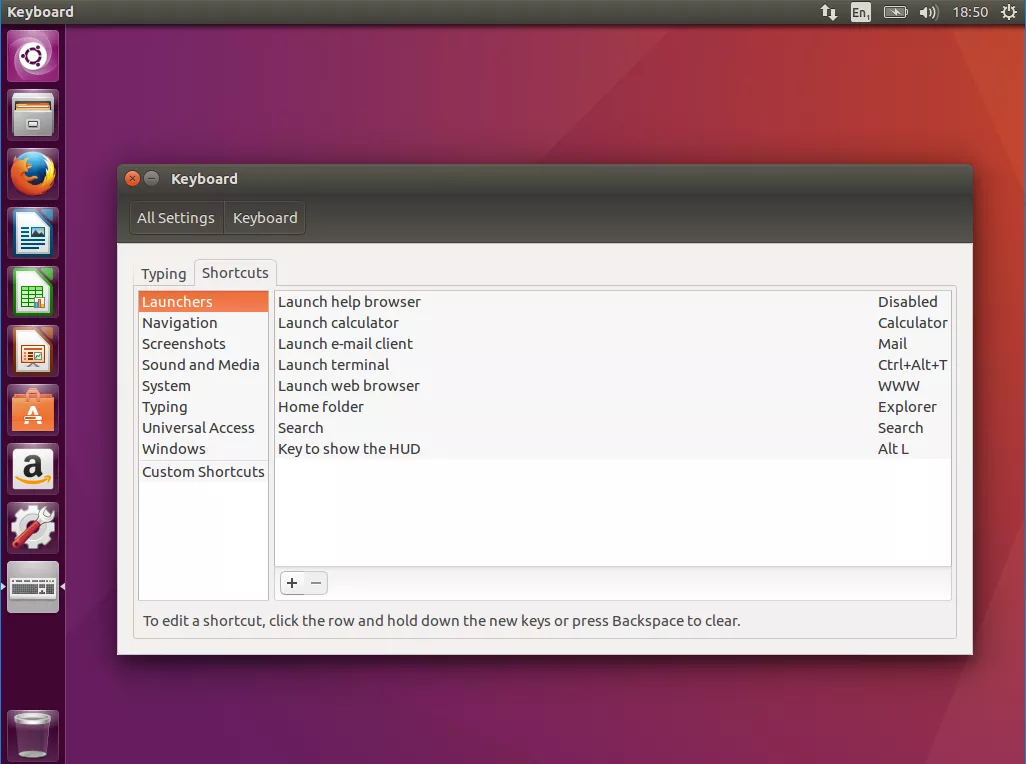
# Shortcut Keys

**Note**: To manage all shortcuts on the system, use Keyboard application. Open it by typing "Keyboard" on the Dash menu.



## Terminal

* Ctrl + Alt + T: Open a new Terminal shell.
* Ctrl + Shift + T: Open a new Terminal tab.
* Ctrl + L: Clean the Terminal's screen without clearing any commands.
* Ctrl + C: Close a program which is running on the Terminal shell.
* Ctrl + Shift + C: Copy text from the Terminal shell.
* Ctrl + Shift + V: Paste text to the Terminal shell.

## Windows

* Hide / show all windows: Ctrl + Super + D → Super + D
* Show all applications: Super + A
* Open the *Home* folder: Ctrl + H
* Switch between opening windows: Alt + Tab, or Super + Tab (*same as Windows*)
* Hide current opening window: Super + H (*same as Windows*)
* Move windows to the left/right: Super + Left/Right/Top/Bottom (*same as Windows*)
* Maximize / minimize current opening window: Super + Top/Bottom (*same as Windows*)

## Others

* Show the run command prompt: Alt + F
* Switch among input sources: Super + Space (*same as Windows*)
* Log screen: Super + L (*same as Windows*)
* Log out: Ctrl + Alt + Delete (*same as Windows*)

# Common Configurations

## Vietnamese Input

<https://blogchiasekienthuc.com/linux/cach-cai-dat-bo-go-tieng-viet-tren-ubuntu.html>

Override

## Make "rm" Move Files to Trash

<http://www.webupd8.org/2010/02/make-rm-move-files-to-trash-instead-of.html>

**Tip**: Emty Trash from CML: $ cd ~/.local/share/Trash; $ rm -rf files/

## Minimize Windows to Dock

To enable minimizing windows when clicking on its icon in the Dock, run:

$ gsettings set org.gnome.shell.extensions.dash-to-dock click-action 'minimize'

To set things back to default, run:

$ gsettings reset org.gnome.shell.extensions.dash-to-dock click-action

## Show Window Thumbnail from Dock

To enable showing window thumbnail when hovering over its icon in the Dock, run:

$ gsettings set org.gnome.shell.extensions.dash-to-dock middle-click-action 'previews'

**Note**: You have to middle click on the icon to see window thumbnail.

**Tip**: There are other options besides middle clicking, to see them all, run:

$ gsettings range org.gnome.shell.extensions.dash-to-dock click-action

## Show Date in Top Bar

By default, time info displayed on the top bar is only about hour, day of week and year. To also get the date info, run:

$ gsettings set org.gnome.desktop.interface clock-show-date true

## Add Display Resolution

Ideally, Linux can know all possible display resolutions it (and your monitor) can support. These resolution choices are listed in Settings > Devices > Displays > Resolution. But in some cases, it misses many. To add these resolutions manually, follow the instruction at <https://superuser.com/a/311671>.

## Disable Auto Screen Look

Run:

gsettings set org.gnome.desktop.lockdown disable-lock-screen 'true'

# To revert change:

# gsettings set org.gnome.desktop.lockdown disable-lock-screen 'false'

Note: The UI way does NOT work. (Settings -> Privacy -> Screen Lock -> and changed 'Automatic Screen Lock' to 'Off').

## Disable Screen Saver Locking

Screen saver is invoked after a certain period of inactivity. it requires input to get your desktop back.

Run:

gsettings set org.gnome.desktop.screensaver lock-enabled false

# To revert change:

# gsettings set org.gnome.desktop.screensaver lock-enabled true

## MS Office on Linux

**Way #1**: PlayOnLinux (based on Wine):

<https://www.makeuseof.com/tag/install-use-microsoft-office-linux/>

**Way #2**: Wine only

<https://help.ubuntu.com/community/Wine>

**Notes before installation (for both ways):**

* Wine will work better with **older versions** of applications (including MS Office), so the older the version of your app, the more likely it is to work without any trouble.
* **64-bit apps** are not really supported by Wine (more details [here](https://www.playonlinux.com/en/topic-12407-64bit.html)). That’s why it’s best to install MS Office x86 instead of x64.
* Wine **uses .exe file (not .iso file)** to install apps. So after downloading the .iso package, extract it to get the .exe file. When Wine asks for the setup file, choose the .exe file.
* To search apps supported by Wine, check <https://appdb.winehq.org/>.

## Customize Grub (Change Boot Sequence, Wait Time, etc.)

<http://www.linuxandubuntu.com/home/customizing-grub-the-easy-way-grub-customizer>

# Linux on Virtual Machines

See *Personal\Tutorials\Others\Virtual Machine Tutorial.docx*.

# Common Issues and Solutions

## <Username> is not in the sudoers file

<https://www.linuxuprising.com/2019/09/fix-username-is-not-in-sudoers-file.html> (tested on CentOS and Ubuntu).

# Commands

## Terminal Control

### clear (or Ctrl + L)

**Usage**: Clear the screen.

**Syntax**: clear

### reset

**Usage**: Re-initialize the terminal (the screen is also cleaned). Note that it won't re-instantiate the shell (bash) – so the bash's state doesn't change.

**Syntax**: reset

### nautilus

**Usage**: Open the UI explorer from the terminal.

**Syntax**: nautilus <any-directory>

## Command Helper

### --help

**Usage**: Show the guideline on how to use a particular command. This is in fact an option. Most commands in Linux have this option.

**Syntax**: <any-command> --help

### --version

**Usage**: Check the version of a package/command in Linux.

**Syntax**: <package-name> --version

**Example**: Use python --version to see the current version of Python installed in your Linux.

### man

**Usage**: View the system's reference manuals. It allows scroll up and down, search for occurrences of specific text, and other useful functions.

**Syntax**: man [options] <any-command>

More details: <https://www.computerhope.com/unix/uman.htm>

### info

**Usage**: Read documentation in the info format, giving detailed information for a command. The info pages are made using the *texinfo* tools because of which it can link with other pages, create menus and easy navigation.

**Syntax**: info [options] <any-command>

More details: <https://www.computerhope.com/unix/info.htm>

### history

**Usage**: View all commands we had used.

**Syntax**: history

**Tips:**

* Clear history: history -c

### alias

**Usage**: Instructs the shell to replace one string with another when executing commands.

**Syntax**: alias <name=['command-list']> in which:

* name specifies the alias name.
* command specifies the command the name will use as the alias.
* Note: There cannot be a space between the name and the '=' sign

**Examples:**

1. alias ls='ls --color=auto' --> Using the ls command will always display color output.

2. alias desk='cd Desktop; ls -la' --> Using the desk command will always go to Desktop directory, then list all subdirectories and files in the listing format.

**Note**:

If you create an alias from the Terminal, then it will create an alias temporarily. It works until you close your terminal.

To create an alias permanently, you need to add it to the ~/.bashrcfile.

Once you’ve done your editing, refresh the configuration by running: source ~/.bashrc. Or else, you have to re-open the Terminal.

**Tip**: Use unalias command to remove an alias.

More details:

<https://www.computerhope.com/unix/ualias.htm>

<https://www.computerhope.com/unix/uunalias.htm>

## Process Management

### shutdown (or reboot)

**Usage**: Shut the system down in a secure way. All logged-in users are notified that the system is going down, and login operations are blocked. It is possible to shut the system down immediately, or after a specified delay.

**Syntax**: shutdown [options] <time>

**Notes:** The time argument specifies when to perform the shutdown operation. It can be:

* now: Shuts the system down immediately.
* +m: Wait for some minutes.
* hh:mm: It is a format, in which, hh is the hour (1 or 2 digits, from 0 to 23) and mm is the minute of the hour (in two digits). This option allows you to schedule the system to shut down at a specific time.

**Common options**:

|  |  |
| --- | --- |
| -r | Reboot after shutdown. |
| -P | Instruct the system to shut down and then power down. |

### nohup

**Usage**: Stands for "no hangup." The hangup (HUP) signal, which is normally sent to a process to inform it that the user has logged off (or "hung up"), is intercepted by nohup, allowing the process to continue running after you log out or exit the shell.

**Syntax**: nohup <command>

**Notes**:

* If [standard input](https://www.computerhope.com/jargon/s/stdin.htm) is the Terminal, nohup redirects it from /dev/null. Therefore, terminal input is NOT possible when running a command with nohup.
* If [standard output](https://www.computerhope.com/jargon/s/stdout.htm) is the Terminal, command output is appended to the file nohup.out if possible, or $HOME/nohup.out otherwise.
* If [standard error](https://www.computerhope.com/jargon/s/stderr.htm) is a terminal, it is redirected to standard output.

**Tips**:

* The nohup command is usually used with the "[&](#_1fob9te)" operator (nohup <command> &). The "&" symbol at the end of any command instructs bash to run the command in the background. It can be brought back to the foreground with the fg command. Also note that when using "&", you'll see the bash job ID in brackets and the process ID (PID) listed after (e.g. [1] 25132).
* To kill the program which is running on the background, use the kill command.

More details:

<https://www.computerhope.com/unix/unohup.htm>

<https://www.youtube.com/watch?v=sqXZirjvo6c>

### ps

**Usage**: Stand for "process status". It reports a snapshot of the status of currently running processes.

**Syntax**:

* General use: ps [options]
* Display the information of a specific process: ps [options] <PID>

**Common options**:

|  |  |
| --- | --- |
| -A | List all processes |
| -a | List all processes (except ones not associated with the Terminal and session leaders) |
| -f | List processes with more detailed information |
| -u | List processes with user and start time |
| T | List all processes on this Terminal |
| r | List only running processes |

**Tips:**

* Display a tree of processes: pstree

More details:

<https://www.computerhope.com/unix/ups.htm>

[https://www.binarytides.com/Linux-ps-command/](https://www.binarytides.com/linux-ps-command/)

### kill

**Usage**: Send a signal to a process. If you don't specify which signal to send, by default the TERM signal is sent, which terminates the process. The kill command is mostly used after a ps command.

**Syntax**: kill [options] <PID>

**Common options**:

|  |  |
| --- | --- |
| -9  (or -SIGKILL) | Sends the SIGKILL signal to end a particularly unruly process. |
| -l | List all available signals without their descriptions |

**Other uses:**

* Kill a process using its **name** (instead of its PID): pkill <name-of-process)
* Kill a process and **all of its instances** and child processes: killall <name-of-process>. In fact, we use this command more often than the original kill.
* Kill a **job** (running processes where background or foreground): kill %<job-id>

References:

<https://www.computerhope.com/unix/ukill.htm>

[https://www.tecmint.com/how-to-kill-a-process-in-Linux/](https://www.tecmint.com/how-to-kill-a-process-in-linux/)

[https://www.linode.com/docs/tools-reference/tools/use-killall-and-kill-to-stop-processes-on-Linux/](https://www.linode.com/docs/tools-reference/tools/use-killall-and-kill-to-stop-processes-on-linux/)

### bg and fg

bg:

Press control + Z, which will pause it and send it to the background. Then enter bg to continue it's running in the background.

Alternatively, if you put a & at the end of the command to run it in the background from the start.

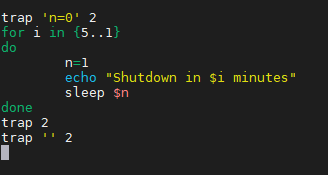
### jobs

List all processes running in background

### trap

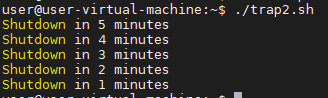
**Example 1:**

Nội dung file 'trap2.sh':

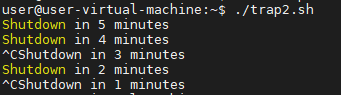


Kết quả chạy script:

Không nhấn nút DEL:

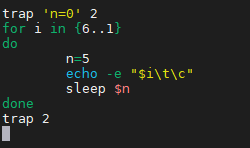


Có nhấn nút DEL:



**Example 2:**

**Nội dung file 'uncount.sh':**



Không nhấn nút DEL:



Có nhấn nút DEL:



## Task Managers

<http://www.linuxandubuntu.com/home/10-best-linux-task-managers>

### top

## Memory Managers

### free

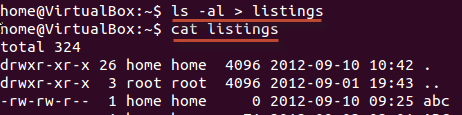
## Redirection Operator

### Output Redirection (>)

**Usage**: The > operator is used to send standard output to a file. To append content to an existing file instead of overriding it, use >> operator.

**Examples**:

1. The output of command ls -alis re-directed to the file listings instead of the screen



2. You can redirect standard output to not just files, but also devices!

$ cat music.mp3 > /dev/audio

The cat command reads the file music.mp3 and sends the output to /dev/audio which is the audio device. If the sound configurations in your PC are correct, this command will play the file music.mp3.

### Input Redirection (<)

**Usage**: The < operator is used to read standard input from a file.

### Error Output Redirection (2>)

**Usage**: The 2> operator is used to send standard error to a file. To append content to an existing file instead of overriding it, use 2>> operator.

**Example**: With the command $ myprogram 2> errorsfile, we are executing a program named myprogram while re-directing its error output to a file named errorfile. Thus, program output is not cluttered with errors.

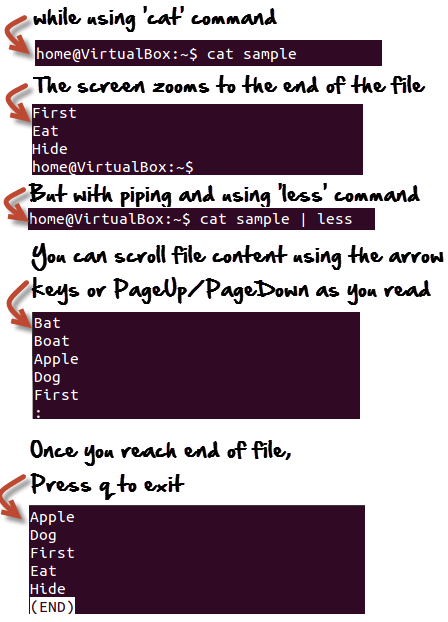
### Pipe Redirection (|)

**Usage**: The pipe (|) operator is used to redirect a stream from one program to another.

When a program's standard output is sent to another through a pipe, the first program's data, which is received by the second program, will not be displayed on the terminal. Only the filtered data returned by the second program will be displayed.

**Example**: cat filename | less

When you use cat command to view a file which spans multiple pages, the prompt quickly jumps to the last page of the file, and you do NOT see the content in the middle. To avoid this, you can pipe the output of the cat command to the less program which will show you only one scroll length of content at a time.



## Chaining Operators

### Semi-Colon Operator (;)

This operator makes it possible to **run several commands in a single line** and the execution of command occurs sequentially.

### Ampersand Operator (&)

This operator **allows the command to run in background**. Just add a '&' to the end of the command. Note that when you exit the shell, the operator will be terminated.

### AND Operator (&&)

This operator executes the second command only if the execution of the first command succeeds.

**Example**: $ /home/tecmint# ping -c3 www.tecmint.com && links [www.tecmint.com](http://www.tecmint.com)

-> Visit tecmint.com after checking if the host is live or not.

### OR Operator (||)

This operator executes the second command only if the execution of first command fails.

**Example**: $ apt-get update || links tecmint.com

-> Execute apt-get update from non-root account, and if the first command fails, then the second links www.tecmint.com command will execute.

### AND – OR Operator

The above operator is actually a combination of AND and OR operators.

**Example**: $ ping -c3 www.tecmint.com && echo "Verified" || echo "Host Down"

-> Ping to tecmint.com, if success echo Verified, else echo Host Down.

### NOT Operator (!)

This operator is like an 'except' statement. This command will execute all, except the condition provided.

**Example**: $ rm -r !(\*.html)

-> Delete all files in the current directory, except ‘html’ files all at once.

## Directories and Files

### ls

**Usage**: Show a list of names of files and subdirectories in a directory.

**Syntax**: ls [options] [dir]

**Common options**:

|  |  |
| --- | --- |
| -a | Show hidden files and directories (those starting with a ".") |
| -l | List files and directories line by line (also display the file size, modification time, permission info, etc.) |
| -h | Print sizes in human-readable format (with postfix K, M, G, T, etc.). |
| -S | Sort by file size. |
| -t | Sort by modification time, newest first. |
| -R | List subdirectories [recursively](https://www.computerhope.com/jargon/r/recursive.htm). |

**Example:**

**1. With option -l**

$ ls -l

total 10568

d rwx rwx r-x 2 triho triho 4096 Mar 13 22:56 dosdevices

d rwx rwx r-x 8 triho triho 4096 Mar 13 22:07 drive\_c

- rw- rw- r-- 1 triho triho 51 Mar 13 22:48 playonlinux.cfg

- rw- rw- r-- 1 triho triho 1222848 Mar 13 22:56 playonlinux.log

Where:

* total <num>: represent the total files in the directory and subdirectories.
* 1st column: represents whether the item is a file or directory (d for directory and - for file).
* 2nd – 4th columns: represent current modes of the file or directory when the reference is owner, group and others respectively. More details at section [chmod](#_chmod).
* 5th and 6th columns: represent the owner and group, respectively, of file or directory.
* 7th column: represents the size (in **bytes**) of the file or directory. Note that with the ls command, directories always has size 4096 (why [here](https://superuser.com/a/142900)). That’s why we use the su command to check size of directories in Linux.

**2. With option -h**

$ ls -lh

total 11M

drwxrwxr-x 2 triho triho 4.0K Mar 13 22:56 dosdevices

drwxrwxr-x 8 triho triho 4.0K Mar 13 22:07 drive\_c

-rw-rw-r-- 1 triho triho 51 Mar 13 22:48 playonlinux.cfg

-rw-rw-r-- 1 triho triho 1.2M Mar 13 22:56 playonlinux.log

**What do the different colors mean?**



More details:

<https://www.computerhope.com/unix/uls.htm>

<https://askubuntu.com/questions/17299/what-do-the-different-colors-mean-in-ls>

### cp

**Usage**: Make copies of files (by default) or directories (with option). Wildcards may be used to copy multiple files.

**Syntax**: cp [options] <file-or-dir> <destination-dir>

**Example**:

1. $ cp picture.jpg picture-02.jpg

-> Make a copy of picture.jpg, the new file is named as picture-02.jpg. Both files now exist in the same working directory.

2. $ cp /home/pictures/picture.jpg /home/backup

-> Make a copy of picture.jpg in /home/pictures. The new file is also named picture.jpg but located in /home/backup.

More details: <https://www.computerhope.com/unix/ucp.htm>

**Common options**:

|  |  |
| --- | --- |
| -R | Copy directories and their contents [recursively](https://www.computerhope.com/jargon/r/recursive.htm). |
| -v | Verbose mode; explain at all times what is being done. |
| -n, --no-clobber | Do NOT overwrite an existing file. |

### mv

**Usage**: Move or rename files and directories. Wildcards may be used to move multiple files.

**Syntax**: mv [options] <source-dir-or-file> <destination-dir-or-file>

**Examples**:

1. Move file myFile.txt from the current directory into /home/Downloads/: $ mv myFile.txt /home/Downloads/

2. Move all sub-directories and files in current directory into Downloads: $ mv \* /home/Downloads/

3. Rename directory temp to non-temp: $ mv /home/user/temp /home/user/non-temp

More details: <https://www.computerhope.com/unix/umv.htm>

### rm

**Usage**: Delete files (by default) or directories (with option). Wildcards may be used to remove multiple files.

**Syntax**: rm [options] <file-or-dir>

**Common options**:

|  |  |
| --- | --- |
| -r | Remove directories and their contents [recursively](https://www.computerhope.com/jargon/r/recursive.htm). |
| -d | Remove empty directories. Note: rm -d is equivalent to using [rmdir](https://www.computerhope.com/unix/urmdir.htm). |
| \* | Remove all files in the working directory. If it is write-protected, you will be prompted before rm removes it. |
| -v | Verbose mode; explain at all times what is being done. |

**Examples**:

1. Delete all files, except filename1 and filename2: rm !("filename1"|"filename2")

**Caution**: There is **NO Trash Can for rm**. So cannot recover the deleted files. For alternative for rm, but support Trash Can, check [here](https://unix.stackexchange.com/a/10884) or [here](https://superuser.com/a/324132).

More details:

<https://www.computerhope.com/unix/urm.htm>

<https://www.tecmint.com/delete-all-files-in-directory-except-one-few-file-extensions/>

### chmod

**Usage**: Standing for 'change mode', it changes the permissions of files or directories.

**Syntax**:

chmod [reference]=[mode] <file-or-dir>

chmod [reference][operator][mode] <file-or-dir>

Where:

* reference:

|  |  |  |
| --- | --- | --- |
| Reference | Class | Description |
| u | owner | file's owner |
| g | group | users who are members of the file's group |
| o | others | users who are neither the file's owner nor members of the file's group |
| a | all | anyone |

* operator:

|  |  |
| --- | --- |
| Operator | Description |
| + | Adds the specified modes to the specified classes |
| - | Removes the specified modes from the specified classes |
| = | The modes specified are to be made the exact modes for the specified classes |

* mode:

|  |  |
| --- | --- |
| Mode | Description |
| r | Permission to read the file. |
| w | Permission to write (or delete) the file. |
| x | Permission to execute the file, or in the case of a directory, search it. |

**Common options**:

|  |  |
| --- | --- |
| -R | Change mode of directories and files [recursively](https://www.computerhope.com/jargon/r/recursive.htm). |
| -v | Verbose mode; explain at all times what is being done. |

**Example:**

Assume your directory has following items (using ls -l):

- rwx r-x r-- mik man code.c

d rwx rwx r-x mik man EXAM

Where:

* 2nd – 4th columns: represent current modes of the file or directory when the reference is owner, group and others respectively. For example: The file code.c has owner’s permission as rwx (meaning the owner mik read, write and execute the file), group’s permission as rx- (meaning the group man can only read and execute the file) and others’ permission as r-- (meaning others can only read it).

Note that when a directory has the x set, this takes the special meaning of “permitted to search this directory”.

Now let change code.c’s mode so that the group man can now write the file, and others can now execute it:

# Before: - rwx r-x r-- mik man code.c

$ chmod g=rwx,o=x code.c

# After: - rwx rwx r-x mik man code.c

**Tip:**

We can use number to change mode of code.c as follows:

$ chmod 775 code.c

Here the digits 7, 7 and 5 represent the permissions for the user, group, and others respectively. Each of these digits is the sum of following numbers:

- 4 stands for "read"

- 2 stands for "write"

- 1 stands for "execute"

- 0 stands for "no permission"

So 7=4+2+1 (read, write, and execute) and 5=4+0+1 (read, no write, and execute).

Other common chmod number:

* 600 – owner can read and write
* 700 – owner can read, write and execute
* 666 – all can read and write
* 777 – all can read, write and execute

More details: <https://www.computerhope.com/unix/uchmod.htm>

### tree

**Usage**: List the contents of directories and files in a tree-like format. It's a really neat and useful to view the structure of directories.

**Syntax**: tree [options] [dir]

**Common options**:

|  |  |
| --- | --- |
| -d | List directories only |
| -a | List all directories and files (including hidden directories and files). |

More details: <https://www.computerhope.com/unix/tree.htm>

## Directories

### pwd

**Usage**: Print the full pathname of the working directory.

**Syntax**: pwd [options]

More details: <https://www.computerhope.com/unix/upwd.htm>

### cd

**Usage**: Stand for "change directory". It changes the shell's current working directory.

**Syntax**: cd [options] <dir>

**Common options**:

|  |  |
| --- | --- |
| ~ | Return you to the home directory |
| .. | Return you to the parent directory |
| - | Return you to the previous directory |
| / | Return you to the root directory (the first directory in your filesystem hierarchy. All other directories are sub-directories of the root directory) |
| -P | Use the physical directory structure without following symbolic links. In other words, only change into the specified directory if it actually exists as named; [symbolic links](https://www.computerhope.com/jargon/s/symblink.htm) will not be followed. This option is the opposite of the -L option, and if they are both specified, this option will be ignored. |

More details: <https://www.computerhope.com/unix/ucd.htm>

### du

**Usage**: Standing for "disk usage". It's usually used to display size of directories.

**Syntax**: du [options] <dir>

**Common options**:

|  |  |
| --- | --- |
| -s | Display only a summary for each argument. |
| -h | Print sizes in human-readable format (with postfix K, M, G, T, etc.).  By default, du prints size in **kilobytes** format. |

**Example**:

# current directory

$ du -sh

2.4G

$ du -sh drive\_c/

121M drive\_c/

**Note**: With du, the file size is always 4 (or 4.0K). So do not use this command to check files’ size (use ls -lh instead).

### mkdir

**Usage**: Create a new directory.

**Syntax**: mkdir [options] <dir>

**Common options**:

|  |  |
| --- | --- |
| -p | Create parent directories if necessary. For example:  # Without this option:  $ ls -a  . ..  $ mkdir dir/subdir  cannot create 'dir/subdir': no such file or directory  # With this option:  $ mkdir dir/subdir |

**Example**:

1. Create a new directory called myfile in the current directory: mkdir myfile

2. Create a new directory called myfile in the home directory: mkdir ~/myfile

More details: <https://www.computerhope.com/unix/umkdir.htm>

## Files

### cat

**Usage**: Stand for "[catenate](https://www.computerhope.com/jargon/c/concaten.htm)." It can be used to:

* Display text files
* Create text files (used with >)
* Copy text files into a new document (used with >)
* Append the contents of a text file to the end of another text file

**Syntax**: cat [options] <file>

**Examples**:

1. Read the content in mytext.txt: cat mytext.txt

2. Create a txt file named mytext.txt: cat > mytext.txt. Once the file is created, you can edit it. To save the file, press *Ctrl + Z*.

3. Copy a text file: cat mytext.txt > newfile.txt.(another way is using [cp](#_3znysh7) command)

More details: <https://www.computerhope.com/unix/ucat.htm>

### tar

**Usage**: Create, maintain, modify, and extract files that are archived in the .tar, .gz or .zip format.

**Syntax**: tar [options] <tar-file>

**Examples**:

1. Extract the archive file archive.tar.gz: $ tar -xzvf archive.tar.gz
2. Create a new archive file named archive.tar holding dir1 folder: $ tar -cf archive.tar dir1
3. Create a new archive file named archive.tar holding file1 and file2: $ tar -cf archive.tar file1 file2
4. List content of archive.tar in verbose mode (more info): $ tar -tvf archive.tar

**Common options**:

|  |  |
| --- | --- |
| -x | Tells tar to **extract** files from an archive. |
| -c | Tells tar to **create** a new archive. |
| -t | Tells tar to **list content** of an archive. |
| -v | Operate verbosely. |
| -z | Tells tar to read/write archives through [gzip](https://www.computerhope.com/jargon/g/gzip.htm), allowing tar to directly operate on several kinds of compressed archives transparently.  This option **should be used, for example, when operating on files with the extension .tar.gz**. |
| -f | Tells tar that the next argument will be the name of the archive to operate on. |

More details: <https://www.computerhope.com/unix/utar.htm>

### unrar

**Usage**: Extract files that are archived in the .rar format.

**Syntax**:

1. List file content: $ unrar l <rar-file>
2. Extract file: $ unrar x <rar-file>

### whereis

**Usage**: Locate the binary, source, and manual page files for a command. It searches for files in a restricted set of locations:

* Binary file directories: /bin/, /sbin/, /usr/bin, /usr/sbin, usr/local/bin, usr/local/sbin, usr/local/, usr/share, etc.
* Man page directories: /usr/man/, /usr/share/man, etc.
* Library directories: /lib/, /usr/lib/, usr/local/lib, usr/include/, /usr/src/, etc.

**Syntax**: whereis [options] <command>

**Example**:

Search for binary, source, and manual page files of the command whereis:

How to find location of binary file using whereis

**Common options**:

|  |  |
| --- | --- |
| -b | Search only for binaries. |
| -m | Search only for manual pages. |
| -s | Search only for sources. |
| -l | See paths that whereis is designed to search in |

### which

**Usage**: Locate the executable file for a command in the directories specified by the environment variable PATH.

**Syntax**: which <file>

**Example**:

$ which ls

/bin/ls

**Note**: What is the difference between which and whereis? whereis searches for "possibly useful" files, while which only searches for executable file.

### locate

**Usage**: Search a file by its name. It can match parts of the file name. By default, it queries strings using Regex and in a case-sensitive manner.

**Syntax**: locate [options] <patterns>

**Note**: The locate command relies on a database called mlocate. This database needs to be updated (using sudo updatedb) regularly for the locate command to work efficiently.

**Common options**:

|  |  |
| --- | --- |
| -i | Ignore case sensitivity and show results for both uppercase and lowercase |
| -c | Display the count of all matching entries |
| -e | Avoid seeing results of files not present in your machine at the time of using the *locate* command. That’s because even when you have an updated mlocate database, the *locate* command still produces results of files whose physical copies are deleted (no longer **e**xist) from your system. |

More details:

[https://www.tecmint.com/Linux-locate-command-practical-examples/](https://www.tecmint.com/linux-locate-command-practical-examples/)

<https://www.computerhope.com/unix/ulocate.htm>

### file

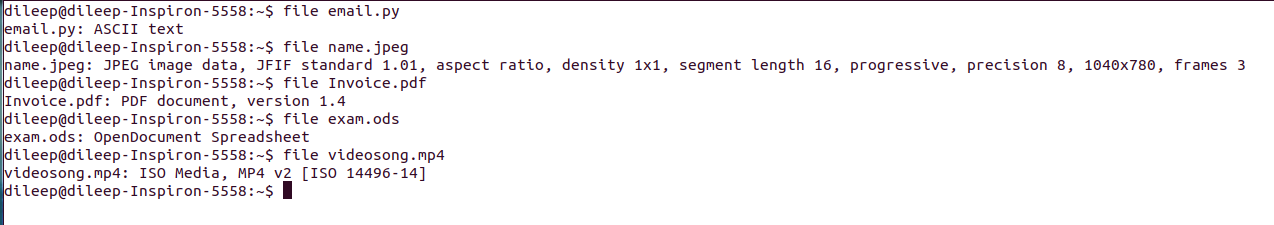
**Usage**: Reports a file's type.

**Syntax**: file [options] <file>

**Common options**:

|  |  |
| --- | --- |
| dir/\* | Displays the all files’ type in the directory 'dir'. |

**Example**:



## Text Manipulation

### vi

**Usage**: vi is an interactive text editor that is display-oriented: the screen of your terminal acts as a window into the file you are editing.

**Syntax**: vi <file>

**Example**: Edit a txt file named mytext.txt: vi mytext.txt.

**Notes**:

* The screen will clear and the text of your file will appear on the screen.
* If the file doesn't exist yet, vi will start you in a new file.

**Tips**: After editing the file, you can press[Esc] key anytime to return to command mode.

* To save file without exiting, press [Esc] key and type :w.
* To save and exit, press [Esc] key and type :wq.
* To exit file without saving, press [Esc] key and type :q!.

More details: <https://www.computerhope.com/unix/uvi.htm>

### sed

**Usage**: Stand for "stream editor". It allows you to filter and transform text.

**Syntax**: sed [options] <script> <file>

**Common options**:

|  |  |
| --- | --- |
| -i | Edit files in place to save back to the original file (without -i, you cannot save your file after editing) |

**Examples**:

1. Replacing words or characters: sed -i 's/original/new/g' file.txt

In which:

* s is the substitute command
* original is the text you want to replace
* new is the text to replace it with
* g is the global flag (replace all, not just the first occurrence)

Note: To replace the whole word only, use sed -i 's/\boriginal\b/new/g'

More details:

<https://www.computerhope.com/unix/used.htm>

[https://www.tecmint.com/Linux-sed-command-tips-tricks/](https://www.tecmint.com/linux-sed-command-tips-tricks/)

### grep

**Usage**: Stand for "global regular expression print". It processes text line by line and prints any lines which match a specified pattern.

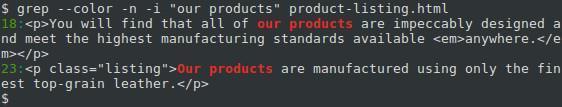
**Syntax**: grep [options] <patterns> <files>

**Common options**:

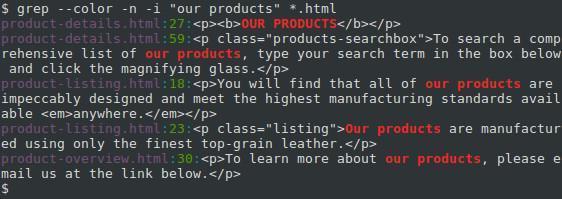
|  |  |
| --- | --- |
| -i | Ignore case sensitivity and show results for both uppercase and lowercase |
| -n | Print the line number of each matching |
| --color | Highlight each matching with a default color |
| -r | Extend our search to subdirectories and any files they contain. In this case, have to use wildcard “\*” the express the file name |
| -A <*NUM*> | Print *NUM* lines of trailing context *after* matching lines |
| -B <*NUM*> | Print *NUM* lines of trailing context *before* matching lines |
| -C <*NUM*> | Print *NUM* lines of trailing context *before and after* matching lines |

**Examples**:

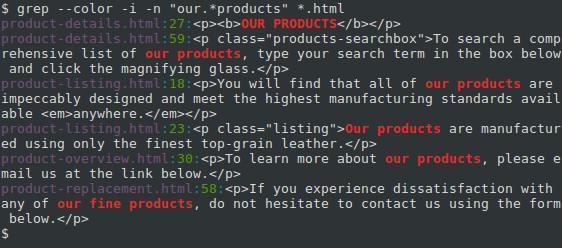
1. Without a [wildcard](https://www.computerhope.com/jargon/w/wildcard.htm) in the file name:



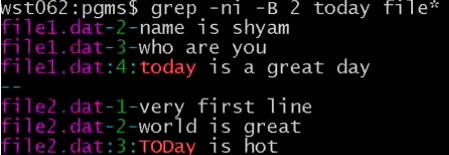
2. With a wildcard in the file name:



3. With a wildcard in the search pattern:



4. With -B option:



More details:

<https://www.computerhope.com/unix/ugrep.htm>

## Package Management

### For Debian and Ubuntu

#### apt

The apt tool is a software package manager commonly used to work with Debian-based system’s APT (Advanced Packaging Tool) library. It’s introduced in Ubuntu 16.04 as a more powerful and easier-to-use replacement for [apt-get](#_apt-get) and [apt-cache](#_apt-cache_(used_along).

**Syntax**: sudo apt <command> <package-name-list>

**Common commands**:

|  |  |
| --- | --- |
| install | Install package(s) |
| check-update | Check for available package updates |
| update | Update package(s) database (indexes are fetched from /etc/apt/sources.list)  Note: No <package-name-list> means updating the whole system. |
| upgrade | Install the newest versions of all packages currently installed on the system (using the source in /etc/apt/sources.list).  Note: Before an upgrade, an update should always be performed first so that apt-get knows which new versions of packages are available. |
| remove | Remove package(s) but leaves its configuration files on the system |
| purge | Remove package(s) and all of its configuration files on the system |
| search | Search for package(s) |
| show | Show package(s) details |
| list | List available, installed and upgradeable packages.  Can add option --installed or --upgradeable to filter the result. |

**Note**: Differences between apt update and apt upgrade:

Though it sounds like when you do an apt update, it will update the packages. But that’s not true! It only updates the database of the packages. For example, if you have XYX package version 1.3 installed, after apt update, the database will be aware that a newer version 1.4 is available.

Then when you run apt upgrade, it now upgrades (or updates, whichever term you prefer) the installed packages to version 1.4.

So the fastest way to update your system is: sudo apt update && # apt upgrade -y.

Before the birth of apt, we combine these tools to manage packages:

**apt-get**

**Usage**: The older and less powerful version of the apt tool.

**Syntax**: sudo apt-get <command> <package-name-list>

**Common commands**: Same as the apt tool. Except there are NO search, show and list.

**Note:** In addition to the lack of searching packages, listing packages and showing package details, the apt-get also requires to use additional command options to allow following features:

- A progress bar while installing or removing a program.

- The number of packages that can be upgraded when you update the repository database.

By contrast, the apt enables them by default.

**apt-cache**

**Usage**: Because the apt-get tool has no ability to search apt software packages, we use apt-cache to do that. In simple words, this tool is used to search software packages, find new packages, and collects information of packages.

**Syntax**: sudo apt-cache search <search-term>

**Common options**:

|  |  |
| --- | --- |
| -y | Assume the answer "yes" to any prompts. |
| -f | Attempt to fix any broken dependencies when used with install or remove, |

#### add-apt-repository

**Usage**: Add a repository into the /etc/apt/sources.list or removes an existing one.

**Syntax**: sudo add-apt-repository <ppa-repository>

**Why we need it?**

The command adds a PPA to your list of sources, so that Ubuntu knows to look for updates from that PPA as well as from the official Ubuntu sources. Usually, this is used to allow developers to provide updates more quickly than those in the official Ubuntu repositories.

**Example:**

sudo add-apt-repository ppa:maco.m/ruby

sudo apt-get update

sudo apt-get install rubygems

First, we update the source list by adding the PPA for ruby. Then, we run sudo apt-get update to tell update its database of what packages can be installed and where to install them from. In this case, apt-get will see your newly-added PPA and discover that ppa:maco.m/ruby has the newest version of rubygems that it knows about, so it will make a note to install rubygems from the PPA next time someone asks to install it using sudo apt-get install rubygems.

#### dpkg

**Usage**: Standing for Debian Package Manager, dpkg is the main package management program in Debian and Debian-based System. It is used to install, build, remove, and manage .deb packages.

**Syntax**:

1. Install a package: sudo dpkg -i <deb-file>
2. Install all packages from a directory: sudo dpkg -R --install <dir-to-deb-files>. This recursively installs all the regular files matching pattern "\*.deb" found at specified directories and all of its subdirectories.
3. List all the installed packages on your system: sudo dpkg -l
4. Remove the ".deb" package: sudo dpkg -r <package-name>. To remove configuration files along with the package, use option -p instead of -r.
5. Check a package is installed or not: sudo dpkg -s <package-name>
6. Check the location of packages installed: sudo dpkg -L <package-name>. This lists all files installed by the package.

### For Red Hat, CentOS and Fedora

#### dnf

dnf is a software package manager commonly used in RPM-based Linux distributions. Introduced in Fedora 18, it has become the default package manager since Fedora 22.

This package management is the next generation of [yum](https://fedoraproject.org/wiki/Yum). It roughly maintains compatibility with yum and defines a strict for extensions and plugins. As of Fedora 22, *yum* has been replaced with dnf and doesn't need to be install. But dnf can installed using yum: sudo yum install dnf.

**Syntax**: sudo dnf <command> <package-name-list>

**Common commands**:

|  |  |
| --- | --- |
| search | Search package details |
| install | Install package(s) |
| check-update | Check for available package updates |
| update | Update package(s) |
| reinstall | Reinstall package(s) |
| remove | Remove package(s) |
| history | Display the transaction history.  sudo dnf history displays all installation/update/remove events in the past |
| info | Display details about package(s) |

**Return values**:

* 0: Operation was successful.
* 1: An error occurred, which was handled by dnf.
* 3: An unknown unhandled error occurred during operation.
* 100: See check-update
* 200: There was a problem with acquiring or releasing of locks.

More details: <https://dnf.readthedocs.io/en/latest/command_ref.html>

Should read: <https://www.digitalocean.com/community/tutorials/package-management-basics-apt-yum-dnf-pkg>

#### rpm

rpm (RPM Package Manager) is a utility for installing software on Unix-like systems.

**Syntax**: sudo rpm [options] <rpm-file>

**Common options**:

|  |  |
| --- | --- |
| -i | Install package |
| -U | Upgrade package or reinstall package |
| -F | Upgrade package if already installed |
| -e | Erase (uninstall) package |

The following is an example of how to use rpm:

1. Log in as root, or use the su command to change to the root user at the workstation on which you want to install the software.
2. Download the package you want to install (something like DeathStar0\_42b.rpm).
3. To install the package, use: rpm -i DeathStar0\_42b.rpm.
4. If you are upgrading from an earlier version of the software package, run rpm in upgrade mode as: rpm -U DeathStar0\_42b.rpm.

## Download and Upload

### wget

**Usage**: Download files from the web.

**Syntax**: wget [options] <url(http\_or\_https\_or\_ftp)>

**Why wget**?

* It is non-interactive, meaning that it can **work in the background**, even while the user is not logged on, which allows you to start a retrieval and disconnect from the system, letting wget finish the work. By contrast, most web browsers require constant user interaction, which make transferring a lot of data difficult.
* It is designed for robustness over slow or unstable network connections. **If a download fails due to a network problem, it will keep retrying until the whole file has been retrieved**. If the server supports re-getting, it will instruct the server to continue the download from where it left off.

**Examples**:

1. Download file following the link <http://website.com/files/file.zip>:

wget http://website.com/files/file.zip

### curl

<https://www.geeksforgeeks.org/curl-command-in-linux-with-examples/>

<https://unix.stackexchange.com/questions/47434/what-is-the-difference-between-curl-and-wget>

## User Management

### Root User

The root user is basically equivalent to the administrator on Windows – the root user has maximum permissions and can do anything to the system. To do something that requires these permissions, you’ll have to acquire them with sudo or su.

Whenever a user tries to install, remove and change any piece of software, he/she has to have the root privileges to perform such tasks.

#### sudo

**Usage**: Stand for "superuser do". It gives users the temporary root privileges to perform some particular tasks.

**Syntax**: sudo [options] <any-command>

**Common** **options**:

|  |  |
| --- | --- |
| –b | Run the command in the background. |
| –s | Same as su (don’t forget to exit). |

More details: <https://www.computerhope.com/unix/sudo.htm>

#### su

**Usage**: Stand for switch user. It allows to switch to the root user and execute several commands without having to specify sudo at the beginning of each command.

Once you’re done running commands in the root shell, you should run exit to leave the root shell and go back to limited-privileges mode.

**Example**:

[triho@localhost opt]$ su

Password:

[root@localhost opt]# exit

[triho@localhost opt]$

Note: The # symbol indicates that you're now the root user

Difference between su, sudo -s, sudo -i and sudo -su**:** <https://askubuntu.com/questions/70534/what-are-the-differences-between-su-sudo-s-sudo-i-sudo-su>

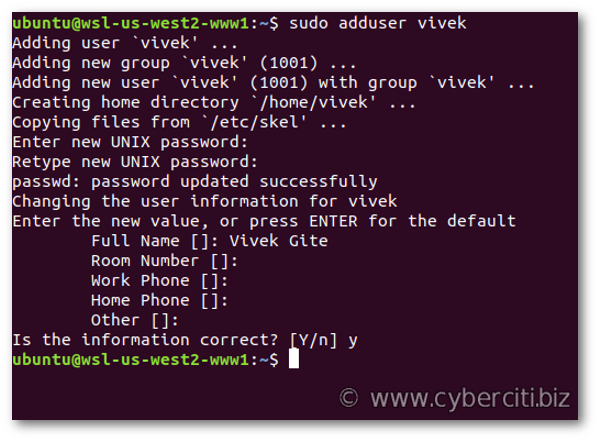
### Add New User

**Method #1:**

To add a new user called "vivek", run the following adduser command:

$ sudo adduser vivek

Then type password and other info:



**Method #2:**

The useradd command is a low-level utility for the adduser command. The syntax is:

$ sudo useradd -s /path/to/shell -d /home/{dirname} -m -G {secondary-group} {username}

$ sudo passwd {username}

For example, to create a new user named "vivek", run:

$ sudo useradd -s /bin/bash -d /home/vivek/ -m -G sudo vivek

$ sudo passwd vivek

Where,

* -s /bin/bash – Set /bin/bash as login shell of the new account
* -d /home/vivek/ – Set /home/vivek/ as home directory of the new Ubuntu account
* -m – Create the user’s home directory
* -G sudo – Make sure vivek user can sudo i.e. give admin access to the new account

### Change User Password

To change password of <username>, run the following passwd command:

sudo passwd <username>

First, the user is prompted for their current password. If the current password is correctly typed, a new password is requested. The new password must be entered twice to avoid password mismatch errors.

### Delete User

To delete user <username>, run the following userdel command:

sudo userdel <username>

To remove home directory and mail spool too, run:

sudo userdel -r <username>

### Verify All Users

All user info is stored in the passwd file, open it:

$ cat /etc/passwd

For example:

$ cat /etc/passwd

root:x:0:0:root:/root:/bin/bash

daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin

bin:x:2:2:bin:/bin:/usr/sbin/nologin

sys:x:3:3:sys:/dev:/usr/sbin/nologin

sync:x:4:65534:sync:/bin:/bin/sync

games:x:5:60:games:/usr/games:/usr/sbin/nologin

man:x:6:12:man:/var/cache/man:/usr/sbin/nologin

lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin

mail:x:8:8:mail:/var/mail:/usr/sbin/nologin

news:x:9:9:news:/var/spool/news:/usr/sbin/nologin

uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin

proxy:x:13:13:proxy:/bin:/usr/sbin/nologin

........

There are seven fields delimited by colons that contain the following:

* User name
* Encrypted password (x means that the **password is stored in the /etc/shadow file**)
* User ID number (UID)
* User’s group ID number (GID)
* Full name of the user (GECOS)
* User home directory
* Login shell (defaults to /bin/bash)

## Wildcard

Wildcards (also referred to as meta characters) are symbols or special characters that represent other characters. It is an extremely useful command which helps you work with Linux in a much faster way.

You can use wildcards with any command, such as ls command or rm command to list or remove files matching a given criteria, receptively.

More details:

[https://www.tecmint.com/use-wildcards-to-match-filenames-in-Linux/](https://www.tecmint.com/use-wildcards-to-match-filenames-in-linux/)

## Network

### ifconfig

**Usage**: Stands for "interface configuration". It is used to display information about all network interfaces currently in operation.

More details: <https://www.computerhope.com/unix/uifconfi.htm>

### ping

**Usage**: Check whether the network connection is available or not

**Example**: Use ping -c3 [www.google.com](http://www.google.com/) to check if the computer can access to Google.

### SSH Protocol

This is a protocol used to share folders/files between different computers in the same local network.

Steps:

1. On the server (the computer containing folders/files you want to access), run: sudo apt-get install openssh-server
2. On the client, run: sudo apt-get install openssh-client
3. On the client, go to File -> Connect to Server, then enter sftp://<IP-of-the-server>
4. Enter the username and password of the server

More details:

<https://askubuntu.com/questions/156169/how-do-i-set-up-file-sharing-between-two-ubuntu-laptops-on-my-wireless-network>

# Linux Architecture

## Components

### Kernel

It is the core part of Linux, responsible for all major activities of this OS. It consists of various low-level modules and it interacts directly with the underlying hardware.

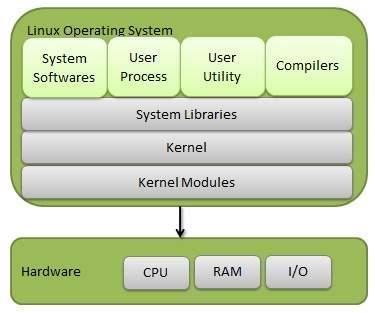
### System Library

It provides special **libraries to access kernel's features**. These libraries implement most functionalities of the OS and do not requires kernel module's code access permissions.

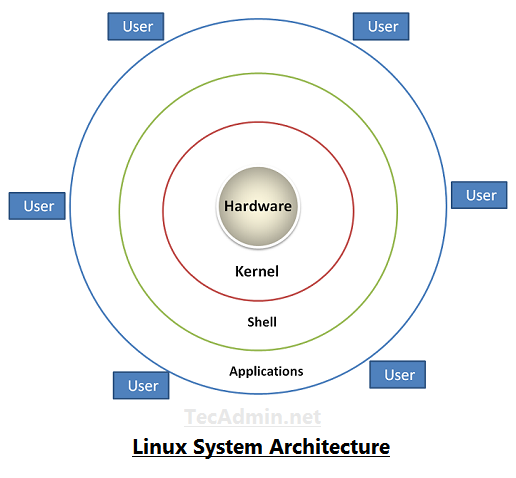
The term "library" is usually used to describe a collection of implementations of behavior written in terms of a language. It contains a well-defined interface by which the behavior is invoked. It means people who want to make a higher-level program can use the library to make system calls over and over again. The library can be requested by multiple individual programs simultaneously, so that the library has been coded in such a way that multiple programs can use the library even though the programs have no connection to each other.

### System Utility

The system utility software focuses on **how the computer infrastructure (including the hardware, OS, application and data storage) operates**. It, along with the OS, is used to support the computer infrastructure. It's very different from application software which is aimed at directly performing tasks that benefit ordinary users.



## Architecture



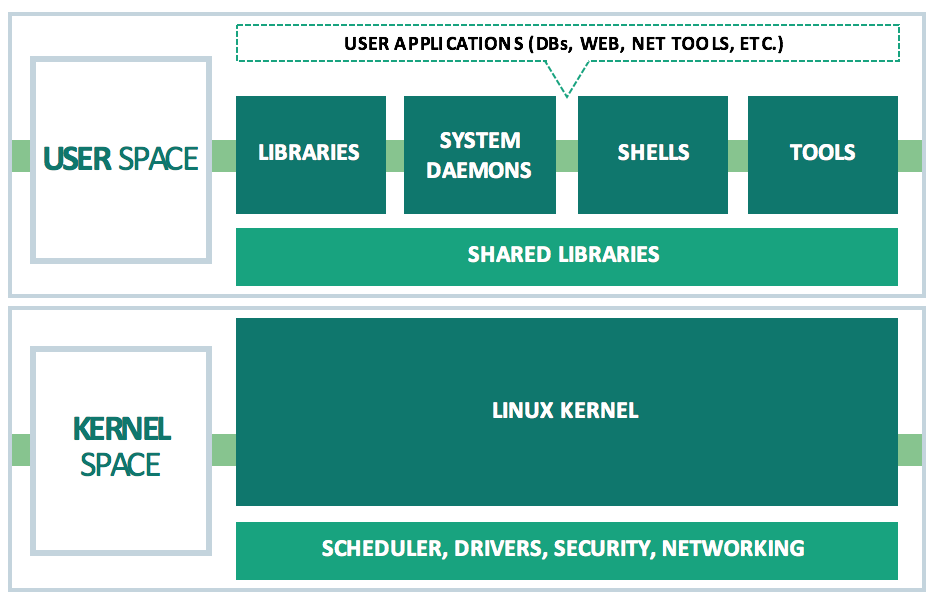
* **Hardware** − Hardware consists of all physical devices attached to the system. For example: hard disk, RAM, motherboard, CPU, etc.
* **Kernel** – It is the core part of Linux.
* **Shell** – It is an interface between the user and the kernel, hiding complexity of kernel's functions from the user. It takes commands from the user and executes kernel’s functions. The shell is present in two main types: **command-line shells** and **graphical shells**.
* **Applications** − They are programs which runs on shell. For example: web browser, media player, text editor, etc.

# User Space vs. Kernel Space

It’s important to know that OSs all execute their kernel in protected and restricted memory called kernel space to prevent the kernel from terminating and crashing the system.

When a user runs an application, that application is executed in another memory called user space (also called *userland*).

This distinction is critical. Applications can come from a variety of sources, may be poorly developed or originate unknown sources. By running these separately from the kernel space, they can’t tamper with the kernel resources and cause the system to crash.



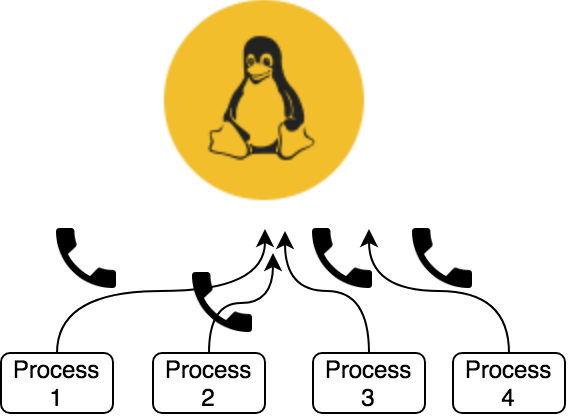
All applications, even system daemon processes which perform critical OS functions, must make what is called a [system call](#_4d34og8) to the kernel in the kernel space in order to access system resources such as memory or network devices. Every modern multi-user OS has some type of user space versus kernel space design, which is intended to keep it secure, high-performing and reliable.

# System Call

## What Are System Calls?

When you run a program which calls open, fork, read, write (and many others) you are making a system call.

System calls are how a program or application enters the kernel to perform some special operations, such as creating processes, doing network and file IO, etc. For a full list of system calls in Linux, check the [man page for syscalls(2)](http://man7.org/linux/man-pages/man2/syscalls.2.html).



System calls are the only entry points into the kernel system. Using them is a must for all programs needing kernel resources.

## Services Provided by System Calls

* Process creation and management
* Main memory management
* File access, directory and file system management
* Device handling (I/O)
* Protection
* Networking
* Etc.

## Types of System Calls

* Process control: end, abort, create, terminate, allocate and free memory.
* File management: create, open, close, delete, read file etc.
* Device management
* Information maintenance
* Communication

## Examples of Windows and Unix System Calls

|  |  |  |
| --- | --- | --- |
|  | **Windows** | **Unix** |
| Process control | CreateProcess() ExitProcess() WaitForSingleObject() | fork() exit() wait() |
| File manipulation | CreateFile() ReadFile() WriteFile() CloseHandle() | open() read() write() close() |
| Device manipulation | SetConsoleMode() ReadConsole() WriteConsole() | ioctl() read() write() |
| Information maintenance | GetCurrentProcessID() SetTimer() Sleep() | getpid() alarm() sleep() |
| Communication | CreatePipe() CreateFileMapping() MapViewOfFile() | pipe() shmget() mmap() |
| Protection | SetFileSecurity() InitlializeSecurityDescriptor() SetSecurityDescriptorGroup() | chmod() umask() chown() |

## System Call vs. Interrupt

[https://www.slashroot.in/what-is-system-call-in-unix-and-Linux](https://www.slashroot.in/what-is-system-call-in-unix-and-linux)

## How Does A Program Make System Call?

[https://www.slashroot.in/what-is-system-call-in-unix-and-Linux](https://www.slashroot.in/what-is-system-call-in-unix-and-linux)

## System Calling with GNU C Library (*glibc*)

More details: [https://blog.packagecloud.io/eng/2016/04/05/the-definitive-guide-to-Linux-system-calls/#user-programs-the-kernel-and-cpu-privilege-levels](https://blog.packagecloud.io/eng/2016/04/05/the-definitive-guide-to-linux-system-calls/#user-programs-the-kernel-and-cpu-privilege-levels)

# Kernel Mode vs. User Mode

**Kernel Mode**

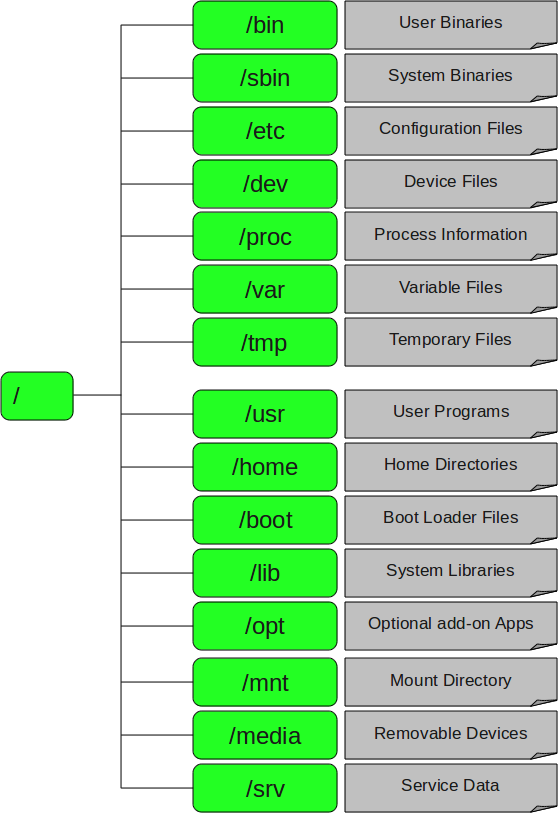
In kernel mode, the executing code has unrestricted access to the underlying hardware, meaning it has **full access to all resources of the computer**. It can execute any CPU instruction and reference any memory address. Kernel mode is generally reserved for the lowest-level, most trusted functions of the OS. Crashes in kernel mode are catastrophic that can halt the entire PC.

**User Mode**

In user mode, the executing code has no ability to directly access the underlying hardware and kernel code. It **must delegate to system libraries to access hardware or memory**. Due to the protection afforded by this sort of isolation, crashes in user mode are always recoverable – Instead of the entire system crashing, only that particular application crashes. Most of the code running on your computer will execute in user mode.

**Note**: The kernel runs at the most privileged level, called "**Ring 0**". User programs run at a lesser level, typically "**Ring 3**".

# Folder Structures



**1. / – Root**

* Every single file and directory starts from the root directory.
* Only root user has writing privilege under this directory.
* Please note that /root is root user’s home directory, which is not same as /.

**2. /bin – User Binaries**

* Contains binary executables.
* Provide common Linux commands you usually use in the user mode.
* For examples: ps, ls, ping, grep, cp.

**3. /sbin – System Binaries**

* Contains binary executables, but used typically by system administrator.
* For examples: iptables, reboot, fdisk, ifconfig, swapon.

**4. /etc – Configuration Files**

* Contains configuration files required by all programs.
* Also contains startup and shutdown shell scripts used to start/stop individual programs.
* For examples: /etc/resolv.conf, /etc/logrotate.conf.

**5. /dev – Device Files**

* Contains device files, including terminal devices, USB, or any device attached to the system.
* For example: /dev/tty1, /dev/usbmon0.

**6. /proc – Process Information**

* Contains information about system process.
* This is a pseudo filesystem contains information about running process. For example: /proc/{pid} directory contains information about the process with that particular pid.
* This is a virtual filesystem with text information about system resources. For example: /proc/uptime.

**7. /var – Variable Files**

* Stands for variable files.
* Contains content of files expected to grow. This includes — system log files (/var/log), packages and database files (/var/lib), emails (/var/mail), print queues (/var/spool), lock files (/var/lock), temp files needed across reboots (/var/tmp).

**8. /tmp – Temporary Files**

* Contains temporary files created by system and users.
* Files under this directory are deleted when system is rebooted.

**9. /usr – User Programs**

* Contains binaries, libraries, documentation, and source-code for second level programs.
* /usr/bin contains binary files for user programs. If you can’t find a user binary under /bin, look under this. For example: at, awk, cc, less, scp.
* /usr/sbin contains binary files for system administrators. If you can’t find a system binary under /sbin, look under this. For example: atd, cron, sshd, useradd, userdel.
* /usr/lib contains libraries for /usr/bin and /usr/sbin.
* /usr/local contains user programs that you install from source. For example, when you install apache from source, it goes under /usr/local/apache2.

**10. /home – Home Directories**

* Home directories for all users to store their personal files.
* For example: /home/john, /home/nikita.

**11. /boot – Boot Loader Files**

* Contains bootloader related files, such as Kernel initrd, vmLinux, grub files.
* For examples: initrd.img-2.6.32-24-generic, vmlinuz-2.6.32-24-generic.

**12. /lib – System Libraries**

* Contains library files that supports the binaries located under /bin and /sbin.
* Library filenames are either ld\* or lib\*.so.\*.
* For examples: ld-2.11.1.so, libncurses.so.5.7.

**13. /opt – Optional add-on Applications**

* Stands for optional.
* Contains add-on applications from individual vendors.
* Add-on applications should be installed under either /opt/ or /opt/ sub-directory.

**14. /mnt – Mount Directory**

* Temporary mount directory where sysadmins can mount filesystems.

**15. /media – Removable Media Devices**

* Temporary mount directory for removable devices.
* For examples: /media/cdrom for CD-ROM, /media/floppy for floppy drives, /media/cdrecorder for CD writer.

**16. /srv – Service Data**

* Stands for service.
* Contains server specific services related data.
* For examples: /srv/cvs contains CVS related data.

# Filesystem

## What Is Filesytem?

Filesystem is a bit of software that **specifies how information is laid out in storage** which is about things like:

* Where each file is located on the disk?
* How fast files can be read or written?
* Whether files get messed up if a disk loses power while being written?
* How long file names can be?
* What letters are permitted in file names?
* How tightly files are packed together
* What sort of data about files is stored (who owns it, who can access it, etc.)?
* Etc.

## Why Does Every OS Need Filesystem?

You probably noticed that **files are organized in a tree-like structure**: one root (say C:/ on Windows) and inside that folder you can add other folders (say Program Files, Windows, etc. on Windows) and files. These other folders can also have sub-folders and files themselves, in the end forming a hierarchical tree structure. **But the disk does not have a hierarchical structure**, so someone must figure out where the file C:/Folder1/File1.txt actually is on the disk.

The filesystem will play that "someone" role. Without it, you wouldn't know how to find a file in your PC. So, the filesystem contains a collection of methods and data structures (considering them as rules, and each OS has its own rules) used by the OS to organize and keep track of files on the disk.

Among other information, these data structures contain mappings between the logical name/location in the hierarchical structure and their location on the physical disk.

## How Does Filesystem Work?

You can view the disk as a really long series of equally-sized blocks arranged in a single sequence:

[ 1 ] [ 2 ] [ 3 ] [ 4 ] [ 5 ] [ 6 ] ...

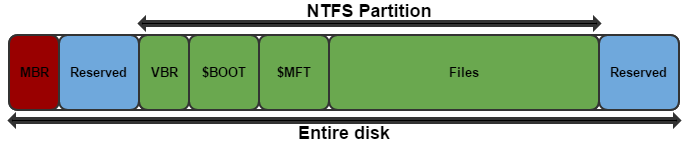
Each file on the disk is stored in at least one of these blocks and the OS must know which blocks belong to each file. For example, a short text file might need only one block and the OS may assign block 1 to it. If you download a video from the Internet, it might need more than one block and may be assigned blocks 2, 3 and 4.

The way this information is stored and the way the blocks are assigned depends on how the filesystem is implemented.

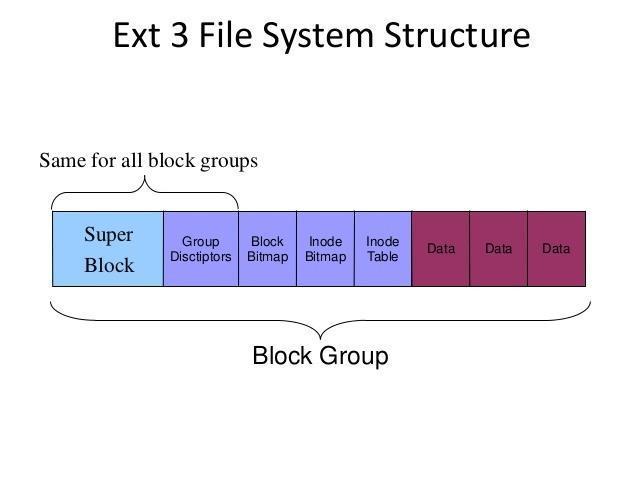
But in reality, thing might not that simple. For example, if the OS assign blocks 2, 3 and 5 for the video file because the block 4 is used by another file, what happens if you delete a file? You will have to use that space again, but how do you deal with the fact that you now have a gap? If the file has more than one block, how do you decide what blocks to give to the file? If you assign the blocks consecutively, how do you store a big file if you have many small gaps? If you assign blocks that are far apart from each other, how will you deal with the fact that it takes much longer to find them?

This gap problem is called *fragmentation* and is only one of the problems you have to think about when designing a file system. People tried to find different ways to work around this problem, and that results in the wide selection of filesystems available today.

But it is not as simple as that, because you cannot objectively say that one file system is better than the other unless you are comparing a modern filesystem with an old one. This is not a problem for general use, but if you use your computer in a very specialized way, one filesystem may have significant advantages over the rest.



*NTFS filesystem (on Windows)*



*Ext3 filesystem (in Linux)*

## Different Types of Filesystems in Linux

Linux supports many (up to ~100 types of filesystems, eg: ext2, ext3, ext4, XFS, JFS, btrfs, etc.), but Windows supports very few (FAT16, FAT32 and NTFS). Each can be used for a specific purpose.

The following is a list of common filesystems currently supported by most Linux distributions:

<https://www.howtogeek.com/howto/33552/htg-explains-which-linux-file-system-should-you-choose/>

# Mounting and Unmounting

## What Is Mounting and Unmounting in Linux?

Unix systems have a single directory tree where / is the one and only root directory. All accessible storage must have an associated location in this single directory tree. This is unlike Windows where there is one directory tree per storage component (drive).

Before your computer can use any kind of storage device (hard drive, CD-ROM, network share, etc.), your operating system must make it accessible through the computer's file system. This process is called *mounting*.

In particular, mounting is the act of associating a storage device to a particular location in the directory tree (and you **can only access files on mounted media**). For example, when the system boots, a particular storage device (commonly called the root partition) is associated with the root of the directory tree, i.e., that storage device is mounted on /.

Let's say you want to access files on a CD-ROM. You must mount the CD-ROM on a location in the directory tree (this is usually done automatically by the OS when you insert the CD). Let's say the CD-ROM device is /dev/cdrom and the chosen mount point is /media/cdrom. The corresponding command is:

mount /dev/cdrom /media/cdrom

After that command is run, a file whose location on the CD-ROM is /dir/file is now accessible on your system as /media/cdrom/dir/file. When you've finished using the CD, you run the command:

umount /dev/cdrom

umount /media/cdrom

(both will work; typical desktop environments will do this when you click on the "eject" or "safely remove" button).

## How to Mount and Unmount Filesystems?

<https://linuxize.com/post/how-to-mount-and-unmount-file-systems-in-linux/>

# Processes

## What Is A Process?

A process refers to a program/command in execution; it’s a **running instance of a program/command**. This instance consists of all the resources/services utilized by the process under execution.

Whenever a program/command is issued in Linux, it creates (and then starts) a new process. For example, when a pwd command is issued, a process starts; after it finished printing the current directory location, the process ends.

## What Is Process ID?

Process ID (or PID) is a 5-digit number Linux uses to keep track of a process. Each process in the system has a unique PID.

## Types of Processes

There are fundamentally two types of processes in Linux:

### Foreground Processes

Also referring to *interactive processes*, foreground processes are **initialized and controlled through a terminal session**. In other words, there has to be a user connected to the system to start such processes; they haven’t started automatically as part of the system functions/services.

Every process when started runs in foreground by default, receiving input from the keyboard and sending output to the screen.

For example:

$ pwd

Output:

/home/user/root

When a process is running in the foreground and is taking a lot of time, no other processes can be started because **the prompt would not be available until the program finishes processing and comes out**.

### Background Processes

Also referring to *non-interactive/automatic processes*, background processes are not connected to a terminal, and they **don’t expect any user input**.

**Other processes can be done in parallel with background processes** since they do not have to wait for the previous process to be completed.

For example:

$ pwd &

Output:

[1] 3244

$ /home/user/root

That first line contains information about the background process – the job number and the PID.

Since pwd does not wants any input from the keyboard, it goes to the stop state until moved to the foreground and given any data input. Thus, on pressing Enter again:

Output:

[1] + Done pwd

$

It tells you that the background process finishes successfully. Then it prompts for another command.

### Others

#### Parent and Child Processes

Parent processes **create other processes** during run-time. In Linux, every process has a parent except the init process and a few others.

Child processes **are created by other processes** during run-time. Child processes belonging to a parent are terminated with the parent. In Linux, all processes except init are children of a process. When you kill a child, the parent doesn't terminate.

#### Zombie and Orphaned Process

**Zombie**

When a child process exits, the parent process must wait on it to get its exit code. That exit code is stored in the process table. The act of reading that exit code is called "reaping" the child. Between that time (after the child exits and before it is reaped), it is called a *zombie*.

Zombies only occupy space in the process table. They take no memory or CPU. However, the process table is a finite resource, and excessive zombies can fill it, meaning that no other processes can launch. Beyond that, they are bothersome clutter, and should be strongly avoided.

**Orphan**

Now consider what would happen if a parent is terminated (or killed) and didn't wait to get its child exit code, leaving its child as *orphan*. These orphaned processes are immediately "adopted" by the init processes. The init process periodically wait, thereby allowing the exit code of any orphan to be collected and releasing the orphan identifier and process table entry.

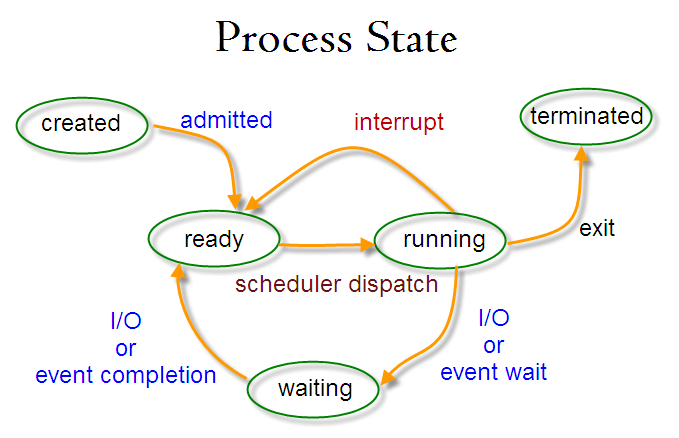
#### Daemons

These are special types of background processes that start at system startup and keep running forever as a service; they don’t die. They are started as system tasks (run as services), spontaneously. However, they can be controlled by a user via the init process. In fact, init process is a daemon.

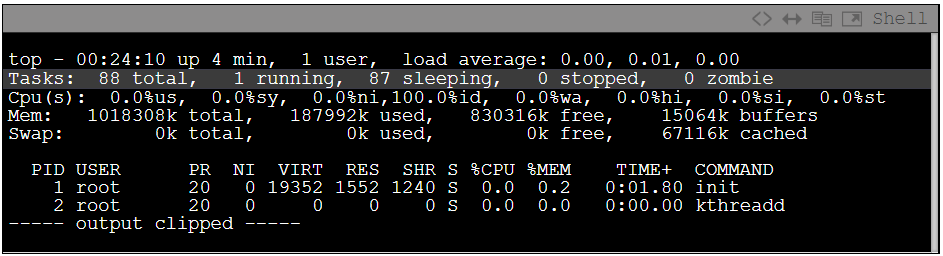
The normal way to launch a daemon process is to in the directory /etc/init.d. In addition, it's a common convention to name daemon executables with a "d" at the end. For example, system ftp daemon is typically named ftpd.

## States of Processes

During execution, a process changes from one state to another depending on its environment/circumstances. In Linux, a process has the following possible states:



* **Running** – here it's either running (it is the current process in the system) or it’s ready to run (it’s waiting to be assigned to one of the CPUs).
* **Waiting (or Sleeping)** – here it's waiting for an event to occur or for a system resource. Additionally, the kernel also differentiates between two types of waiting processes: *interruptible waiting processes* (can be interrupted by signals) and *uninterruptible waiting processes* (are waiting directly on hardware conditions and cannot be interrupted by any event/signal).
* **Terminated (or Stopped)** – here it has been stopped, usually by receiving a signal.
* **Zombie** – here it's dead but it's still has an entry in the process table.



## How to Create a Process in Linux?

<https://www.tutorialspoint.com/how-to-create-a-process-in-linux>

## Process Schedulers

### What Are Process Schedulers?

Schedulers are special system software which handle process scheduling in various ways. Their main task is to **decide which process to run next**.

The idea behind the scheduler is simple: to best utilize processor time. Assuming there are runnable processes, a process should always be running. If there are more processes than processors in a system, some processes will not always be running. These processes are waiting to run. Deciding what process runs next, is a fundamental decision the scheduler must make.

### Types of Process Schedulers

#### Long-Term Scheduler

It is also called ***job scheduler***. A long-term scheduler determines which programs are admitted to the system for processing. It selects processes from the queue and loads them into memory for execution. Process loads into the memory for CPU scheduling.

The primary objective of job scheduler is to provide a balanced mix of jobs, such as I/O bound and processor bound. It also controls the degree of multiprogramming. If the degree of multiprogramming is stable, then the average rate of process creation must be equal to the average departure rate of processes leaving the system.

#### Short-Term Scheduler

It is also called ***CPU scheduler***. Its main objective is to increase system performance in accordance with the chosen set of criteria. It is the change of ready state to running state of the process. CPU scheduler selects a process among the processes that are ready to execute and allocates CPU to one of them.

#### Medium-Term Scheduler

Medium-term scheduling is a part of swapping. It removes the processes from the memory and reduces the degree of multiprogramming. It is in-charge of handling the swapped out-processes.

*What is swapping?*

A running process may become suspended if it makes an I/O request. A suspended process cannot make any progress towards completion. In this condition, to remove the process from memory and make space for other processes, the suspended process is moved to the secondary storage. This process is called *swapping*, and the process is said to be swapped out or rolled out. Swapping may be necessary to improve the process mix.

### I/O-Bound vs. Processor-Bound Processes

Processes can be classified as either I/O-bound or processor-bound. The former is characterized as a process that spends much of its time submitting and waiting on I/O requests. Consequently, such a process is often runnable, but only for short periods, because it will eventually block waiting on more I/O (this is any type of I/O, such as keyboard activity, and not just disk I/O). Conversely, processor-bound processes spend much of their time executing code. They tend to run until they are preempted because they do not block on I/O requests very often. Because they are not I/O-driven, however, system response does not dictate that the scheduler run them often. The scheduler policy for processor-bound processes, therefore, tends to run such processes less frequently but for longer periods. Of course, these classifications are not mutually exclusive. The scheduler policy in Unix variants tends to explicitly favor I/O-bound processes.

The scheduling policy in a system must attempt to satisfy two conflicting goals: fast process response time (low latency) and high process throughput. To satisfy these requirements, schedulers often employ complex algorithms to determine the most worthwhile process to run, while not compromising fairness to other, lower priority, processes. Favoring I/O-bound processes provides improved process response time, because interactive processes are I/O-bound. Linux, to provide good interactive response, optimizes for process response (low latency), thus favoring I/O-bound processes over processor-bound processors. As you will see, this is done in a way that does not neglect processor-bound processes.

### Process Priority

A common type of scheduling algorithm is priority-based scheduling. The idea is to rank processes based on their worth and need for processor time. **Processes with a higher priority will run before those with a lower priority, while processes with the same priority are scheduled round-robin (one after the next, repeating)**.

On some system (including Linux), processes with a higher priority also receive a longer [timeslice](#_2s8eyo1). The runnable process with timeslice remaining and the highest priority always runs. Both the user and the system may set a processes priority to influence the scheduling behavior of the system.

**Linux builds on this idea and provides dynamic priority-based scheduling. This concept begins with the initial base priority, and then enables the scheduler to increase or decrease the priority dynamically to fulfill scheduling objectives**. For example, a process that is spending more time waiting on I/O than running is clearly I/O bound. Under Linux, it receives an elevated dynamic priority. As a counterexample, a process that continually uses up its entire timeslice is processor bound—it would receive a lowered dynamic priority.

The Linux kernel implements two separate priority ranges.

* The first is the nice value, a number from –20 to 19 with a default of 0. Larger nice values correspond to a lower priority—you are being nice to the other processes on the system. Processes with a lower nice value (higher priority) run before processes with a higher nice value (lower priority). The nice value also helps determine how long a timeslice the process receives. A process with a nice value of –20 receives the maximum timeslice, whereas a process with a nice value of 19 receives the minimum timeslice. Nice values are the standard priority range used in all Unix systems.
* The second range is the real-time priority. By default, it ranges from 0 to 99. All real-time processes are at a higher priority than normal processes. Linux implements real-time priorities in accordance with POSIX. Most modern Unix systems implement a similar scheme.

### What Is Preemption?

Multitasking OSs come in two flavors: *cooperative multitasking* and *preemptive multitasking*. Linux, like all Unix variants and most modern OSs, provides preemptive multitasking.

In preemptive multitasking, the scheduler decides when a process is to stop running and when a new process is to resume running.

**The act of involuntarily suspending a running process is called *preemption***.

When a process enters the TASK\_RUNNING state, the kernel checks whether its priority is higher than the priority of the currently executing process. If it is, the scheduler is invoked to pick a new process to run (presumably the process that just became runnable).

### What Is Timeslice?

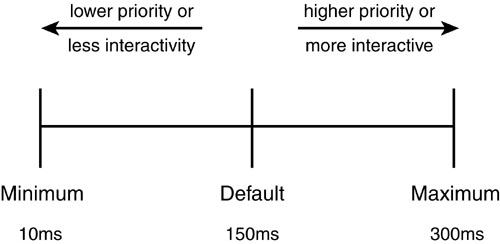
**The time a process runs before it is preempted is called the *timeslice* of the process**; in other words, how long a task can run until it is preempted.

When a process's timeslice reaches zero, it is preempted and the scheduler is invoked to select a new process.

The scheduler must dictate a default timeslice, which is not simple:

* A timeslice that is too long will cause the system to have poor interactive performance. The system will no longer feel as if applications are being concurrently executed.
* A timeslice that is too short will cause significant amounts of processor time to be wasted on the overhead of switching processes, as a significant percentage of the system's time will be spent switching from one process with a short timeslice to the next. Furthermore, the conflicting goals of I/O-bound versus processor-bound processes again arise; I/O-bound processes do not need longer timeslices, whereas processor-bound processes crave long timeslices (to keep their caches hot, for example).

In many OSs, this observation is taken to heart, and the default timeslice is rather low—for example, 20ms. Linux, however, takes advantage of the fact that the highest priority process always runs. The Linux scheduler bumps the priority of interactive tasks, enabling them to run more frequently. Consequently, the Linux scheduler offers a relatively high default timeslice (see table about process timeslice calculation below).



Furthermore, the Linux scheduler dynamically determines the timeslice of a process based on priority. This enables higher priority, allegedly more important, processes to run longer and more often. Implementing dynamic timeslices and priorities provides robust scheduling performance.

Note that a process does not have to use all its timeslice at once. For example, a process with a 100 millisecond timeslice does not have to run for 100 milliseconds in one go or risk losing the remaining timeslice. Instead, the process can run on five different reschedules for 20 milliseconds each. Thus, a large timeslice also benefits interactive tasks—while they do not need such a large timeslice all at once, it ensures they remain runnable for as long as possible.

When a process's timeslice runs out, the process is considered expired. A process with no timeslice is not eligible to run until all other processes have exhausted their timeslice (that is, they all have zero timeslice remaining). At that point, the timeslices for all processes are recalculated. The Linux scheduler employs an interesting algorithm for handling timeslice exhaustion that is discussed in a later section.

# Inter-Process Communication (IPC)

IPC mechanisms:

* Shared files
* Shared memory (with semaphores)
* Pipes (named and unnamed)
* Message queues
* Sockets
* Signals

## Shared Files

### What Is It?

Consider the simple case in which one process (*producer*) creates and writes to a file, and another process (*consumer*) reads from this same file:

A race condition might arise: the *producer* and the *consumer* might access the file at the same time, thereby making the outcome indeterminate. To avoid this, the **file must be locked** in a way that prevents a conflict between a write operation and any other operation, whether a read or a write.

The locking can be summarized as follows:

* The *producer* should gain an ***exclusive lock*** on the file before **writing** to it. An exclusive lock can be held by one process at most, which rules out the race condition because no other process can access the file until the lock is released.
* The *consumer* should gain at least a ***shared lock*** on the file before **reading** it. Multiple readers can hold a shared lock at the same time, but no writer can access a file when even a single reader holds a shared lock.

A *shared lock* promotes efficiency. If one process is just reading a file and not changing its contents, there is no reason to prevent other processes from doing the same. Writing, however, clearly demands *exclusive lock* to a file.

### Pros and Cons

The Shared file's contents could be very lengthy, arbitrary bytes (e.g., a digitized movie), which makes file sharing an impressively flexible IPC mechanism.

The downside is that file access is relatively slow, whether the access involves reading or writing.

### Example

**The producer:**

#include <stdio.h>

#include <fcntl.h>

#include <unistd.h>

#include <string.h>

#define FILENAME    "data.dat"

#define DATASTRING  "Now is the winter of our discontent\nMade glorious summer by this sun of York\n"

int main()

{

    struct flock lock;          // File lock

    lock.l\_whence = SEEK\_SET;   // Seek from the beginging of the file

    lock.l\_start = 0;           // 1st byte in file

    lock.l\_len = 0;             // Until EOF

    lock.l\_pid = getpid();      // Current process id

    // Open file (in read/write, auto-create-if-not-exist mode)

    int fd; // File descriptor

    if ((fd = open(FILENAME, O\_RDWR | O\_CREAT, 0666)) < 0) {

        perror("open failed...");

        return -1;

    }

    // Before writing, set the file lock

    //    The lock here is Write lock (prevent any reading/writing while the file is writing)

    //    and this is an Exclusive Lock

    lock.l\_type = F\_WRLCK;

    if (fcntl(fd, F\_SETLK, &lock) < 0) {

        perror("fcntl failed to get lock...");

        close(fd);

        return -1;

    }

    write(fd, DATASTRING, strlen(DATASTRING));

    fprintf(stderr, "Process %d has written to data file...\n", lock.l\_pid);

    // After writing, release the file lock

    lock.l\_type = F\_UNLCK;      // Unlock

    if (fcntl(fd, F\_SETLK, &lock) < 0) {

        perror("explicit unlocking failed...");

        close(fd);

        return -1;

    }

    close(fd);

    return 0;

}

**The consumer:**

#include <stdio.h>

#include <fcntl.h>

#include <unistd.h>

#define FILENAME    "data.dat"

int main()

{

    struct flock lock;          // File lock

    lock.l\_type = F\_WRLCK;      // Write lock

    lock.l\_whence = SEEK\_SET;   // Seek from the beginging of the file

    lock.l\_start = 0;           // 1st byte in file

    lock.l\_len = 0;             // Until EOF

    lock.l\_pid = getpid();      // Current process id

    // Open file (in read mode)

    int fd; // File descriptor

    if ((fd = open(FILENAME, O\_RDONLY)) < 0) {

        perror("open to read failed...");

        return -1;

    }

    // Get file lock

    //   If it is locked, return

    fcntl(fd, F\_GETLK, &lock);

    if (lock.l\_type != F\_UNLCK) {

        perror("file is still write locked...");

        close(fd);

        return -1;

    }

    // Before reading, set the file lock

    //   The lock here is Read lock (prevent any writing while the file is reading)

    //   and this is a Shared Lock, so other processes can also read the file during the locking

    lock.l\_type = F\_RDLCK;

    if (fcntl(fd, F\_SETLK, &lock) < 0) {

        perror("can't get a read-only lock...");

        close(fd);

        return -1;

    }

    // Read the bytes (they happen to be ASCII codes) one at a time

    int buffer;

    while (read(fd, &buffer, 1) > 0) {      // 0 signals EOF

        write(STDOUT\_FILENO, &buffer, 1);   // Write one byte to the standard output

    }

    // Release file lock

    lock.l\_type = F\_UNLCK;

    if (fcntl(fd, F\_SETLK, &lock) < 0) {

        perror("explicit unlocking failed...");

        close(fd);

        return -1;

    }

    close(fd);

    return 0;

}

## Shared Memory

### What Is It?

Linux provide two APIs for shared memory:

* System V API (legacy)
* POSIX API (modern)

These APIs should never be mixed in a single application.

A downside of the POSIX API is that features are dependent upon the installed kernel version, which impacts code portability. For example, the POSIX API, by default, implements shared memory as a *memory-mapped file*: for a shared memory segment; the system maintains a backing file with corresponding contents. Shared memory under POSIX can be configured without a backing file, but this may impact portability. The backing file combines the benefits of memory access (speed) and file storage (persistence).

To synchronize access to the shared memory, a ***semaphore*** is used. In general, semaphore has two types:

* **Counting semaphore**: Has a value (typically initialized to zero) that can be incremented. Consider a bicycle-renting shop with a hundred of bicycles in stock. Every time a bike is rented, the semaphore is incremented by one; when a bike is returned, the semaphore is decremented by one. Rentals can continue until the value hits 100, but then must halt until at least one bike is returned, thereby decrementing the semaphore to 99.
* **Binary semaphore**: Is a special case requiring only two values: 0 and 1. In this situation, a semaphore acts as a mutex: a mutual exclusion construct. When the semaphore's value is 0, the memwriter alone can access the shared memory. After writing, this process increments the semaphore's value, thereby allowing the memreader to read the shared memory.

### Pros and Cons

### Example