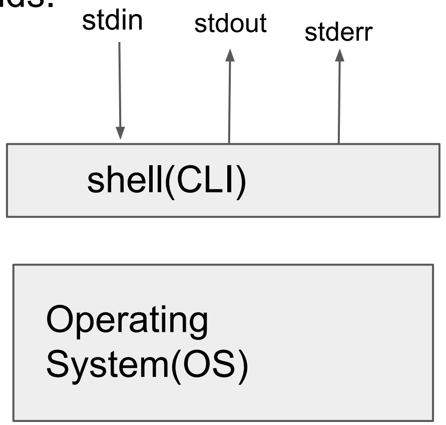
Software Development in Linux Environment

Vehant Technologies

Using Linux as a Developer

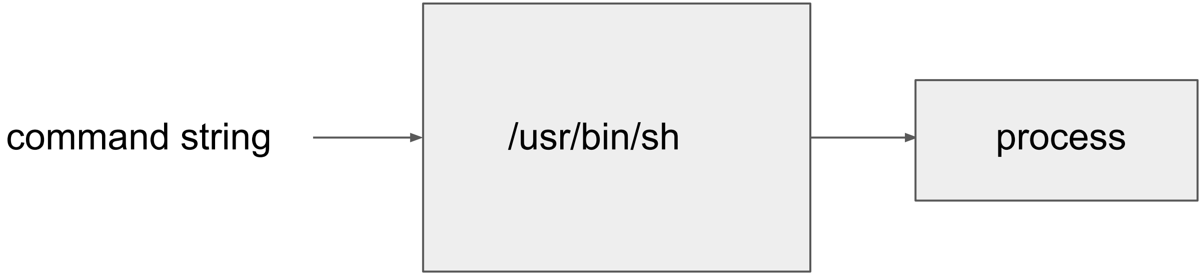
- We interact with the OS via the **shell**, which is a command line interface(**CLI**).
- Using a shell gives us fine grain and direct control over the OS.
- Linux is the OS of choice for backend servers which are usually accessed via shell only.
- We can even write **shell scripts** to automate tasks or set of commands.
- We give **commands** as our input and get results and

  outputs accordingly.
- Shell itself is a **process** running on top of the OS.
- Shell gets input output to and from terminal device(tty, pts, etc).
- Terminal devices are in turn connected to keyboard and screen.

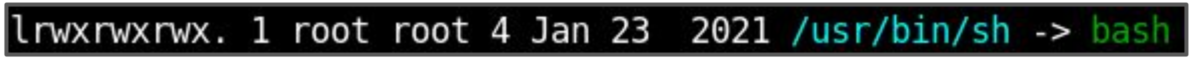


shell - a command line interface, interpreter

- It is used to run commands and thereby interacting with the OS.
- Running commands is same as executing a process.



- **Interactive mode:** Default mode for shell running in terminal or terminal emulators.
- **Non-interactive mode:** Default mode for scripts. Not connected to any terminal.



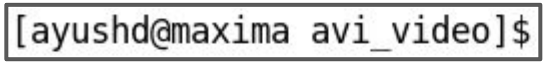
```
lrwxrwxrwx. 1 root root 4 Jan 23  2021 /usr/bin/sh -> bash
```

We use **bash** shell.

shell basics: prompt and command sequences

- Each shell has a **prompt**, which gives us idea about the current session.
- Format: [**username@hostname pwd**]
- Each shell runs as an user

  and has a current working directory.



```
[ayushd@maxima avi_video]$
```

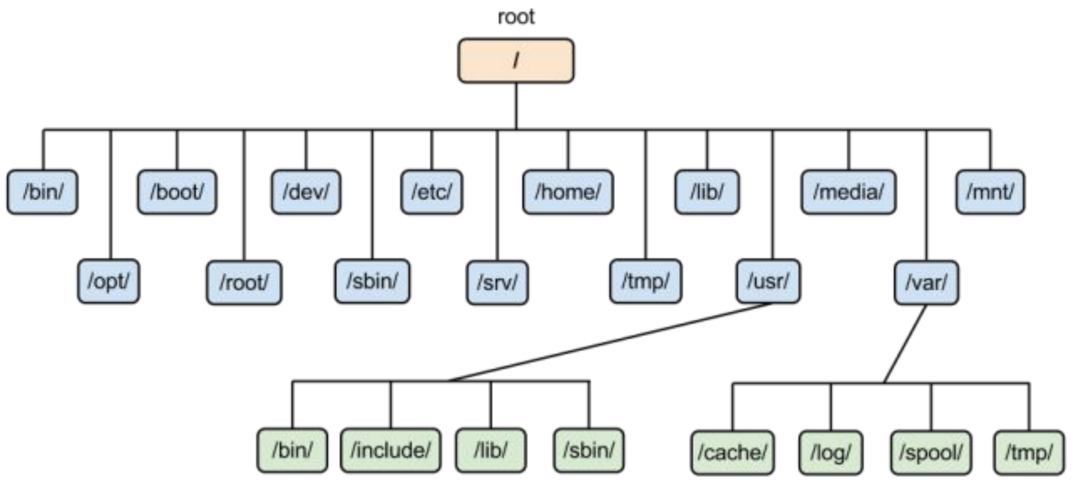
Fig. A shell prompt

- Command syntax: [cmd1] [args]**;**[cmd2] [args]; #unconditional sequence
- Conditional command sequence: [cmd1] [args] **&&** [cmd2] [args]

  cmd2 executes only if cmd1 executes successfully

- Usual flow for using the shell would be to type commands, enter them and wait for the output.
- In a command string **#** indicates start of a comment.

Filesystem structure - tree, hierarchical



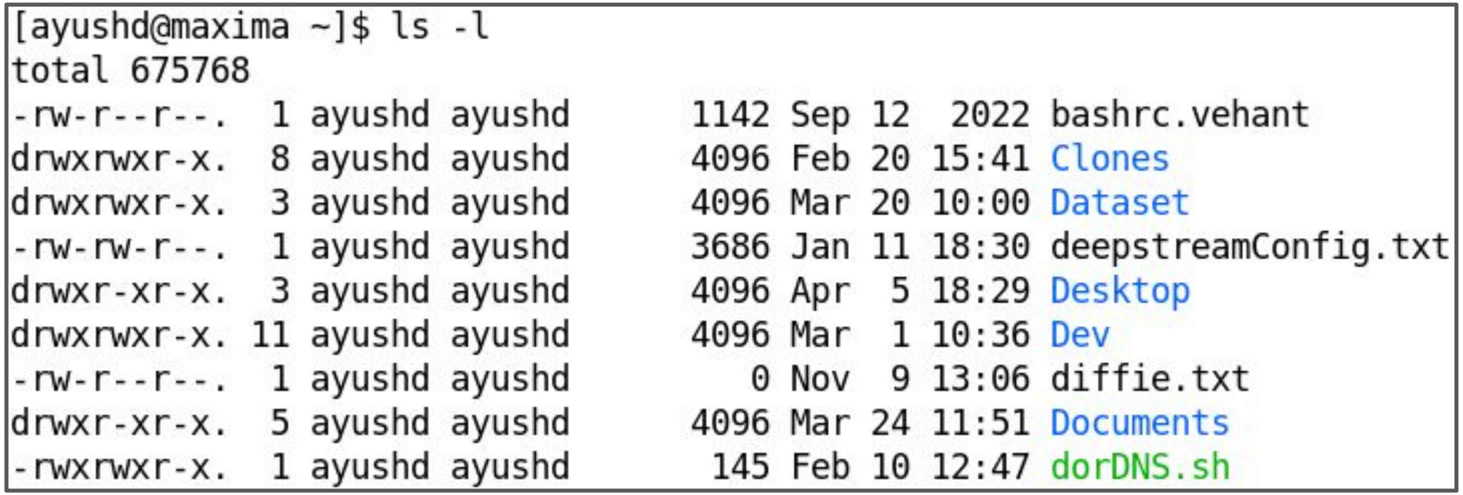
- Each node here is file or directory. Directories themselves are special files.
- **/** (aka **root** directory) is the parent of itself and the starting node.
- Each directory has link to parent directory and to itself.
- These links are have specials names **..**(**double dot**) and **.**(**single dot**) respectively. These are present in each directory.
- File with names starting with a dot are **hidden files** in linux. Example- .bashrc

Important Directories and their purpose

1. /**usr** - Installed software, shared libraries(.so), include files, and static read-only program data. **Common to all** users.
2. /**etc** - **Configuration files** specific to this system. Common to all users.
3. /**opt** - Reserved for the installation of **add-on application** software packages. **Third-party libraries** can be found here.
4. **/tmp** - A world-writable space for temporary files. Files more than 10 days are automatically be deleted from this directory. Can be used for **sharing files between users** as it accessible by all the users.
5. **/home** - Contains home directories for regular users. Can be used to store **user specific** data, libraries and configurations. Usually points to a dedicated disk partition. Example - /home/vehant/ (ie home directory path for user vehant)
6. **/mnt** - Can contain various **mount points** other than the standard ones such as disk partitions or storage devices. At Vehant we have one or two mount points called /mnt/Data0 /mnt/Data1 which point to 2 disk partitions. Used to store data.
7. /boot - Contains boot images for OS and bootloader. It is a system directory and must not be modified.
8. /dev - Contains device files which are basically interface to the underlying hardware.
9. /sys - Contains virtual files to query and interact with the linux kernel(OS).

1.1 ls - list command

- **Long form** output, frequently used
- format: **<permission bits>** <no. of contained files> **<owner user> <owner group> <size> <last update timestamp> <name>**



```
[ayushd@maxima ~]$ ls -l
total 675768
-rw-r--r--.  1 ayushd ayushd      1142 Sep 12  2022 bashrc.vehant
drwxrwxr-x.  8 ayushd ayushd      4096 Feb 20 15:41 Clones
drwxrwxr-x.  3 ayushd ayushd      4096 Mar 20 10:00 Dataset
-rw-rw-r--.  1 ayushd ayushd      3686 Jan 11 18:30 deepstreamConfig.txt
drwxr-xr-x.  3 ayushd ayushd      4096 Apr  5 18:29 Desktop
drwxrwxr-x. 11 ayushd ayushd      4096 Mar  1 10:36 Dev
-rw-r--r--.  1 ayushd ayushd         0 Nov  9 13:06 diffie.txt
drwxr-xr-x.  5 ayushd ayushd      4096 Mar 24 11:51 Documents
-rwxrwxr-x.  1 ayushd ayushd       145 Feb 10 12:47 dorDNS.sh
```

- Adding **-t** sorts the above output according to **last modified** timestamp.
- To also list hidden files(files starting with a dot) use **-a** flag.
- These flags can be combined. Example: ls -lrht .

1.2 cd - change current directory

- cd <destination directory path>   # changes current directory to destination
- cd ~   # ~(tilde) is same as user home directory
- cd -   # jumps to previous working directory
- cd   # changes current directory to home
- cd /   # changes current directory to root directory
- cd $SOME_PATH   # jumps to directory path stored in SOME_PATH variable

Note: Shell variables are frequently used and can be **exported** or **normal**.

Syntax: VAR_1=12   # there should be no space between equal sign variable and it’s value.

Values can later be accessed using **$**VAR (dollar sign) in commands.

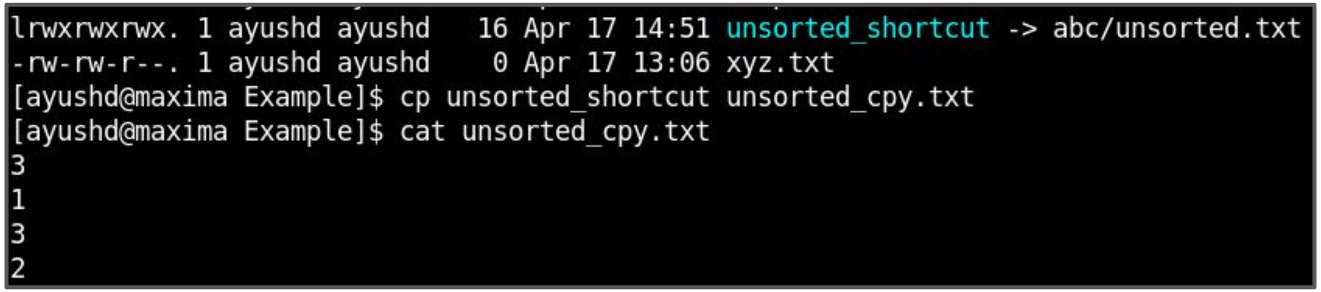
1.3 cp - copy command

Syntax: cp [source] [destination directory path]   # copy file to destination

cp [source] [full path to copy]   # specify name for the copy

cp -r [directory path] [destination]   # copy directory

- Use **-a** flag to preserve permission attributes of the original file.
- If source is a soft-link(aka shortcut in linux) cp creates copy of the file the link is pointing to(ie it dereferences to the file).



```
lrwxrwxrwx. 1 ayushd ayushd   16 Apr 17 14:51 unsorted_shortcut -> abc/unsorted.txt
-rw-rw-r--. 1 ayushd ayushd    0 Apr 17 13:06 xyz.txt
[ayushd@maxima Example]$ cp unsorted_shortcut unsorted_cpy.txt
[ayushd@maxima Example]$ cat unsorted_cpy.txt
3
1
3
2
```

1.4 mv - move, rename command

```
Syntax:  mv [source file]... [destination directory path] # move file(s) to destination

         mv [source file] [new file name]                  # rename file

         mv [file 1] [existing file 2]                     # overwrite file2 and replace it
```

- mv command doesn’t alter the permission bits of the file being moved.
- mv command fails if the user doesn’t have access to the file being moved.

NOTE: Types of path in linux:
- **Absolute path:** It is the full path of a file. Example - **/home/ayushd/Example/sample.txt**
- **Relative path:** It is the path relative to current working directory.
  Example: current directory is /home/ayushd/Example/
  Then relative path of sample.txt would be **./sample.txt** or simply **sample.txt**

1.5 rm - remove command

- removes/unlinks a file. In linux deletion is also called unlinking.
- A file is deleted once all it’s link count becomes 0.
- Use **-r** to **remove a directory**. Also known as **recursive** flag.
- Use **-f** to force remove a file.
- We can use asterisk (*) strings as input to the command to delete multiple files matching a pattern at once. Example - rm *.o

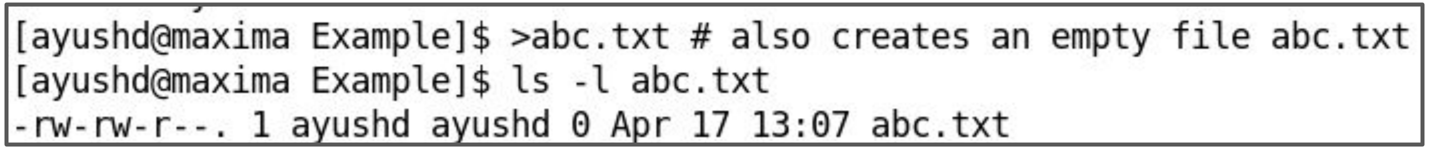
NOTE: A similar command to rm is **rmdir** which removes only empty directories.
Syntax: rmdir [directory name/path(s)]
Command fails if specified directories are not empty.
Use case: Selectively delete only empty directories inside another directory.
Example- rmdir Example/*

1.6 touch - create a file, update file timestamp

Syntax - touch [filename]

- Creates a **regular empty file** if the filename provided doesn't already exist.
- Creates file with default permission attributes.

- Alternate method to create an empty file is by using **> [filename]**

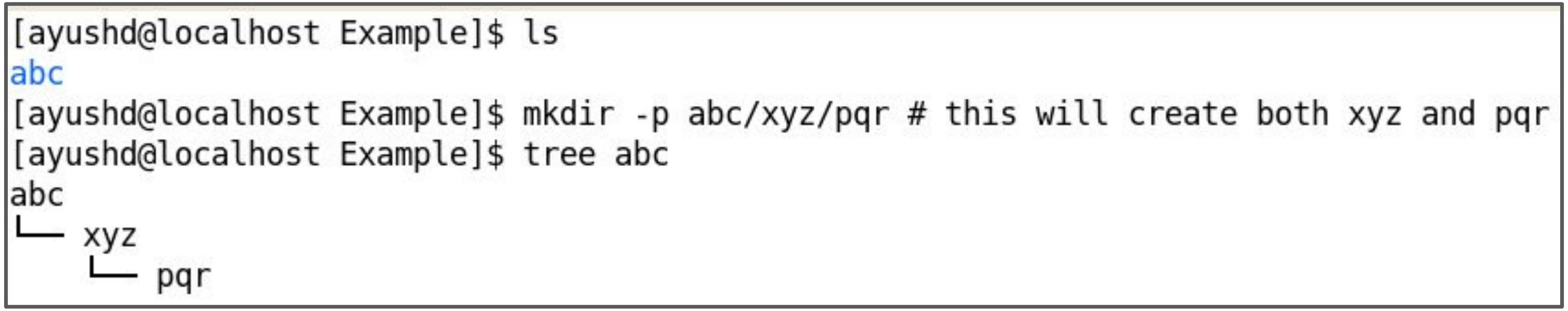


```
[ayushd@maxima Example]$ >abc.txt # also creates an empty file abc.txt
[ayushd@maxima Example]$ ls -l abc.txt
-rw-rw-r--. 1 ayushd ayushd 0 Apr 17 13:07 abc.txt
```

Fig. Creates abc.txt having 0 bytes of data.

1.7 mkdir - create directory

- Used to **create directories**, which are special files to store information and names of other files.
- Syntax - mkdir [directory name/path]
- Use **-p** flag, no error if existing, make parent directories as needed. This flag can be found in Makefiles.



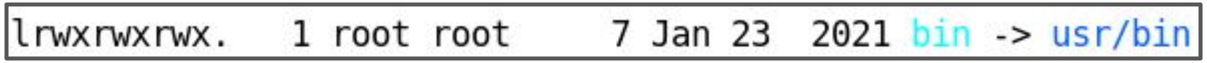
```
[ayushd@localhost Example]$ ls
abc
[ayushd@localhost Example]$ mkdir -p abc/xyz/pqr # this will create both xyz and pqr
[ayushd@localhost Example]$ tree abc
abc
└── xyz
    └── pqr
```

1.8 ln - create link command

- Used to create links, which are special kinds of files.
- Links are of 2 types:
  - **Hard link**: Has an effect of creating an **alias** for the pointed file. **Increments link count** by 1. Deleting the hard link decreasing link count by 1.
  - **Soft link**: Same as shortcut file/**symbolic link**, **doesn't increase link count** of the pointed file. Deleting a soft link doesn't affect the pointed file.

Syntax: ln [target] [link_name] # creates a hard-link by default

ln **-s** [target] [link_name] # creates a soft link/**shortcut** to the target

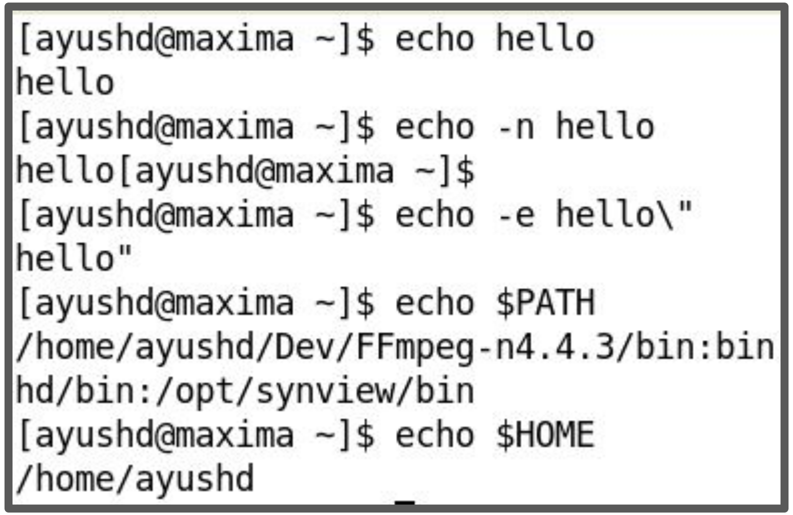


```
lrwxrwxrwx.   1 root root     7 Jan 23  2021 bin -> usr/bin
```

Dig. /bin is symbolic-link to /usr/bin as shown by the long list output of ls command

1.9 echo - outputs a line of text on screen

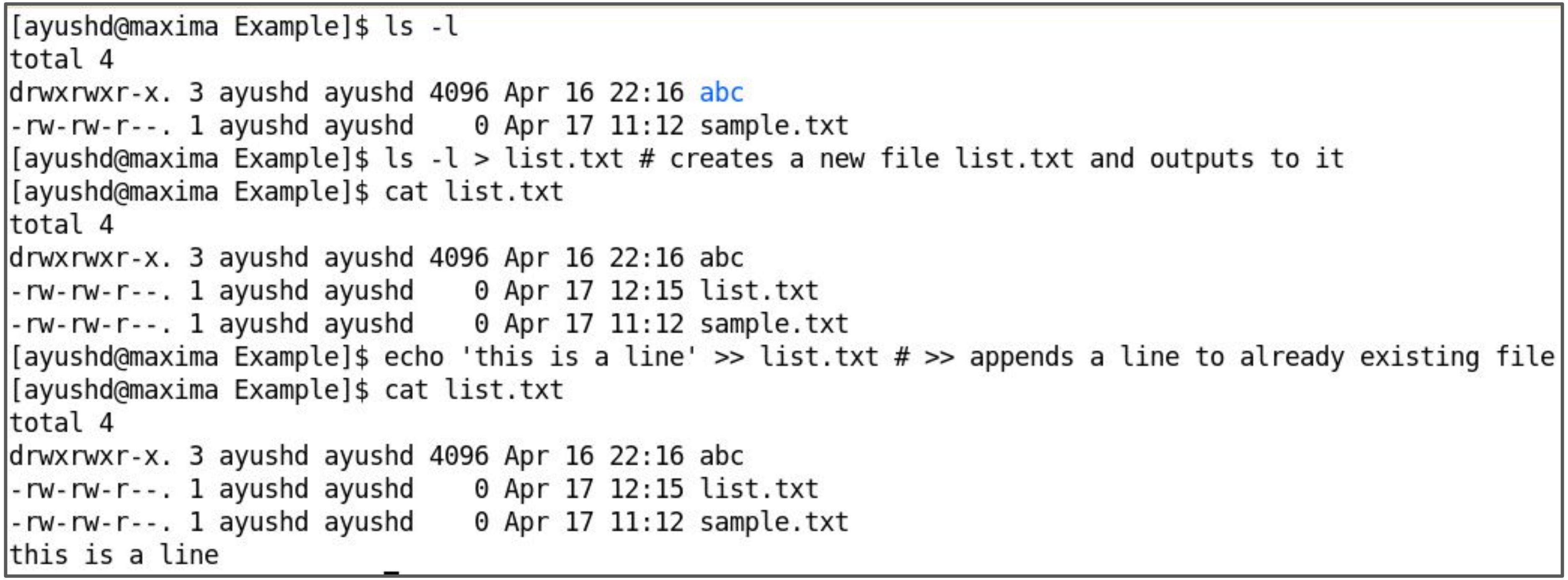
- echo ‘hello!’ # this equivalent of printing “hello\n”
- echo -n ‘hello’ # **-n** flag **removes the newline** from the string “hello”
- echo -e ‘hello\”’ # -e **allows escape sequences** in string hello\”
- Can be used to print shell variables including environment variables.



```
[ayushd@maxima ~]$ echo hello
hello
[ayushd@maxima ~]$ echo -n hello
hello[ayushd@maxima ~]$
[ayushd@maxima ~]$ echo -e hello\"
hello"
[ayushd@maxima ~]$ echo $PATH
/home/ayushd/Dev/FFmpeg-n4.4.3/bin:bin
hd/bin:/opt/synview/bin
[ayushd@maxima ~]$ echo $HOME
/home/ayushd
```

Shell redirection: Saving command outputs to a file

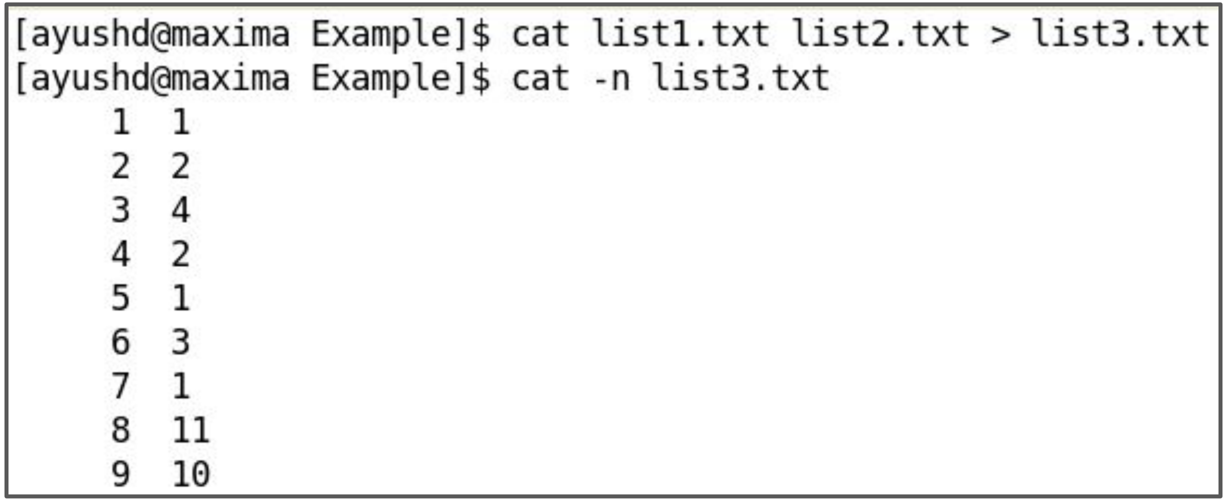
- We can use **>** and **>>** to redirect command outputs to a file.
- If the file doesn't exist, it's created with default permission attributes.



```
[ayushd@maxima Example]$ ls -l
total 4
drwxrwxr-x. 3 ayushd ayushd 4096 Apr 16 22:16 abc
-rw-rw-r--. 1 ayushd ayushd    0 Apr 17 11:12 sample.txt
[ayushd@maxima Example]$ ls -l > list.txt # creates a new file list.txt and outputs to it
[ayushd@maxima Example]$ cat list.txt
total 4
drwxrwxr-x. 3 ayushd ayushd 4096 Apr 16 22:16 abc
-rw-rw-r--. 1 ayushd ayushd    0 Apr 17 12:15 list.txt
-rw-rw-r--. 1 ayushd ayushd    0 Apr 17 11:12 sample.txt
[ayushd@maxima Example]$ echo 'this is a line' >> list.txt # >> appends a line to already existing file
[ayushd@maxima Example]$ cat list.txt
total 4
drwxrwxr-x. 3 ayushd ayushd 4096 Apr 16 22:16 abc
-rw-rw-r--. 1 ayushd ayushd    0 Apr 17 12:15 list.txt
-rw-rw-r--. 1 ayushd ayushd    0 Apr 17 11:12 sample.txt
this is a line
```

1.10 cat - print a text file, con**cat** multiple files

- Syntax - **cat [filename]** # prints contents of the file on terminal
- Use **-n** to also include **line numbers** in the output.

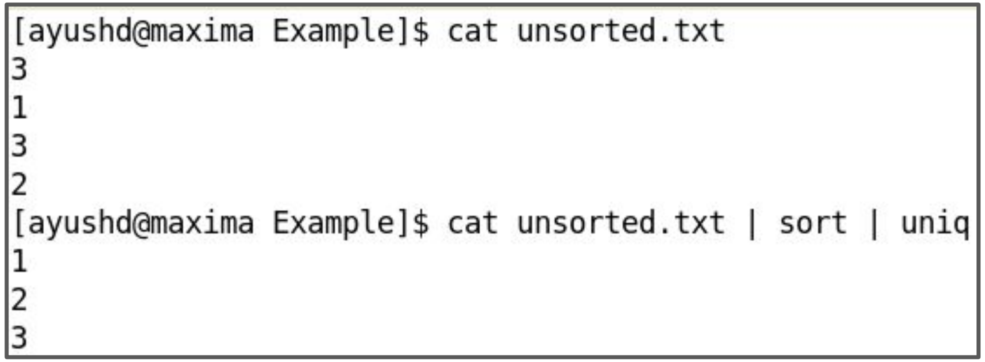


```
[ayushd@maxima Example]$ cat list1.txt list2.txt > list3.txt
[ayushd@maxima Example]$ cat -n list3.txt
     1	1
     2	2
     3	4
     4	2
     5	1
     6	3
     7	1
     8	11
     9	10
```

Fig. The example first concats list1.txt and list2.txt and saves the output to list3.txt. Then it prints list3.txt with line numbers. This demonstrates both uses of cat command.

Pipes - a way to create command pipelines

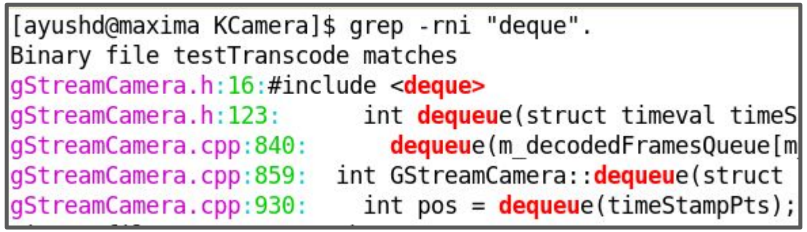
- Various commands can take the output of a previous command as their input.
- We can **combine** such commands together via the **|** shell operator. It is called the **pipe**.
- Simple **pipelines** can often be used for output filtering and transformation using commands such as **grep, tail, head, uniq, sort, awk, sed,** etc. Most of these commands use stream of lines as their input therefore can be attached to other commands via a pipe.
- **Syntax - [cmd1] | [cmd2] | [cmd3]** … # a pipeline can have multiple commands in it
- The flow is from cmd1 to cmd3.



```
[ayushd@maxima Example]$ cat unsorted.txt
3
1
3
2
[ayushd@maxima Example]$ cat unsorted.txt | sort | uniq
1
2
3
```

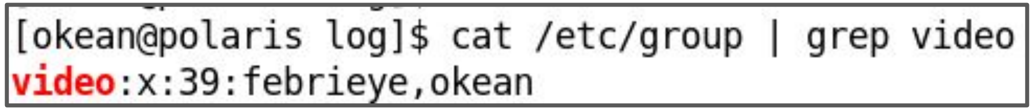
2.1 grep - search and pattern matching

- Syntax - **grep [flags] <pattern string> <input file(s)>** # the pattern string uses **regular expression** syntax
- To search all files in a given **directory** we can use the **-r(recursive)** flag.
- Example - grep -r “name” . # searches for the pattern name in the current directory
- To also print **line numbers** we can add **-n** flag.
- To make output case **insensitive** we can add **-i** flag.
- We can combine these flags. A frequently used when searching in codebases is **-rni**. This form searches all files in a directory and prints matches with line numbers.



```
[ayushd@maxima KCamera]$ grep -rni "deque".
Binary file testTranscode matches
gStreamCamera.h:16:#include <deque>
gStreamCamera.h:123:        int dequeue(struct timeval timeS
gStreamCamera.cpp:840:        dequeue(m_decodedFramesQueue[m
gStreamCamera.cpp:859:  int GStreamCamera::dequeue(struct
gStreamCamera.cpp:930:     int pos = dequeue(timeStampPts);
```

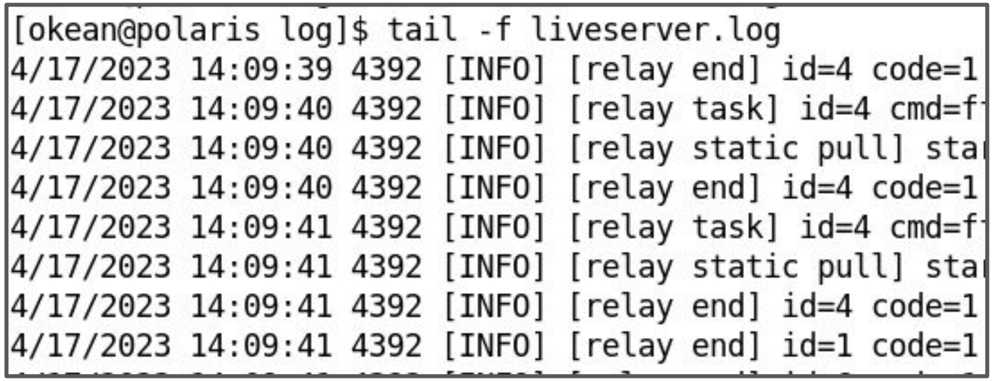
- We can use grep in command pipelines to filter output based off some pattern. Example - we want to find out users in the video group.
- Use **-v** to **invert** grep output.



```
[okean@polaris log]$ cat /etc/group | grep video
video:x:39:febrieye,okean
```

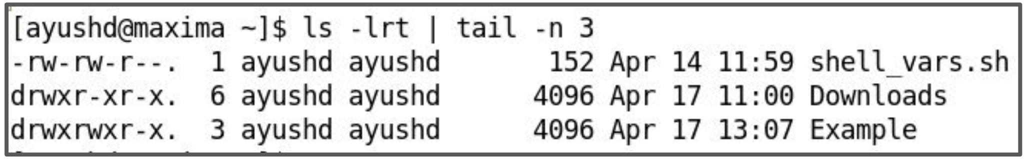
2.2 tail - output last part of file(s)

- Syntax - **tail [filename/path]** # outputs last 10 lines of the file
- Use **-n** flag to specify **number of lines** to be printed.
- Use **-f** flag to follow the file as it grows, useful when the file is continuously growing. Example would be a log file.



```
[okean@polaris log]$ tail -f liveserver.log
4/17/2023 14:09:39 4392 [INFO] [relay end] id=4 code=1
4/17/2023 14:09:40 4392 [INFO] [relay task] id=4 cmd=f
4/17/2023 14:09:40 4392 [INFO] [relay static pull] sta
4/17/2023 14:09:40 4392 [INFO] [relay end] id=4 code=1
4/17/2023 14:09:41 4392 [INFO] [relay task] id=4 cmd=f
4/17/2023 14:09:41 4392 [INFO] [relay static pull] sta
4/17/2023 14:09:41 4392 [INFO] [relay end] id=4 code=1
4/17/2023 14:09:41 4392 [INFO] [relay end] id=1 code=1
```

- tail can also be used in a command pipeline.

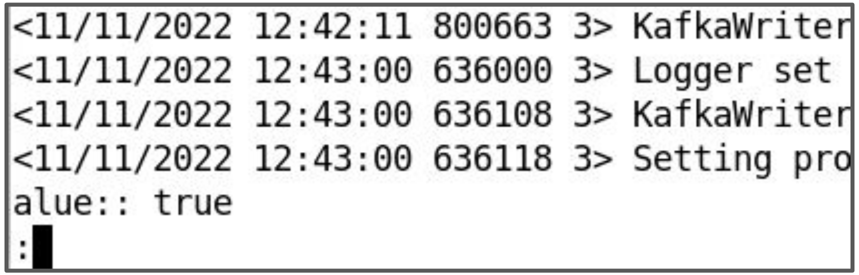


```
[ayushd@maxima ~]$ ls -lrt | tail -n 3
-rw-rw-r--.  1 ayushd ayushd      152 Apr 14 11:59 shell_vars.sh
drwxr-xr-x.  6 ayushd ayushd     4096 Apr 17 11:00 Downloads
drwxrwxr-x.  3 ayushd ayushd     4096 Apr 17 13:07 Example
```

- **head** command is the exact opposite of tail as it output n lines from beginning.

2.3 less - make large output scrollable

- Less commands makes any output **scrollable**.
- We can **scroll up and down** using arrow keys.
- Syntax - [cmd] | less # it is used by attaching the output to less via a pipe

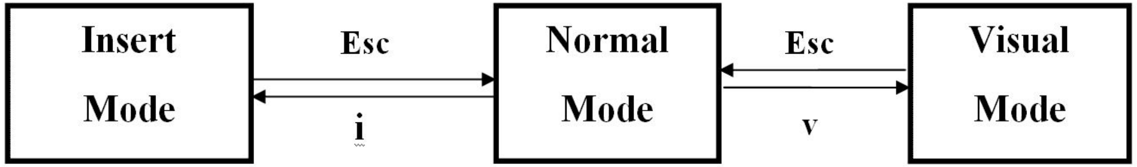


```
<11/11/2022 12:42:11 800663 3> KafkaWriter
<11/11/2022 12:43:00 636000 3> Logger set
<11/11/2022 12:43:00 636108 3> KafkaWriter
<11/11/2022 12:43:00 636118 3> Setting pro
alue:: true
:
```

- We can search forward using /<pattern> command as well as use other **vim commands** in less output.
- To quit we use **q**.

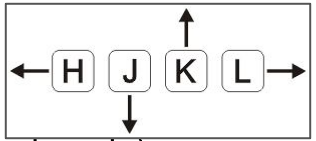
vim - a lightweight command line text editor

- Syntax - **vim [filename/path]** # opens single file in command mode(the screen we first encounter)
- vim has 3 modes:



- Insert mode: we use this mode to directly edit the file, in vim to actually be able to edit we first need to enter this mode.
- **Normal mode**/command mode: **Default mode**. In this mode we can move around the file but cannot directly insert anything. We can execute commands by typing **:(semicolon) followed by a command**
- Simplest commands are the quit commands:
  - **:q - quit the file if not modified, doesn't work if there are unsaved changes**
  - **:q! - force quit the file, unsaved changes are lost**
  - **:wq - save and quit, :w - save the file**

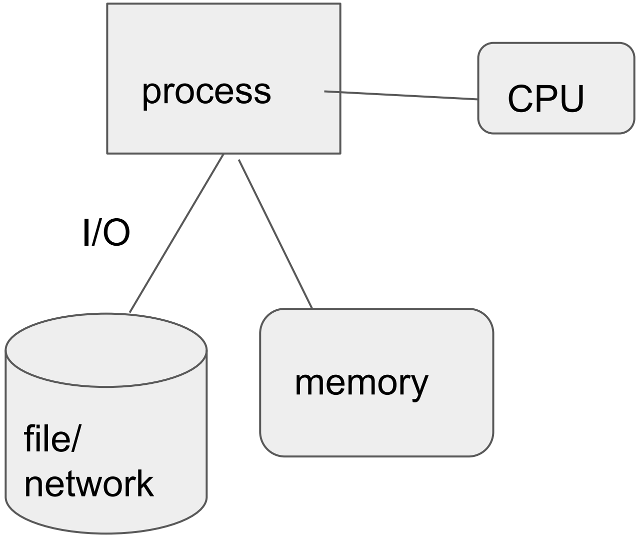
vim - basic workflows



- Basic Navigation - arrow keys(up down right left) or
- Jump to start of specific line **:<linenum>** (in command mode)
- Jump to line 1 , **gg** (in normal mode)
- Jump to last line, **shift + g** (in normal mode)
- Jump to next word, **w** (in normal mode)
- Jump to next curly bracket or scope, **]]** (in normal mode)
- Delete/cut a line, **dd** (in normal mode)
- Undo last change, **u** (in normal mode)
- Copy/yank a line, **yy** (in normal mode)
- Paste line below current line, **p** (in normal mode)
- Insert a line below and enter insert mode, **o** (in normal mode)
- Delete a character under cursor, **x** (in normal mode)
- Replace a character under cursor, **r <new character>** (in normal mode)

Process - running instance of a program/binary

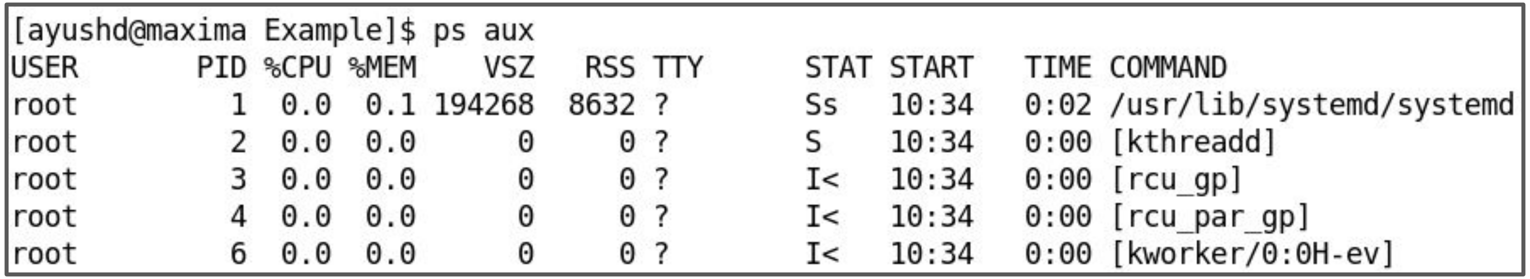
PID - process identifier (int)
UID - user identifier (int), decides access to resources
File descriptor table - table of file handles(inherited from parent process), list of open files, connected devices, sockets, etc. (= Acquired resources)
Environment variables - global key-value (char**) pair(inherited from parent process)
PPID - parent PID(int)



- In linux each process has an **identity** in that it **runs as the owner user**.
- Each process can have multiple threads. Each thread has its own PID.
- A process can have open files and sockets to perform I/O.
- Process can run in **background** as **daemons**.

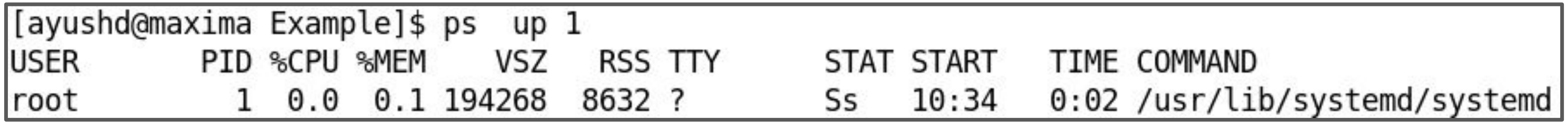
Process management commands: 3.1 ps - list processes

- Syntax - **ps aux** # to see every process on the system



```
[ayushd@maxima Example]$ ps aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.1 194268  8632 ?        Ss   10:34   0:02 /usr/lib/systemd/systemd
root         2  0.0  0.0      0     0 ?        S    10:34   0:00 [kthreadd]
root         3  0.0  0.0      0     0 ?        I<   10:34   0:00 [rcu_gp]
root         4  0.0  0.0      0     0 ?        I<   10:34   0:00 [rcu_par_gp]
root         6  0.0  0.0      0     0 ?        I<   10:34   0:00 [kworker/0:0H-ev]
```

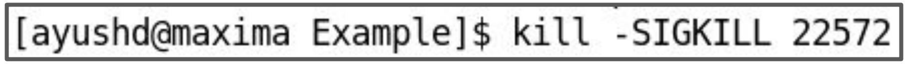
- Checking detail for a list of processes using PID list. Syntax: **ps up <space separated pid list>**



```
[ayushd@maxima Example]$ ps  up 1
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.1 194268  8632 ?        Ss   10:34   0:02 /usr/lib/systemd/systemd
```

Note: In all these forms we **never use '-' dash/hyphen** when specifying flags**.**

3.2 kill - signal a process, kill a process

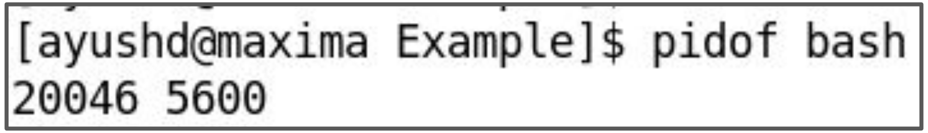
Syntax: **kill -<SIGNUM>/<SIGNAME> <pid>** 

```
[ayushd@maxima Example]$ kill -SIGKILL 22572
```

SIGKILL is the signal which can’t be ignored by any process and leads to it’s termination.

It can be used to terminate unresponsive processes.

3.3 pidof command - gives a list of pid based of process name.

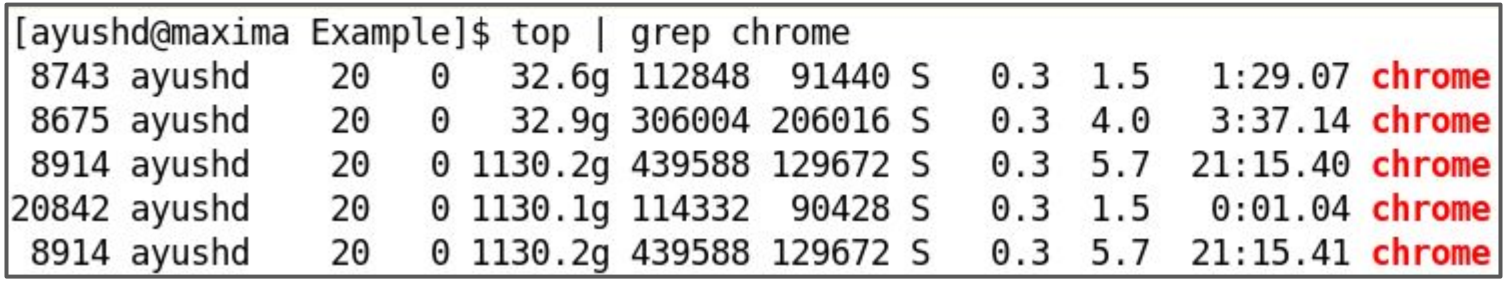


```
[ayushd@maxima Example]$ pidof bash
20046 5600
```

We can use output pids from pidof as input to **kill** and **ps** commands.

3.4 top command - displays process info in real time

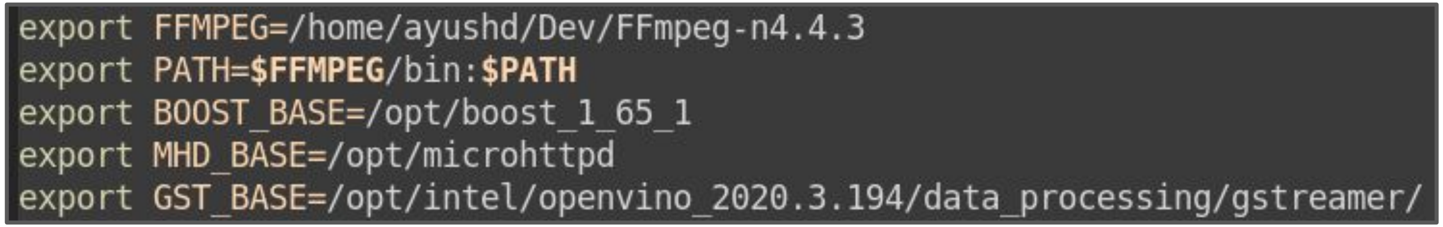
- Can be thought of as a **dynamic version of ps** command.
- It is very useful to analyse **cpu** and **memory usage** of a process over time.
- To view a specific process we can filter the output using **grep** command.



```
[ayushd@maxima Example]$ top | grep chrome
 8743 ayushd    20   0   32.6g 112848  91440 S   0.3  1.5   1:29.07 chrome
 8675 ayushd    20   0   32.9g 306004 206016 S   0.3  4.0   3:37.14 chrome
 8914 ayushd    20   0 1130.2g 439588 129672 S   0.3  5.7  21:15.40 chrome
20842 ayushd    20   0 1130.1g 114332  90428 S   0.3  1.5   0:01.04 chrome
 8914 ayushd    20   0 1130.2g 439588 129672 S   0.3  5.7  21:15.41 chrome
```

Environment Variables - key-value pairs

- **Inherited** from parent process, which is usually the shell.
- For a service these are set using a set environment script.(ex- set_env.sh)
- Shell environment variables are set using builtin **export** command.



```
export FFMPEG=/home/ayushd/Dev/FFmpeg-n4.4.3
export PATH=$FFMPEG/bin:$PATH
export BOOST_BASE=/opt/boost_1_65_1
export MHD_BASE=/opt/microhttpd
export GST_BASE=/opt/intel/openvino_2020.3.194/data_processing/gstreamer/
```

- Bash sets it’s environment variables by running current user’s ***.bashrc*** script at shell startup.
- Environment variables are **exported** which means they are **available to all the commands executed by the shell**.

Important Environment Variables

LD_LIBRARY_PATH - shared library path list for non-standard paths, used by linker during runtime linking/running a binary.

PKG_CONFIG_PATH - list of pkgconfig files(.pc) directory paths for non-standard paths, used by **pkg-config** command, used during compilation.

PATH - list of binaries directory paths.

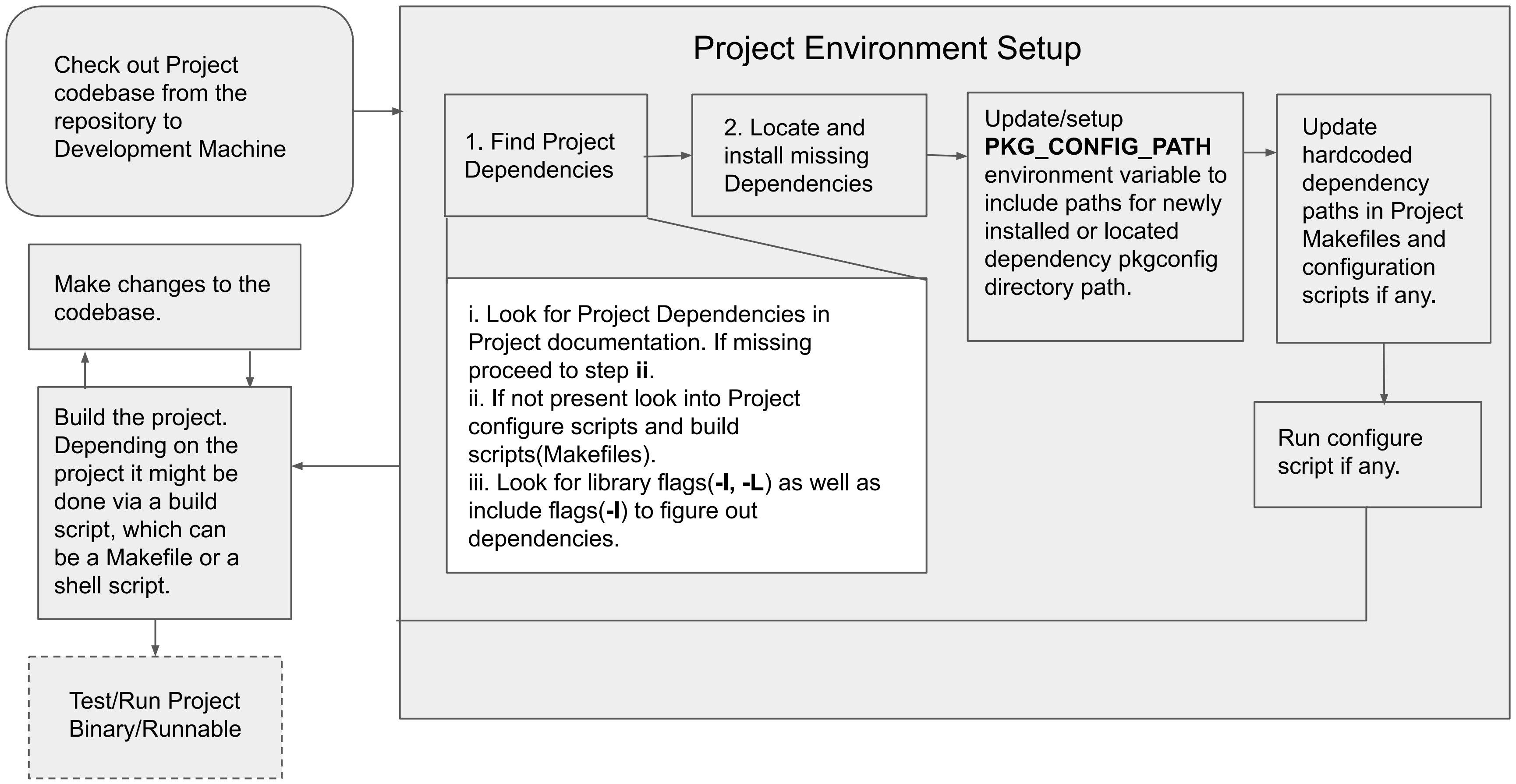
LIBRARY_PATH - shared library path list for non-standard paths, used during compile-time/binary generation.

HOME - points to shell home directory aka ~(tilde)

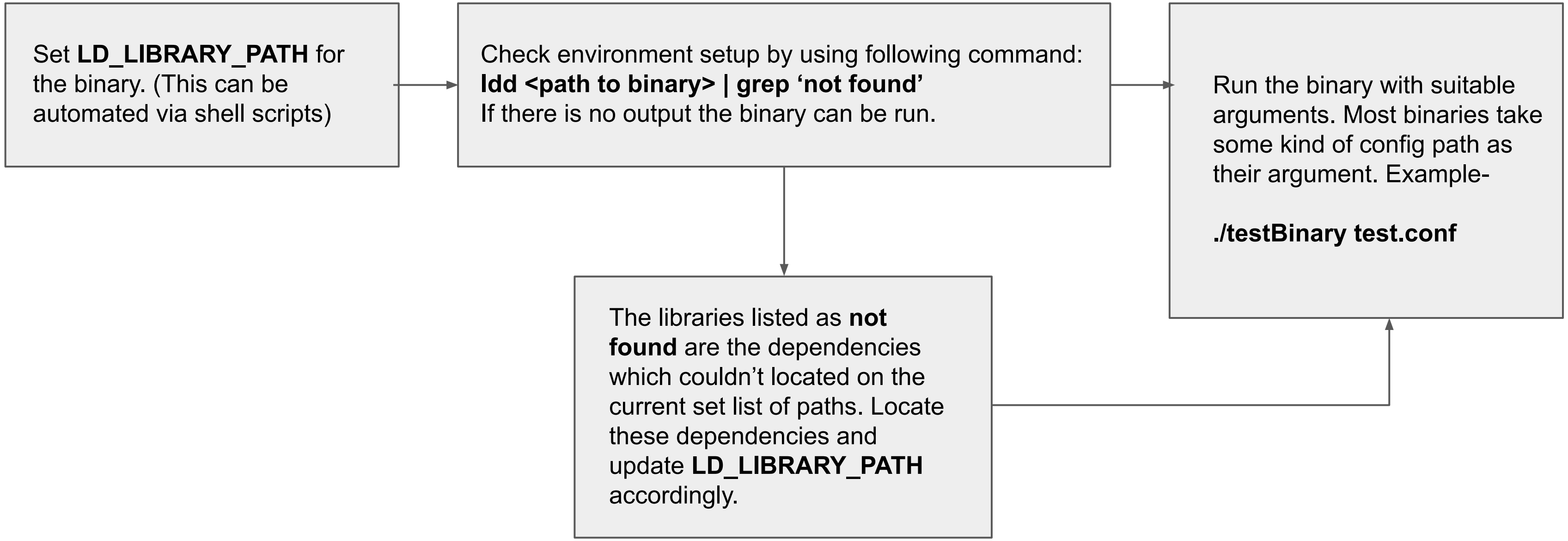
PWD - present working directory

NOTE: The order of paths in these variables does matter, anything that comes first is preferred over the path which comes later.

Basic Development Workflow on a Linux Machine


Check out Project codebase from the repository to Development Machine
Project Environment Setup
1. Find Project Dependencies
2. Locate and install missing Dependencies
Update/setup **PKG_CONFIG_PATH** environment variable to include paths for newly installed or located dependency pkgconfig directory path.
Update hardcoded dependency paths in Project Makefiles and configuration scripts if any.
i. Look for Project Dependencies in Project documentation. If missing proceed to step **ii**.
ii. If not present look into Project configure scripts and build scripts(Makefiles).
iii. Look for library flags(**-l, -L**) as well as include flags(**-I**) to figure out dependencies.
Run configure script if any.
Make changes to the codebase.
Build the project. Depending on the project it might be done via a build script, which can be a Makefile or a shell script.
Test/Run Project Binary/Runnable

Runtime Environment Setup (Running Binaries)



Set **LD_LIBRARY_PATH** for the binary. (This can be automated via shell scripts)

Check environment setup by using following command:
ldd <path to binary> | grep ‘not found’
If there is no output the binary can be run.

The libraries listed as **not found** are the dependencies which couldn’t located on the current set list of paths. Locate these dependencies and update **LD_LIBRARY_PATH** accordingly.

Run the binary with suitable arguments. Most binaries take some kind of config path as their argument. Example-

./testBinary test.conf

Thank you.