Example: tty

Output: /dev/pts/0

Bourne Shell

Shell

- A shell is a command-line interface between the user and the kernel (core of the UNIX OS).
- It reads the commands typed by the user, interprets them, and tells the OS what to do.
- Bourne Shell (sh) is the original UNIX shell, simple and widely portable.
- The Bourne Shell is the original UNIX shell, located at /bin/sh.
- Uses $ as the prompt for normal users.
- Supports:
  - ✓ Variables
  - ✓ Control flow (if, while, for)
  - ✓ Command substitution
- Does not support:
  - ✓ Command history
  - ✓ Auto-completion
  - ✓ Arrays or advanced scripting features
- Creating **cron job** scripts
  - ✓ A cron job is a scheduled task that runs automatically at specified intervals using the cron daemon in UNIX/Linux systems.
  - ✓ You can create cron jobs using Bourne Shell scripts to automate tasks like backups, cleanup, logging, etc.
  - ✓ **Crontab** - The cron table file that contains the list of scheduled jobs. Each user has their own crontab.
  - ✓ Appends(>>) the current date and time to a file: Script: log_time.sh

```
#!/bin/sh
# Appends current date and time to logfile
echo "Current Time: $(date)" >> /home/yourusername/time_log.txt
```

- ✓ Make the script executable:

```
chmod +x /home/yourusername/log_time.sh
```

- ✓ Open the crontab file to schedule the script: nano log_time.sh
- ✓ Enter the cmd crontab –e
- ✓ Add the following line to run the script every day at 6:00 AM:

```
0 6 * * * /bin/sh /home/yourusername/log_time.sh
```

0 – Minute (0th minute)
6 – Hour (6 AM)
* – Every day of the month
* – Every month
* – Every day of the week

- ✓ Once saved, the cron job will be active. You can list all your cron jobs using the
  cmd crontab -l
- ✓ **<u>Cron Syntax Format</u>**

```
* * * * * /path/to/command_or_script
| | | | |
| | | | +----- Day of the week (0 - 7) (Sunday = 0 or 7)
| | | +------- Month (1 - 12)
| | +--------- Day of the month (1 - 31)
| +----------- Hour (0 - 23)
+------------- Minute (0 - 59)
```

Shell Script

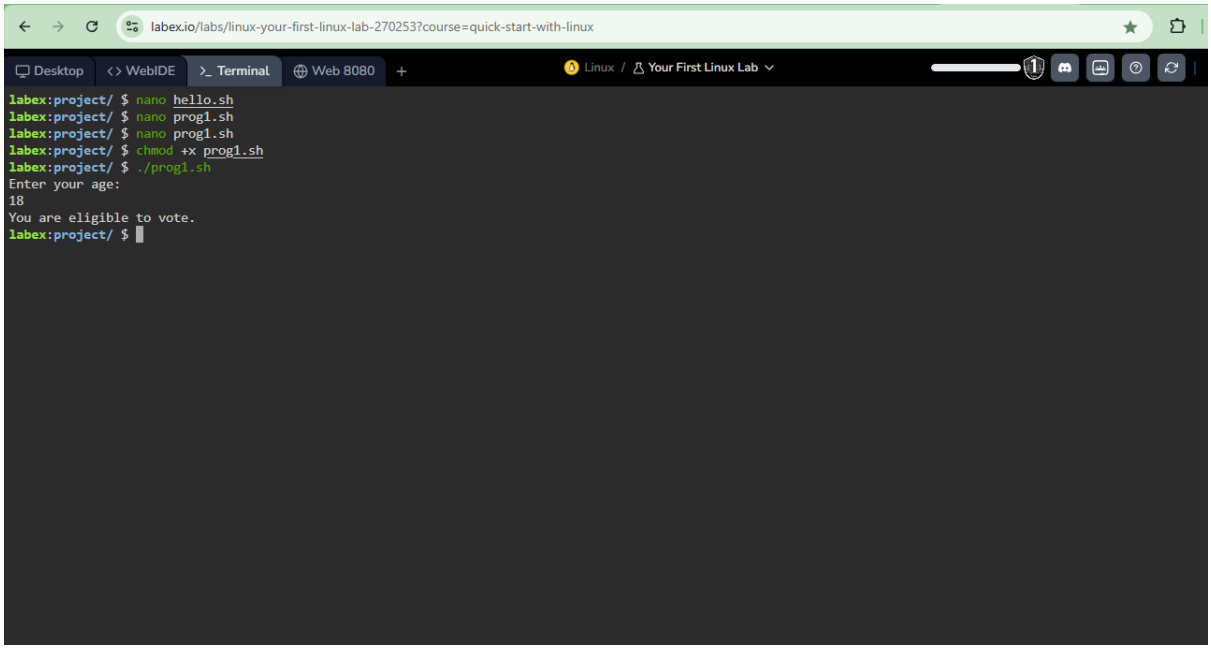
- A shell script is a text file containing a sequence of UNIX/Linux commands written for the shell to execute. It automates tasks like backups, installations, file management, and more.
- Open the Terminal and create a file: nano hello.sh (empty file) new window will be opened, there type the following script and It is saved with a .sh extension (commonly for Bourne and Bash scripts).

```
#!/bin/sh          ← Shebang: tells system which shell to run the script
echo "Hello, World!"
```

```
# User Input Script
#!/bin/sh
echo "Enter your name:"
read name
echo "Welcome, $name!"
```

```
# Loop Script
#!/bin/sh
for i in 1 2 3 4 5
do
  echo "Number is $i"
done
```

```
# Check Voting Eligibility
#!/bin/sh
echo "Enter your age:"
read age
if [ "$age" -ge 18 ]; then
  echo "You are eligible to vote."
else
  echo "You are not eligible to vote."
fi
```



Terminal URL: https://labex.io/labs/linux-your-first-linux-lab-270253?course=quick-start-with-linux

- In nano:
  - Press Ctrl + O
  - Press Enter to confirm the filename hello.sh
  - Press Ctrl + X to exit the editor
- To make the script executable, use cmd-> chmod +x hello.sh and to run the script ./hello.sh

Setting Up Bash

- Most Unix/Linux systems come with Bash pre-installed.
- To check if Bash is installed, open a terminal and type:

```
[user@localhost] $ bash --version
```

- If Bash isn't installed, you can install it using your system's package manager.
- For example, on Ubuntu/Debian, type:

```
[user@localhost] $ sudo apt-get install bash
```

- On macOS, you can install Bash via Homebrew:

```
[user@localhost] $ brew install bash
```

Running Bash Commands

```
[user@localhost] $ bash --version
GNU bash, version 5.2.21(1)-release (x86_64-pc-linux-gnu)
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later

This is free software; you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
```

- This command shows the Bash version installed on your system.
- You can also write scripts (a list of commands) in a file with a .sh extension

Simple Script Example:

```
#!/bin/bash
echo "Hello, Bash!"
```

Save this in a file called hello.sh and run it with.

Example

```
[user@localhost] $ bash hello.sh
Hello, Bash!
```

Common Bash Commands

Bash commands are how you interact with the operating system and perform tasks.

- ls - List directory contents
- cd - Change the current directory
- pwd - Print the current working directory
- echo - Display a line of text
- cat - Concatenate and display files
- cp - Copy files and directories
- mv - Move or rename files

- rm - Delete files or folders
- touch - Create an empty file or update its time
- mkdir - Create a new folder

```
[user@localhost] $ mkdir my_directory
[user@localhost] $ cd my_directory
[user@localhost] $ touch my_file.txt
[user@localhost] $ ls
my_file.txt
```

Shell Wildcards in UNIX

- In UNIX, wildcards are special characters used in the command-line interface (shell) to match files or directories by pattern instead of using exact names.
- This process is also called filename globbing — it saves time and makes file operations more efficient.
- To search or manipulate multiple files with similar names, avoid typing long or repetitive file names, perform bulk operations like copy, move, delete, or list.

Wildcard	Description	Example	Matches
*	Matches zero or more characters	ls *.txt	All .txt files
?	Matches exactly one character	ls file?.txt	file1.txt, fileA.txt
[...]	Matches any one character enclosed in brackets	ls file[123].txt	file1.txt, file2.txt, file3.txt
[^...] or [!...]	Matches any character not in brackets	ls file[^0].txt	file1.txt, but not file0.txt
{a,b,c}	Matches any of the options	echo {Mon,Tue,Wed}	Mon Tue Wed

Types of Shell Wildcards

1. Asterisk (*)

Meaning: Matches any number of characters (including zero).

How it works:

- Can represent nothing (empty string) or any length of text.
- It does not match directory separators (/) unless you explicitly allow it with shopt -s globstar in bash (** can match recursively).

Examples:

Command	Matches	Doesn't Match
ls *.txt	notes.txt, data.txt, .txt (if hidden file)	notes.txt.bak
rm temp*	temp, temp1, temp_file.txt	Ttemp
ls *log	syslog, error.log, debuglog	logfile.txt

note:

* at the start of a pattern won't match hidden files unless you enable dotglob:
command→ shopt -s dotglob

2. Question Mark (?)

Meaning: Matches exactly one character (any character except /).

How it works:

- You must have one character in that position — no more, no less.
- Useful for files with numbered suffixes or fixed-length names.

Examples:

Command	Matches	Doesn't Match
file?.txt	file1.txt, fileA.txt	file.txt, file10.txt
cp log?.dat backup/	log1.dat, logA.dat	log.dat, log123.dat

3. Square Brackets ([])

Meaning: Matches any one character from a set or range.

How it works:

- Can use ranges like [0-9], [A-Z], [a-z].
- You can negate the set with [^...] (matches anything *not* in the set).
- Case-sensitive unless shell options change that.

Examples:

Command	Matches	Doesn't Match
ls [abc]file.txt	afile.txt, bfile.txt, cfile.txt	dfile.txt
rm [0-9]*.dat	1data.dat, 5file.dat	file.dat
grep '[A-Z]' file.txt	Lines with A, B, C	lines with only lowercase

4. Brace Expansion ({ ….})

Meaning: Expands into all combinations of the values inside.
(Not a wildcard — the shell creates a list first, then runs the command.)

How it works:

- Works with comma-separated lists: {a,b,c}
- Works with numeric ranges: {1..5} → 1 2 3 4 5
- Works with alphabet ranges: {A..D} → A B C D
- Can be nested: file{A,B}{1,2}.txt → fileA1.txt fileA2.txt fileB1.txt fileB2.txt

Examples:

Command	Expansion
cp file{1,2,3}.txt archive/	cp file1.txt file2.txt file3.txt archive/

mkdir project_{alpha,beta}	project_alpha, project_beta
echo {A..C}	A B C
touch report_{2021..2023}.txt	report_2021.txt report_2022.txt report_2023.txt

5. Negated Character Class [^...] [!...]

- Match any single character except the ones listed inside the brackets."
- Two notations you might see:
- [^...] → POSIX standard, works in both regex and shell globbing.
- [!...] → Bash and some other shells allow this as an alternative in globbing only (not regex).

Example:

ls [^0-9]*.txt

Matches: file.txt, abc.txt

Doesn't match: 1data.txt, 9notes.txt (start with a digit)

ls [!aeiou]*

Matches: files not starting with a vowel.

Doesn't match: apple.txt, orange.doc

ls [^A-Z]*

Matches: files not starting with uppercase letters.

Doesn't match: Hello.txt, Zebra.doc

Syntax	**Works in**	**Notes**
[^...]	POSIX regex & shell glob	Common and portable
[!...]	Shell glob only	Not valid in POSIX regex

<u>Wildcard in Action</u>

1. Delete all .tmp files

    rm *.tmp

2. List files starting with data and ending with .csv

ls data*.csv

3. Copy specific numbered files

cp file[1-3].txt /backup/

4. Move all files except .txt files

mv *[^t] /folder/

Simple Filters

In UNIX/Linux, filters are the set of commands that take input from standard input stream i.e. **stdin**, perform some operations and write output to standard output stream i.e. **stdout**. The stdin and stdout can be managed as per preferences using redirection and pipes. Common filter commands are: grep, more, sort.

1.grep Command:

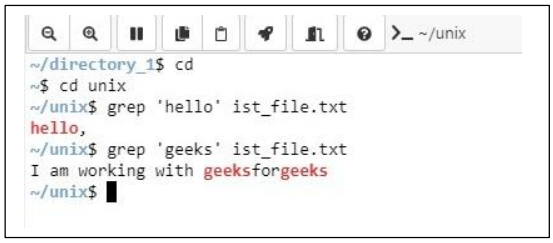
It is a pattern or expression matching command. It searches for a pattern or regular expression that matches in files or directories and then prints found matches.

Syntax:

$grep[options] "**pattern to be matched**" filename

Example:

Cmd→ $grep 'hello' ist_file.txt



S.no	OPTIONS	DESCRIPTiON
01	-v	Returns all lines that do not match the specified regular expression.
02	-n	Returns all lines that match the specified regular expression along with line no.
03	-l	Returns only names of files matching the specified regular expression.
04	-c	Returns count of lines that match regular expression.
05	-i	It is case sensitive option and matches either upper-case or lower-case

Grep command can also be used with meta-characters.