder2/  
magnusdtd@magnusdtd-VivoBook ~/D/test> ls folder2/  
backup.txt  
magnusdtd@magnusdtd-VivoBook ~/D/test> clear
```

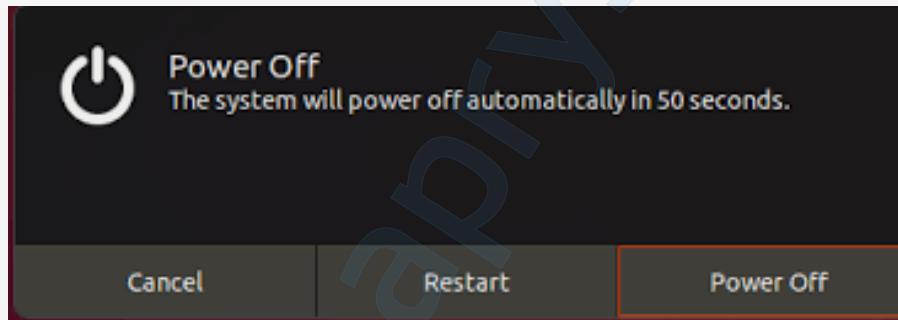
```
magnusdtd@magnusdtd-VivoBook ~/D/test>
```



The screen has been cleaned up after using the **clear** command

# Others

**shutdown/reboot:** Powers off/restarts your computer



! DO NOT TRY THESE COMMANDS RIGHT NOW !



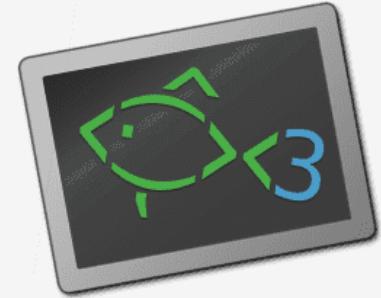
# Fish shell

Wow, that's a lot of stuff. I can't remember all of them ... You can try Fill Shell

Autocomplete feature



A screenshot of a terminal window titled "fish". The window shows the welcome message: "Welcome to fish, the friendly interactive shell. Type help for instructions on how to use fish". Below the message, the command "magnusdtd@magnusdtd-VivoBook ~> cd /D" is typed, followed by a cursor and a blue dashed box highlighting the path separator "/D". A blue dashed arrow points from the text "Autocomplete feature" to this highlighted area.



**Warning:** Some commands in Fish may differ the Bash (Linux default shell)

# Interesting commands

Replace all **unix** words with **linux**

```
sed 's/unix/linux/g' geekfile.txt
```

List all containers' names containing **minio**, skipping the row number 0, and stop them

```
docker ps -a | grep "minio" | awk "NR>1 {print $1}" | xargs docker stop
```

Find large files or folders

```
find / -type f -size +1G OR find / -type d -size +1G
```

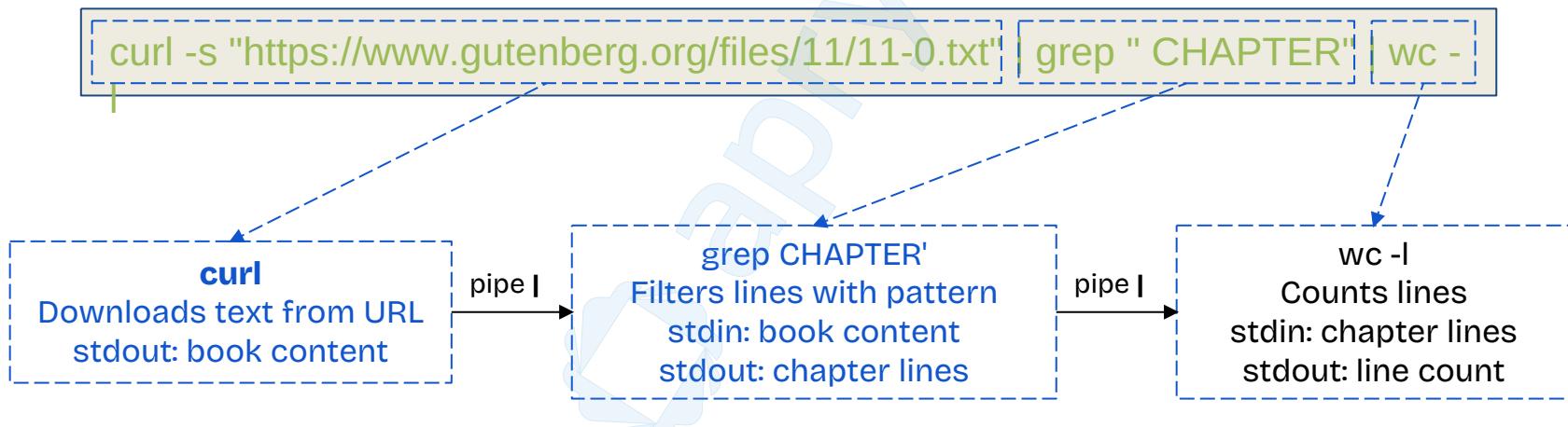
Summary disk usage with

```
du -sh *
```

# Combining command-line tools

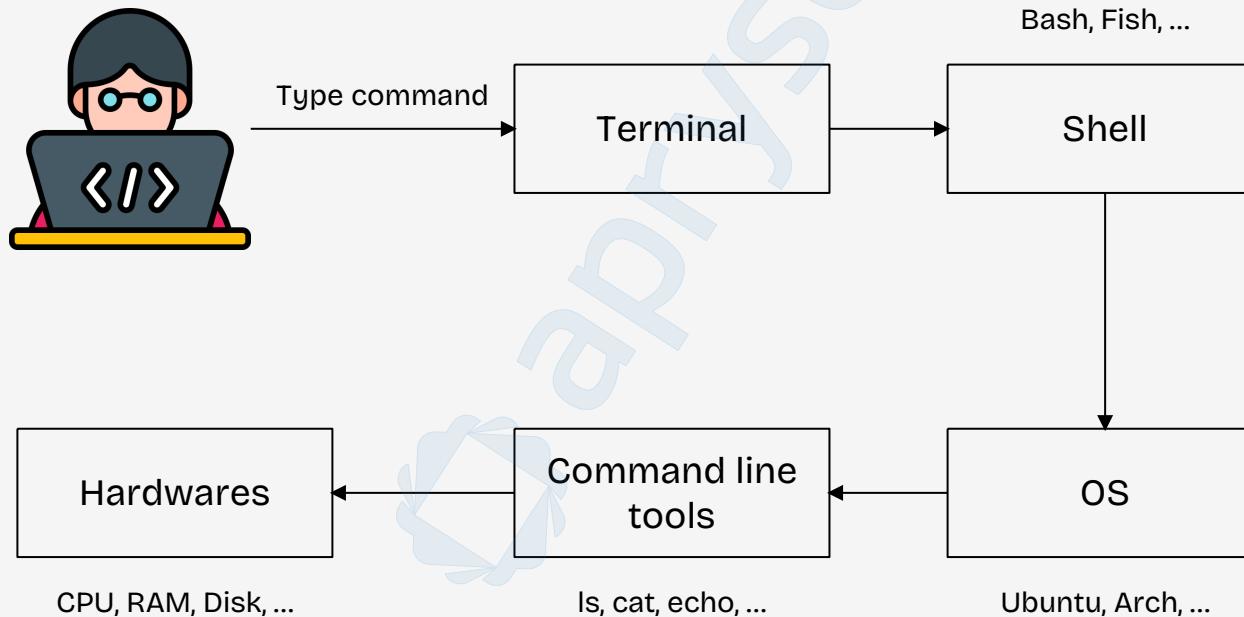


Command-line tools are designed to do one thing and do it well. We can combine these tools to create powerful workflows by chaining them together using **pipes** and **redirects**



The command prints the **number of chapters** in *Alice's Adventures in Wonderland*

# Command line environment



# Command line is useful



**Fast & Interactive**



Instant command execution  
— type, hit Enter, go!

**Seamless Integration**



Connects tools like Python, R, Spark  
—outputs flow as inputs

# Command line is useful

**Automated**



Scriptable for recurring tasks—  
runs on servers anytime

**Ubiquitous**



CLI tools present on all platforms—vital  
for cloud and servers.

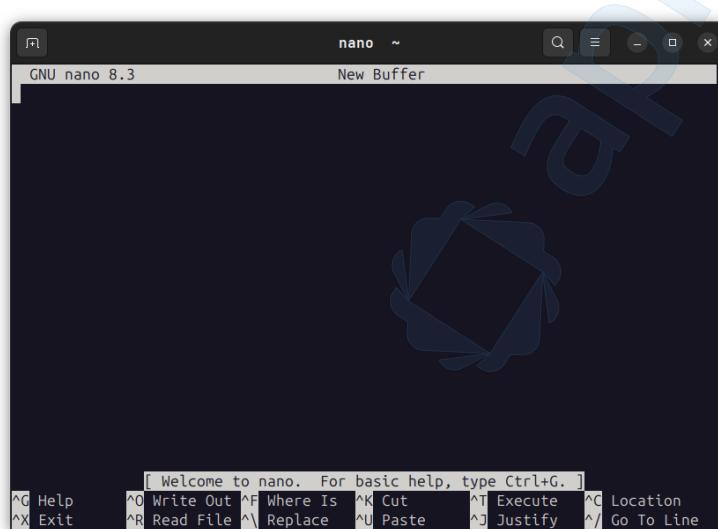
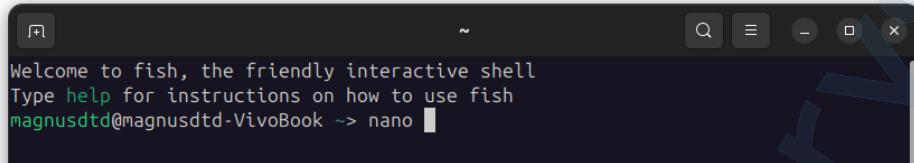
# Bash Scripting

**Transform commands into  
automation !**



# Nano

Nano is a simple text editor that runs inside the terminal



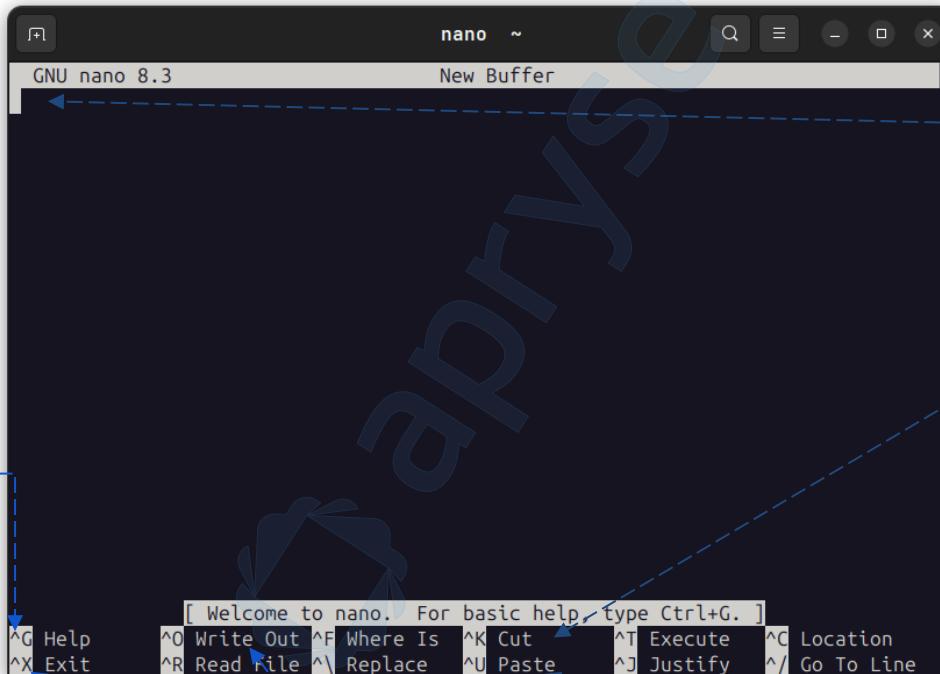
Type nano  
to open  
nano

It lets you create, open, and edit text files without leaving the command line

Comes pre-installed in most Linux systems, so it's always available.

# Nano: Shortcut

The “^” means  
**CTRL**



When typing, your text  
will appear here

**CTRL + K → Cut a line**

**CTRL + X → Exit**

**CTRL + O → Save (Write Out)**

**CTRL + U → Paste**



Vim is a simple text editor like **nano** that runs inside the terminal

### Normal mode

(default)

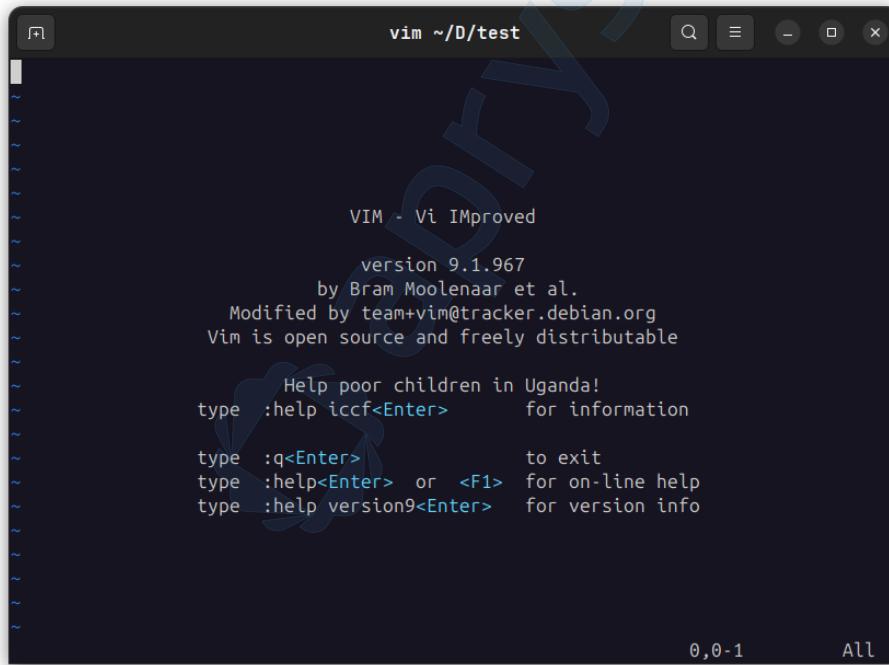
→ move around,  
delete, copy, paste

### Insert mode

→ actually type text

### Command mode

→ give commands  
(like save or quit)



A screenshot of the Vim editor running in a terminal window. The title bar says "vim ~/D/test". The main area displays the Vim startup message:

```
VIM - Vi IMproved
version 9.1.967
by Bram Moolenaar et al.
Modified by team+vim@tracker.debian.org
Vim is open source and freely distributable

Help poor children in Uganda!
type :help iccf<Enter>      for information
type :q<Enter>                to exit
type :help<Enter> or <F1> for on-line help
type :help version9<Enter>    for version info
```

The status bar at the bottom shows "0,0-1" and "All".

Fast  
Lightweight  
Ubiquitous

Not beginner-friendly  
Takes time to master  
Lacks GUI features



i → go to **Insert**  
**mode**  
(start typing)

**Esc** → go back to  
**Normal mode**

**:w → save file**

```
vim ~/_D/test
vim my_document.txt

i
This is an example document.
Press Esc to return to Normal mode.

^
yy
jp
dd
:wq

-- INSERT --
17,1 All
```

**:q → quit vim**

**:wq → save and quit**

**:q! → quit without saving**



A screenshot of the Neovim interface. On the left is a file browser showing the project structure of 'neovim'. The main window displays a portion of the 'screen.c' file with syntax highlighting. A small preview window in the bottom right shows a 'Contributing to Neovim' page.

```
neovim
125 #define MB_FILLER_CHAR '<' /* character used when a double-width
126 * doesn't fit. */
127
128 typedef kvec::withinit_t<DecorProvider> Providers;
129
130 // temporary buffer for rendering a single screenline, so it can
131 // be compared with previous contents to calculate smallest delta.
132 // Per-cell attributes
133 static size_t linebuf_size = 0;
134 static schar_T *linebuf_char = NULL;
135 static sattr_T *linebuf_attr = NULL;
136
137 static match_T search_hi; /* used for 'hlsearch' highlight
138
139 StlClickDefinition *tab_page_click_defs = NULL;
140
141 long tab_page_click_defs_size = 0;
142
143 // for line_putchar. Contains the state that n
144 // putting one character to the next.
145 typedef struct {
146     const char *p;
147     int prev_c; // previous Arabic character
148     int prev_c1; // first composing char for pr
149 } LineState;
150 #define LINE_STATE(p) { p, 0, 0 }
151
152 // Whether to call "ui_call_grid_resize" in w
153 static bool send_grid_resize = false;
154
155 static bool conceal_cursor_used = false;
156
screen.c
```

**NeoVim** is a fork (copy) of the Vim

Highly popular among  
developers for deep  
customization

# Bash Script

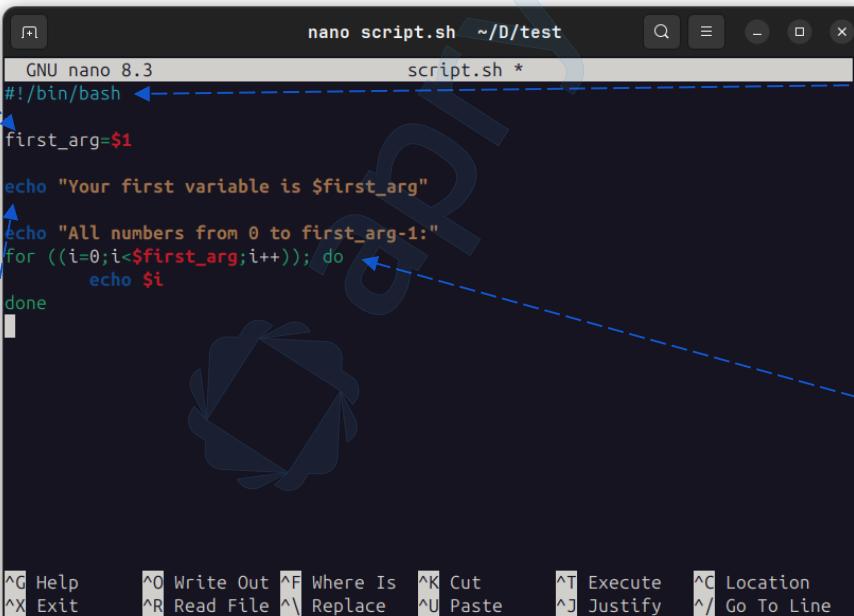
A bash script is like a recipe made of Linux commands

Declare a variable first arg to be your input argument

This line means to use the Bash shell to interpret

Print your first ever variable

Print all numbers from 0 to first arg-1



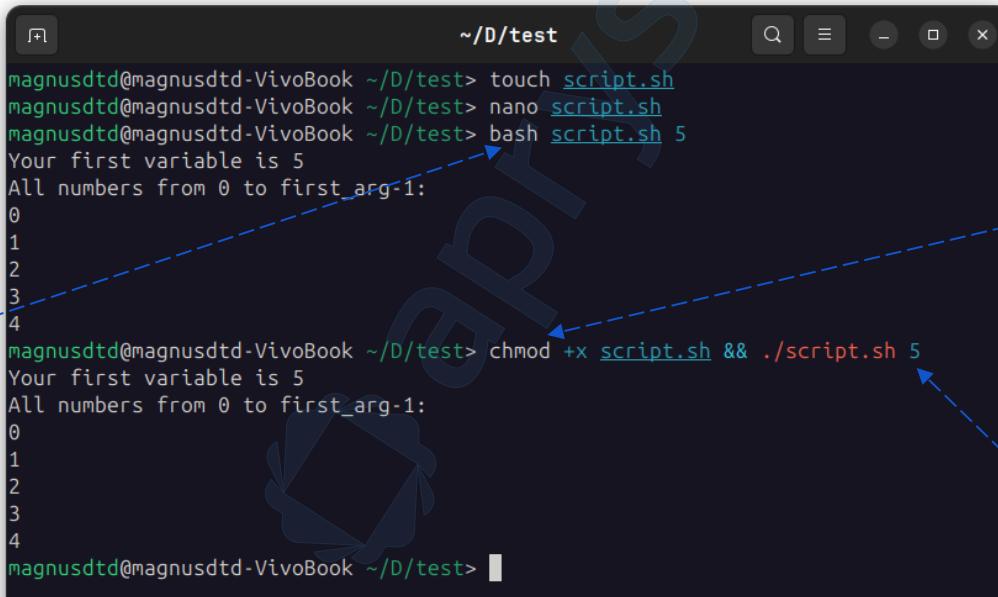
The screenshot shows a terminal window titled "nano script.sh ~/D/test" containing the following bash script code:

```
#!/bin/bash
first_arg=$1
echo "Your first variable is $first_arg"
echo "All numbers from 0 to first_arg-1:"
for ((i=0;i<$first_arg;i++)); do
    echo $i
done
```

The terminal window has a dark background with light-colored text. At the bottom, there is a menu bar with options like Help, Exit, Write Out, Read File, Where Is, Replace, Cut, Paste, Execute, Justify, Location, and Go To Line.

# Bash Script: How to run

Save the script after editing it by **Ctrl + O** and **Ctrl + X**



The terminal window shows the following sequence of commands and output:

```
magnusdtd@magnusdtd-VivoBook ~/D/test> touch script.sh
magnusdtd@magnusdtd-VivoBook ~/D/test> nano script.sh
magnusdtd@magnusdtd-VivoBook ~/D/test> bash script.sh 5
Your first variable is 5
All numbers from 0 to first_arg-1:
0
1
2
3
4

magnusdtd@magnusdtd-VivoBook ~/D/test> chmod +x script.sh && ./script.sh 5
Your first variable is 5
All numbers from 0 to first_arg-1:
0
1
2
3
4
```

Annotations with dashed blue boxes and arrows point to specific parts of the terminal output:

- A box labeled "Run with bash" points to the first execution of the script (`bash script.sh 5`).
- A box labeled "Give execute permission" points to the command (`chmod +x script.sh`) used to make the script executable.
- A box labeled "First argument" points to the value passed as the first argument (`5`) in both executions.

# Bash Script: Syntax

## String

sets the delimiter to /

```
nano string.sh ~/D/test
GNU nano 8.3
string.sh *
my_path="/home/quandv/Documents/home/ml/linux/scripts"
echo "Replaced the first 'home': ${my_path/home/house}"
echo "Split my_path by / into an array:"
IFS="/" readarr my_array <<< "$my_path" && echo ${my_array[-1]}
echo "Delete everything up to the last slash: ${my_path##*/}"
echo "Lower case: ${my_path,,}; Upper case: ${my_path^^}"
```

lowercase

uppercase

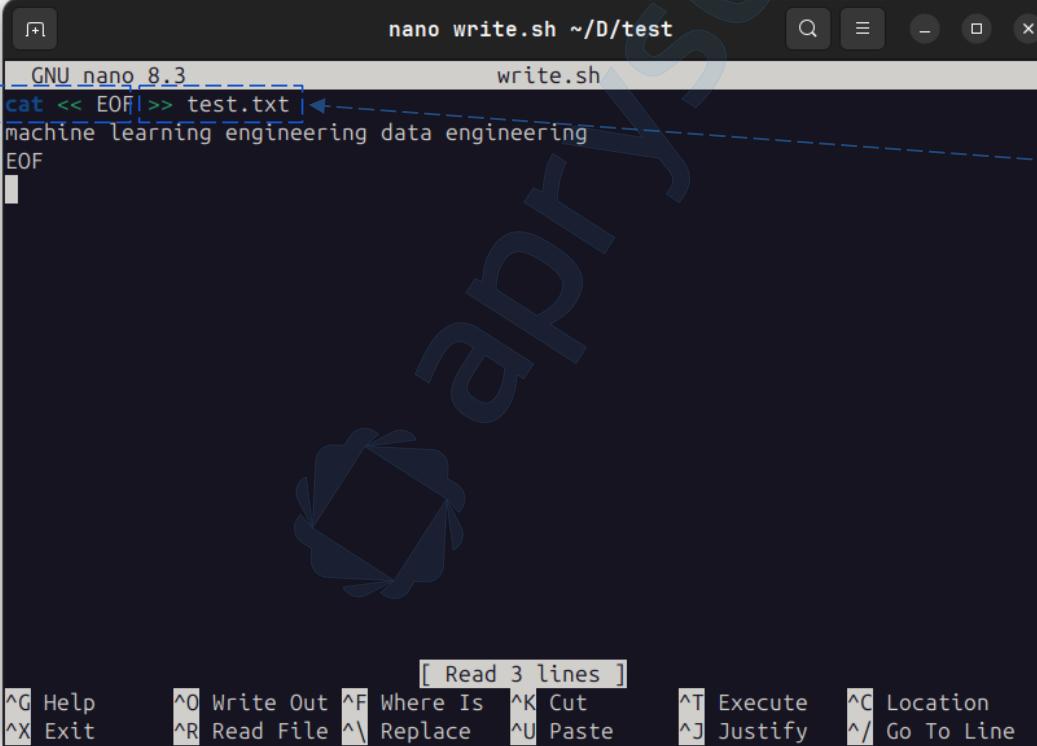
Replaces the first occurrence of home with house

prints the last element of the array

removes everything before the last /

# Bash Script: Syntax

## Write to a file



The screenshot shows a terminal window titled "nano write.sh ~/D/test" with the command "write.sh" entered in the title bar. The text area contains:

```
GNU nano 8.3                               write.sh
cat << EOF >> test.txt
machine learning engineering data engineering
EOF
```

A blue dashed box highlights the line "cat << EOF >> test.txt". A blue arrow points from this box to a callout box containing the text: "starts a **here document**, letting you write multiple lines into a file."

A blue dashed box highlights the line "EOF". A blue arrow points from this box to a callout box containing the text: "appends the content to **test.txt**".

A blue dashed box highlights the line "test.txt". A blue arrow points from this box to a callout box containing the text: "If you use > test.txt, it **overwrites** the file instead of appending".

# Bash Script: Syntax

## Array and condition

```
nano array.sh ~/D/test
array.sh
GNU nano 8.3
my_array=(10 2 300)
echo "First element: ${my_array[0]}"
echo "Last element: ${my_array[-1]}"
echo "Number of elements: ${#my_array[@]}
```

creates an array with 3 elements

gets the **first element**

gets the **last element**

gets the **last element**

string comparison

Don't use **[[ 10 > 9 ]]**  
 but use  
**(( 10 > 9 ))** for  
 numerical operations

```
nano compare.sh ~/D/test
compare.sh *
GNU nano 8.3
if [[ 10 > 9 ]] then
  echo "10 > 9 is true"
else
  echo "10 > 9 is false"
fi
```

# Bash Script: Syntax

## For loop

```
for ((i = 0; i < 5; i++)) do  
    echo $i  
done
```

## While loop

```
i = 5  
while (( $i > 0 )); do  
    echo $i  
    ((i = i - 1))  
done
```

## If else

```
#Demo if-else  
if [[ $first_arg > 0 ]]; then  
    echo "First argument is positive"  
elif [[ $first_arg == 0 ]]; then  
    echo "First argument is zero"  
else  
    echo "First argument is negative"  
fi
```

# Comment

# Bash Script: Syntax



## Function

Defining a Bash Function

```
▶ add() {  
    sum=$(( $1 + $2 ))  
    echo $sum  
}  
result=$(add 5 3)  
echo "The sum is: $result" # Output: The sum is: 8
```

Calling a Bash Function

Another way to define a Bash function

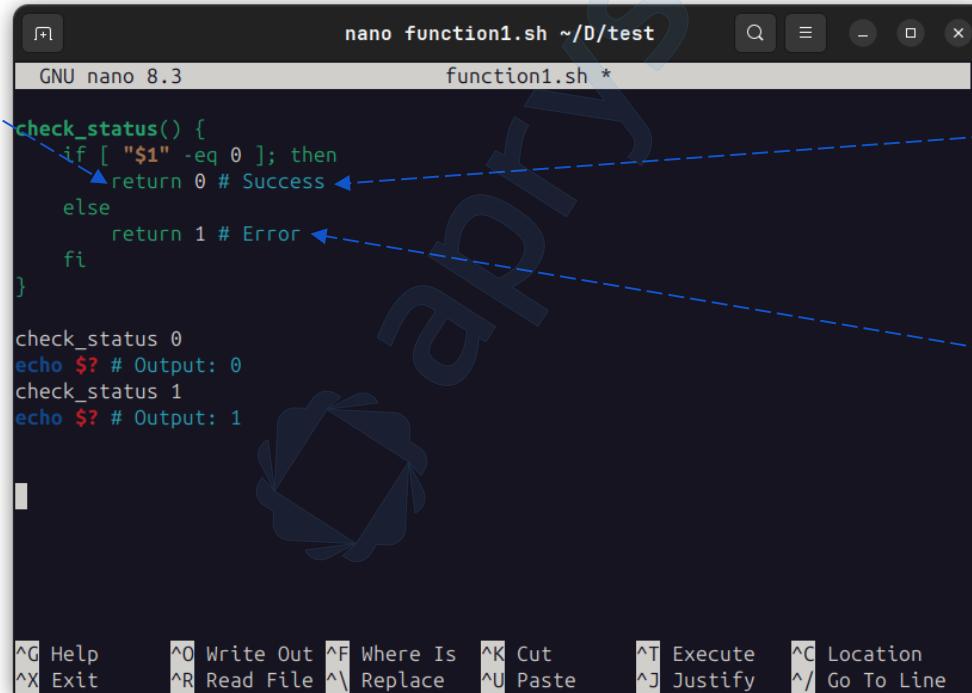
```
▶ function add {  
    sum=$(( $1 + $2 ))  
    echo $sum  
}  
result=$(add 5 3)  
echo "The sum is: $result" # Output: The sum is: 8
```

# Bash Script: Syntax

## Function

Using the **return** statement

This statement sets the exit status of the function (a numerical value between 0 and 255)



The screenshot shows a terminal window titled "nano function1.sh ~/D/test" with the following code:

```
function1.sh *
GNU nano 8.3
check_status() {
    if [ "$1" -eq 0 ]; then
        return 0 # Success
    else
        return 1 # Error
}
check_status 0
echo $? # Output: 0
check_status 1
echo $? # Output: 1
```

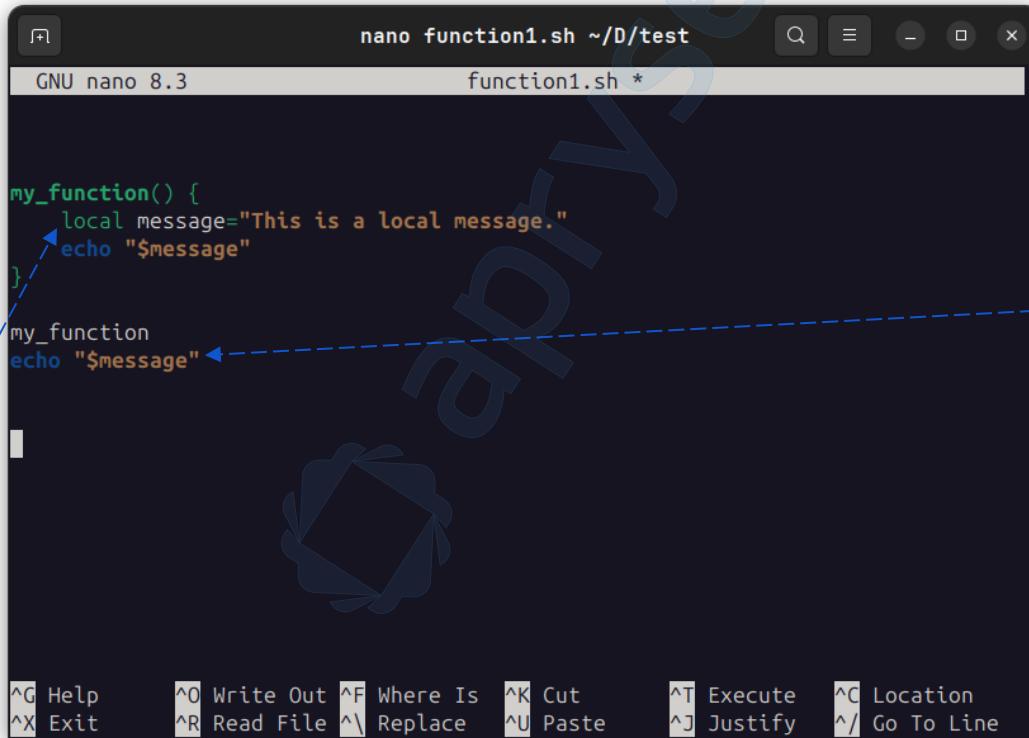
The terminal includes a decorative watermark of a laurel wreath.

0 typically indicates success

non-zero values indicate an error

# Bash Script: Syntax

## Local variable



The screenshot shows a terminal window titled "nano function1.sh ~/D/test" with the command "function1.sh \*". The script content is:

```
my_function() {
    local message="This is a local message."
    echo "$message"
}
my_function
echo "$message"
```

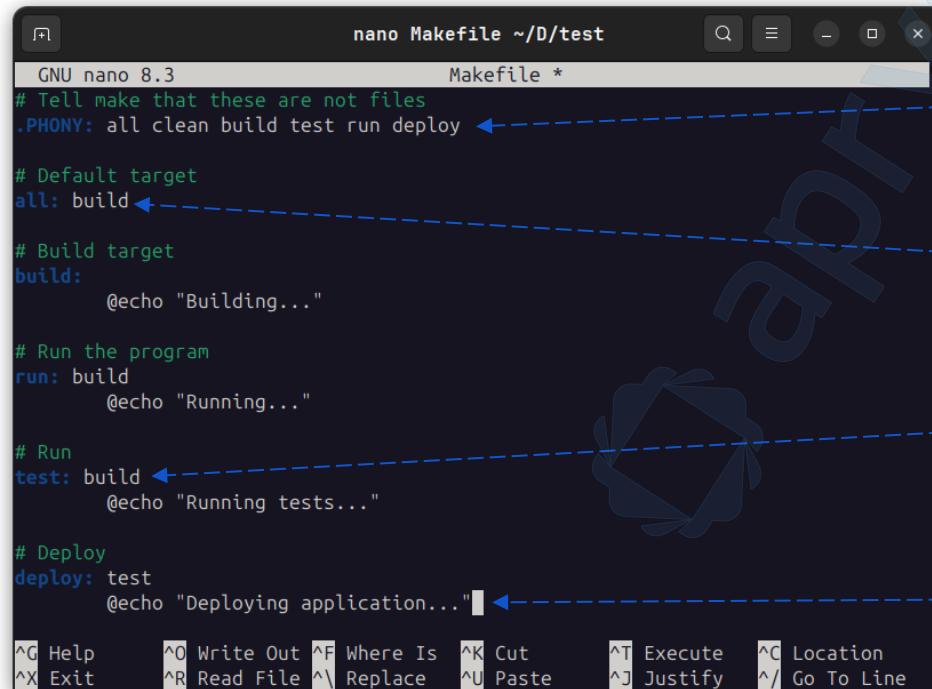
A blue dashed box highlights the line "Defining a local variable" (the first line of the function definition). Another blue dashed box highlights the line "This will be empty or refer to a global variable if it exists" (the line "echo \$message" which refers to the local variable defined in the function).

Defining a local variable

This will be empty or refer to a global variable if it exists

# Makefile

A **Makefile** is like a *to-do list* that automates tasks such as compiling code, running tests, or deploying an application



```
GNU nano 8.3          Makefile *
# Tell make that these are not files
.PHONY: all clean build test run deploy ← .PHONY

# Default target
all: build ← Target

# Build target
build:
    @echo "Building..." ← Dependencies

# Run the program
run: build
    @echo "Running..." ← Recipe

# Run
test: build ← Dependencies
    @echo "Running tests..." ← Recipe

# Deploy
deploy: test
    @echo "Deploying application..." ← Recipe

^G Help      ^O Write Out  ^F Where Is  ^K Cut      ^T Execute  ^C Location
^X Exit      ^R Read File  ^\ Replace   ^U Paste    ^J Justify  ^/ Go To Line
```

**.PHONY**

→ marks targets that are not files

**Target** → a task name

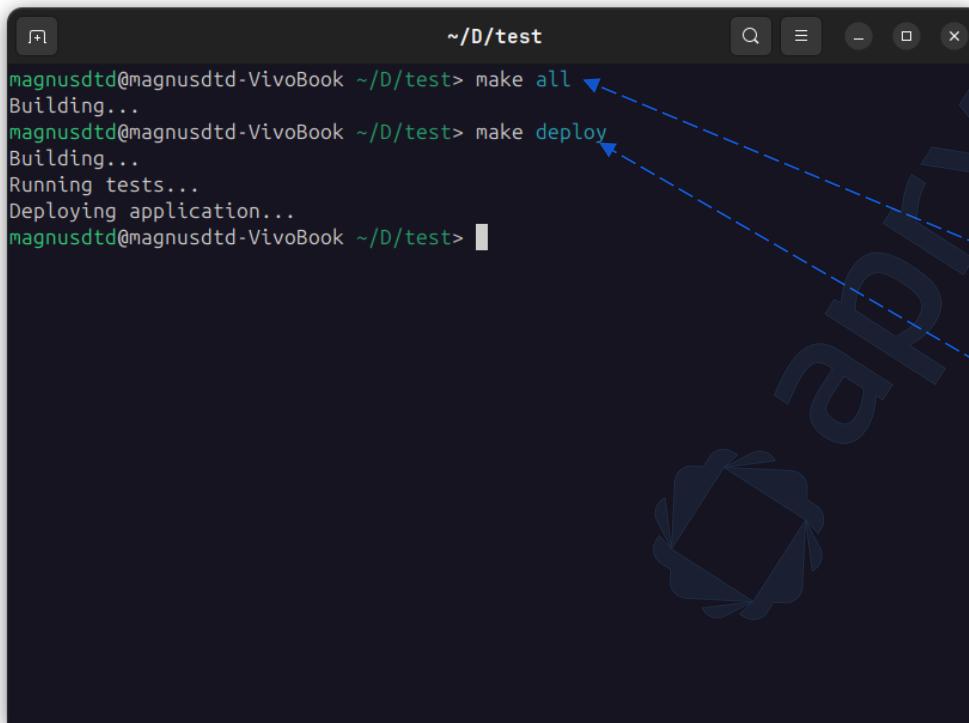
**Dependencies**

→ what must be done before the task

**Recipe**

→ commands to execute the task

# Makefile: How to run



A terminal window titled '~D/test' showing two command executions:

```
magnusdtd@magnusdtd-VivoBook ~/D/test> make all
Building...
magnusdtd@magnusdtd-VivoBook ~/D/test> make deploy
Building...
Running tests...
Deploying application...
magnusdtd@magnusdtd-VivoBook ~/D/test>
```

The terminal has a dark background with light blue text. A faint watermark of a laurel wreath and the word 'apprise' is visible across the screen.

Use the command:

**make + <target>**

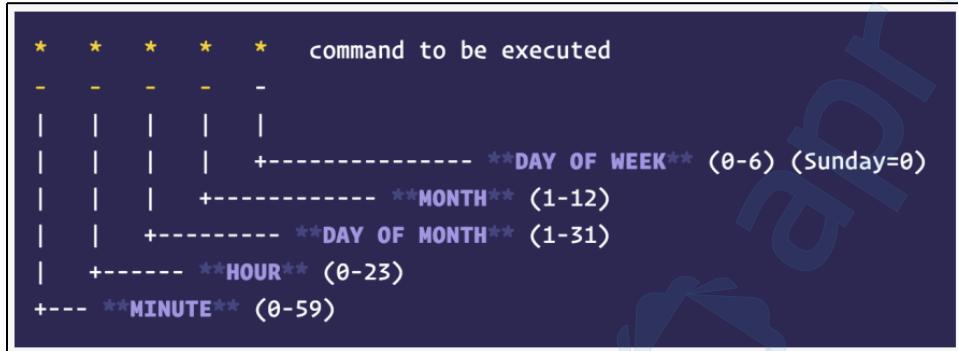
Run the **default target**

Run the **deploy target**

→ Saves time by running repetitive tasks automatically

# CronJob

A **cronjob** is like an **alarm clock** for your Linux tasks.



Source: <https://ahmadawais.com/setup-cron-in-unix-basic-understanding/>

## Examples:

- **5 4 \* \* \*** → At 04:05 everyday
- **\* \* \* \* \*** → Every minute
- **00 03 25 09 \*** → At 03:00 on Sep 25th

# CronJob: How to run

List all crontabs: **crontab -l**

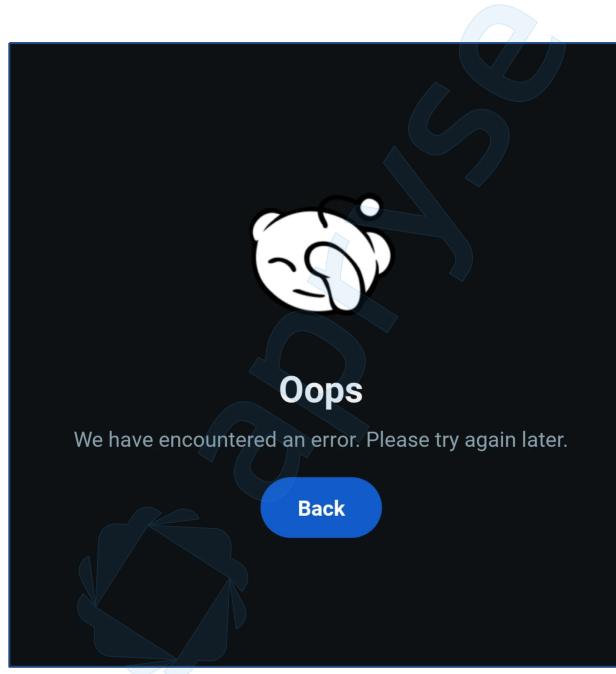
Add a crontab: **crontab -e**

Run a script named **backup.sh** located in your home directory **every day at 3:00 AM**

```
crontab -e ~/D/test
GNU nano 8.3          /tmp/crontab.SpK7AW/crontab *
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').
#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h  dom mon dow   command
0 3 * * * /home/your_username/backup.sh
```

# CronJob: Limitations

No auto-retry

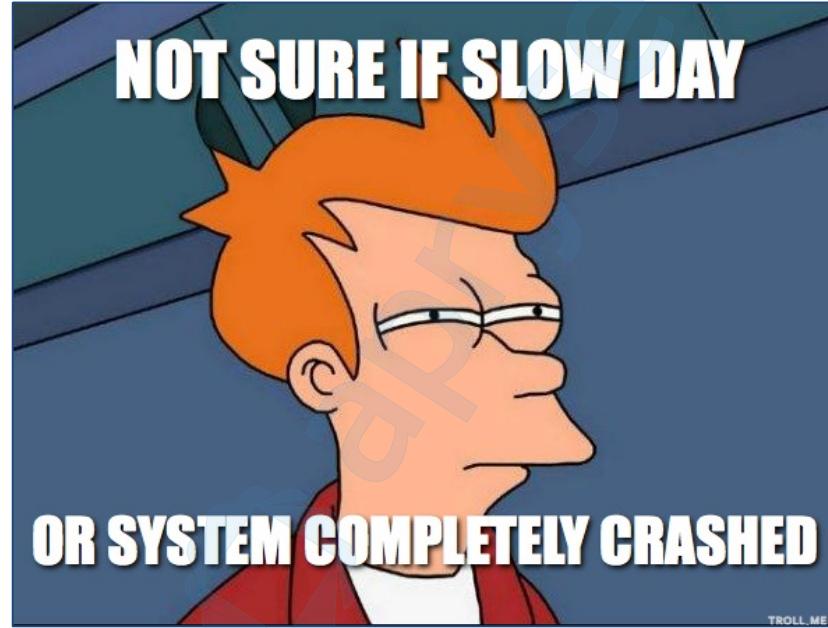


If a job fails, cron will not automatically retry the task

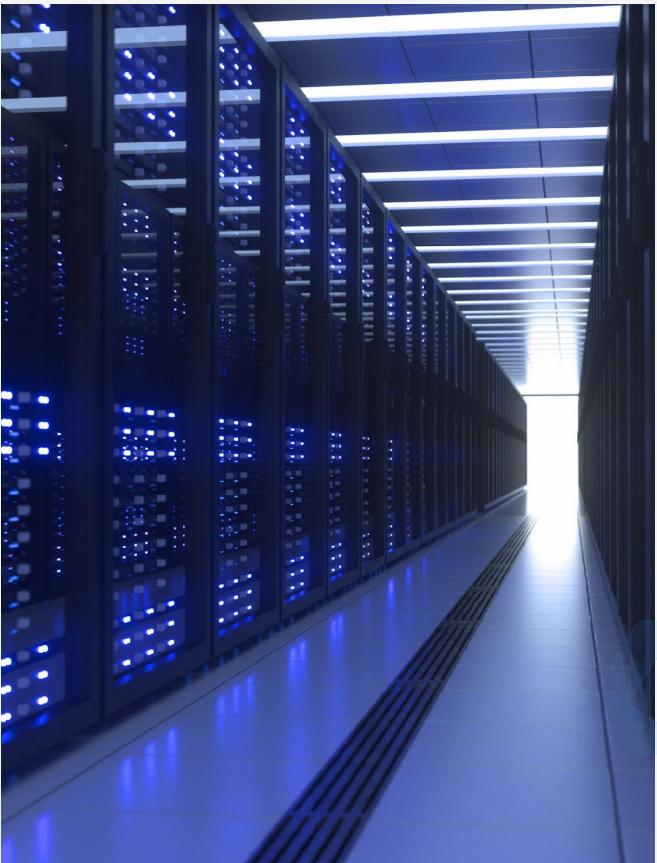
It will only run again at the next scheduled time

# CronJob: Limitations

Missed jobs



If the system is powered off or the cron daemon crashes, any jobs that were scheduled during that time will not run and must be reset manually



# References

- [https://www.cis.rit.edu/class/simg211/unixintro/Access\\_Permissions.html](https://www.cis.rit.edu/class/simg211/unixintro/Access_Permissions.html)
- <https://linuxize.com/post/chmod-command-in-linux/>
- <https://labs.iximiuz.com/playgrounds/ubuntu>
- <https://linuxhandbook.com/linux-directory-structure/>
- <https://www.geeksforgeeks.org/sed-command-in-linux-unix-with-examples/>

# Thank you

64

