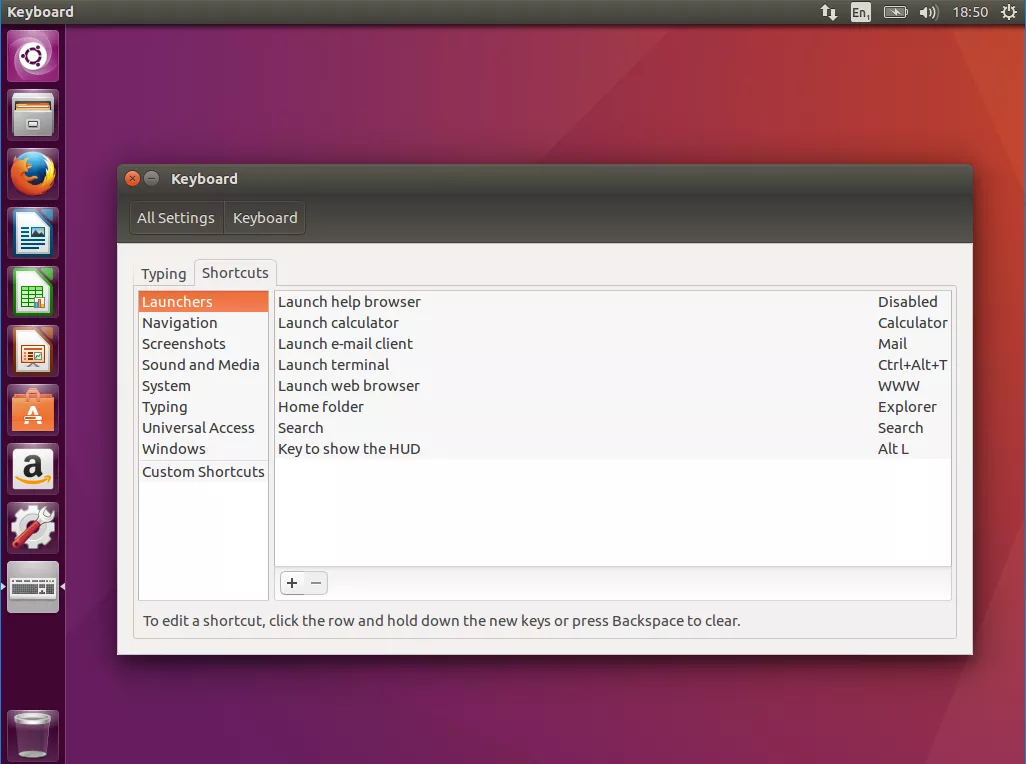
# Shortcut Keys

**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>

# 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.

**Notes**:

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

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

* 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.

**Tips**:

**1.** List all aliases with command alias:

**$ alias**

alias alert='notify-send --urgency=low -i "$([ $? = 0 ] && echo terminal || echo error)" "$(history|tail -n1|sed -e '\''s/^\s\*[0-9]\+\s\*//;s/[;&|]\s\*alert$//'\'')"'

alias egrep='egrep --color=auto'

alias fgrep='fgrep --color=auto'

alias grep='grep --color=auto'

alias l='ls -CF'

alias la='ls -A'

alias ll='ls -alF'

alias ls='ls --color=auto'

**2.** Aliases with Arguments

For example, in the ~/.bashrcfile, you add:

function make-dir-cd() {

mkdir -p -- "$1/$2/$3"

}

After refreshing the configuration, you can create three new nested directories:

$ make-dir-cd bael dung 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 or -e | 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 |
| -H | Show process hierarchy |

**Tips:**

* Display a **tree** of processes: pstree
* View **threads** in a process:

The following command list all threads created by a process:

$ ps -T -p <pid>

For example:

$ ps -T -p 7801

Output:

PID SPID TTY TIME CMD

7801 7801 pts/3 00:00:00 rrh\_main

7801 7819 pts/3 00:00:00 mt\_mkm

7801 7822 pts/3 00:00:00 mt\_send

7801 7823 pts/3 00:00:00 mt\_recv

7801 7821 pts/3 00:00:00 mt\_log

### 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.

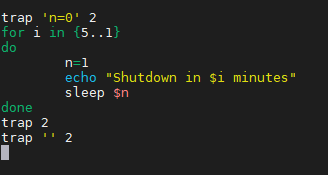
### jobs

List all processes running in background

### trap

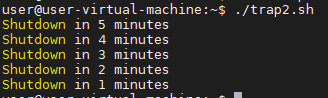
**Example 1:**

Nội dung file 'trap2.sh':

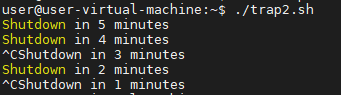


Kết quả chạy script:

Không nhấn nút DEL:

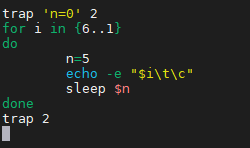


Có nhấn nút DEL:



**Example 2:**

**Nội dung file 'uncount.sh':**



Không nhấn nút DEL:



Có nhấn nút DEL:



### parallel

First, install parallel

$ sudo apt-get update sudo apt-get install parallel

**Usage**: This brings in the parallel tool — think of it as "xargs on steroids", letting you **run the same command in parallel over many inputs**.

**Syntax**:

**Common options**:

**Examples**:

**1. List directory size in ascending order:**

$ parallel du -h --max-depth 1 ::: . | sort -h

4.0K ./scripts

12K ./docs

120K ./config

850K ./assets

2.5M ./src

5.2M ./node\_modules

8.0M .

This command shows the sizes of all top-level items (files and folders) in the current directory, using du, runs that with parallel (though here it only runs once), and sorts the results by size in human-readable format.

Here you see each subfolder’s size, sorted smallest → largest, with the last line (8.0M ) being the total.

**2. Feed parallel a list of directories via find or a shell glob:**

$ find . -mindepth 1 -maxdepth 1 -type d | parallel du -h --max-depth 0 {}

12K ./logs

1.1M ./data

728K ./results

5.3M ./src

This command finds all directories one level below the current one (.), then runs du -h --max-depth 0 on each of them in parallel. That gives you the *total disk usage* for each top-level subdirectory.

## Task Managers

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

### top

View threads in a process:

The following command shows a real-time view of individual threads created by a process.

To restrict the top output to a particular process and check all threads running inside the process:

$ top -H -p <pid>

**Tip**: You can toggle on or off thread view mode while top is running, by pressing 'H' key.

## Memory Managers

### free

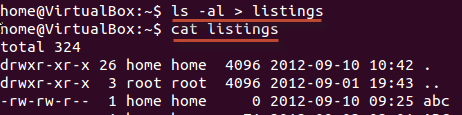
## Redirection Operator

### 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.

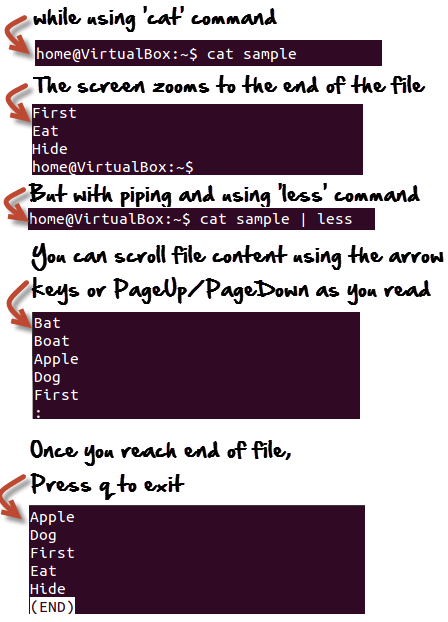
### Pipe Redirection (|)

**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 have 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
* Chmod calculator: <https://chmodcommand.com/>

### 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: $ 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>

### zip

**Usage**: Create files that are archived in the .zip format.

**Syntax**:

1. Create file: $ zip -r <zip-file.zip> source\_folder

### unzip

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

**Syntax:**

1. Extract file: $ unzip <zip-file.zip>
2. Extract file to specific folder: $ unzip <zip-file.zip> -d destination\_folder

**Note:**

* Need to install the package first: $ sudo apt-get install unzip

### 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

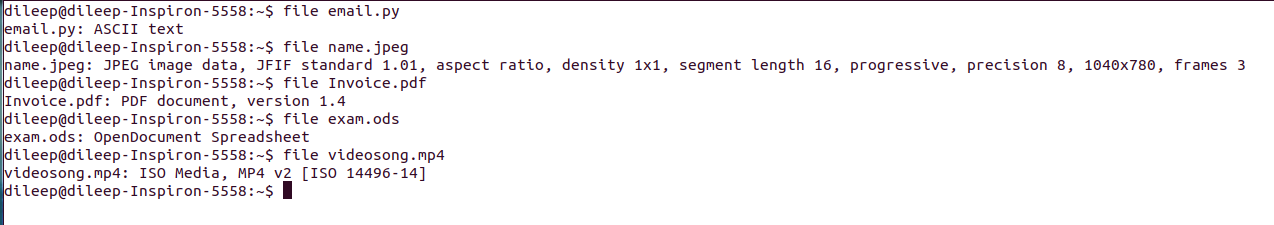
**Usage**: Reports a file's type.

**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. And it works with **Regex**.

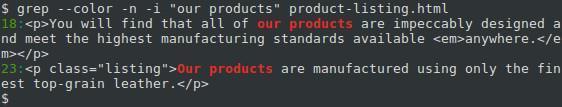
**Syntax**: grep [options] <patterns> <files>

**Common options**:

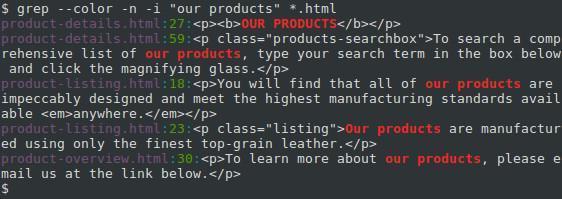
|  |  |
| --- | --- |
| -i | Ignore case sensitivity and show results for both uppercase and lowercase |
| -w | Match whole word only |
| -A <*NUM*> | Print *NUM* lines of trailing context *after* matching lines |
| -B <*NUM*> | Print *NUM* lines of trailing context *before* matching lines |
| -C <*NUM*> | Print *NUM* lines of trailing context *before and after* matching lines |
| -n | Print the line number of each matching |
| --color | Highlight each matching with a default color |

**Examples**:

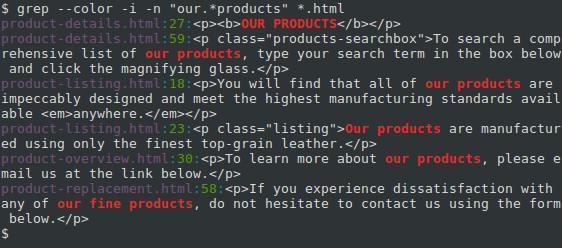
1. Without a [wildcard](https://www.computerhope.com/jargon/w/wildcard.htm) in the file name:



2. With a wildcard in the file name:



3. With a wildcard in the search pattern:



More details:

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

## 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.

### netstat

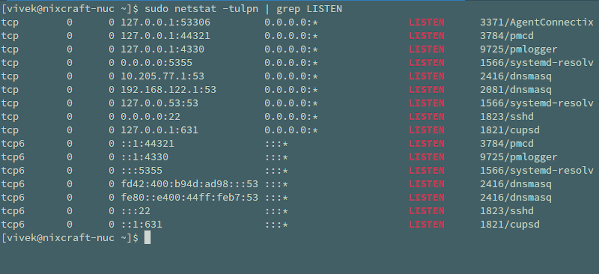
**Usage**: Show network status and protocol statistics

**Example**:

1. Check all open ports: $ netstat -tulpn | grep LISTEN

Where:

* **-t** : All TCP ports
* **-u** : All UDP ports
* **-l** : Display listening server sockets
* **-p** : Show the PID and name of the program to which each socket belongs
* **-n** : Don’t resolve names
* **| grep LISTEN** : Only display open ports by applying grep command filter.



## 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>

## Log

### dmesg

**Usage**: Also called as "driver message" or "display message". Used to examine the kernel ring buffer and print the **message buffer of kernel**. Its output contains the messages produced by the device drivers.

The contents of kernel ring buffer are also stored in /var/log/dmesg file.

Refs: <https://www.geeksforgeeks.org/dmesg-command-linux-driver-messages/>

## 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**:

|  |  |
| --- | --- |
| Command | Usage |
| 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. |

**Common options**:

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

**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 && sudo apt upgrade -y.

|  |
| --- |
| **You might not know!**  Before 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> |

#### 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**:

|  |  |
| --- | --- |
| Command | Usage |
| Package Level | |
| search [--all] <package-name> ... | Search package details |
| install <package-name> | Install package(s) |
| check-update <package-name> ... | Check for available package updates |
| update <package-name> ... | Update package(s) |
| reinstall <package-name> ... | Reinstall package(s) |
| remove <package-name> ... | Remove package(s) |
| history | Display the transaction history.  sudo dnf history displays all installation/update/remove events in the past |
| info <package-name> | Display details about package(s) |
| Module Level | |
| module list | List all available module streams that you can install.  Tip: To list only installed module streams, add option --installed. |
| module list <module-name-search-pattern> | List all available module streams for a particular module.  E.g.: dnf module list python\*  Name Stream Profiles  python27 2.7 [d] common [d]  python36 3.6 [d][e] build, common [d]  python38 3.8 [d] build, common [d] |
| module info <module-name> | Get detailed information about a module stream:  E.g.: dnf module info python39 # shows packages, profiles, dependencies for Python 3.9 |

**Notes**:

* **Package level vs Module level**? Package level focuses on individual packages like python3.9 or python3.9-devel. By contrast, module level is about a specific *module stream*, which is a collection of related packages (e.g., the python39 module stream includes python3.9, python3.9-devel, pip3.9, etc.).

#### rpm

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

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

**Common options**:

|  |  |
| --- | --- |
| Option | Usage |
| -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.

## 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>

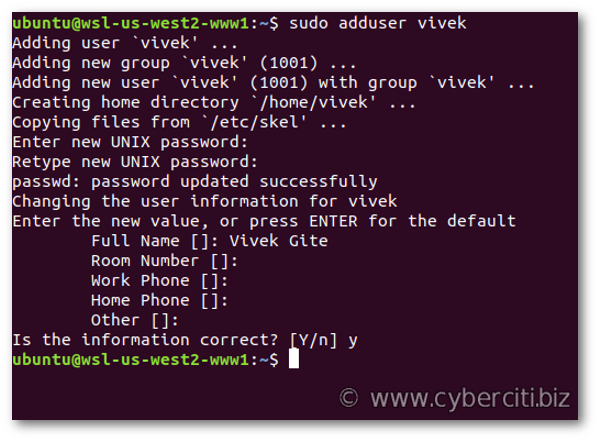
### Add New User

**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/)

# 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.

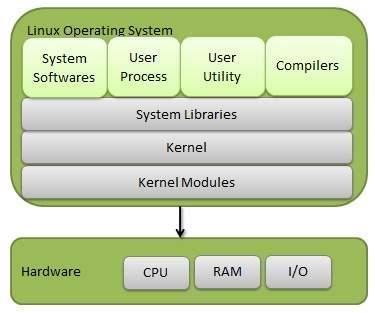
### System Library

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.

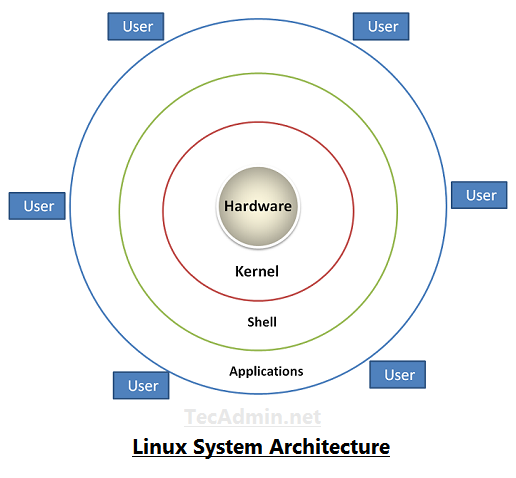
The term "library" is usually used to describe a collection of implementations of behavior written in terms of a language. It contains a well-defined interface by which the behavior is invoked. It means people who want to make a higher-level program can use the library to make system calls over and over 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



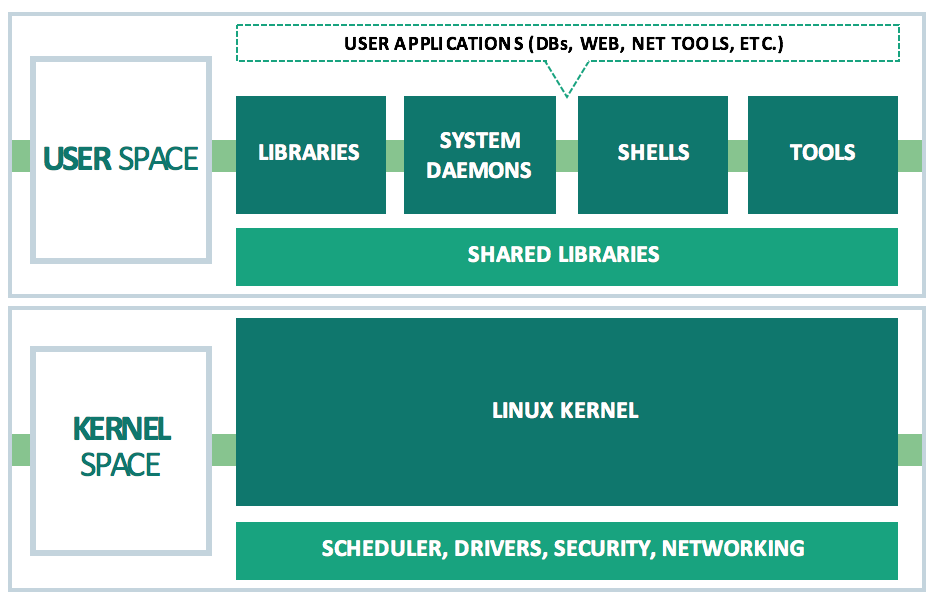
* **Hardware** − Hardware consists of all physical devices attached to the system. For example: hard disk, RAM, motherboard, CPU, etc.
* **Kernel** – It is the core part of Linux.
* **Shell** – It is an interface between the user and the kernel, hiding complexity of kernel's functions from the user. 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.



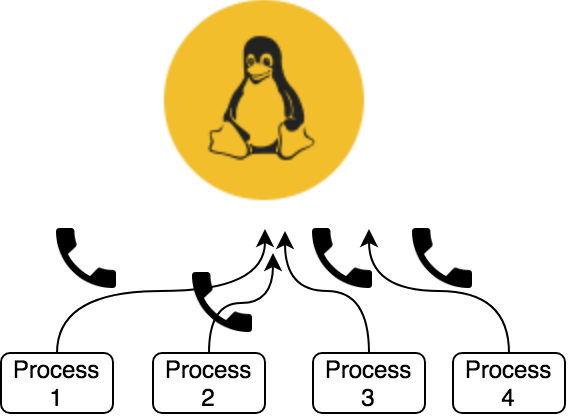
All applications, even system daemon processes which perform critical OS functions, must make what is called a [system call](#_4d34og8) to the kernel in the kernel space in order to access system resources such as memory or network devices. Every modern multi-user OS has some type of user space versus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

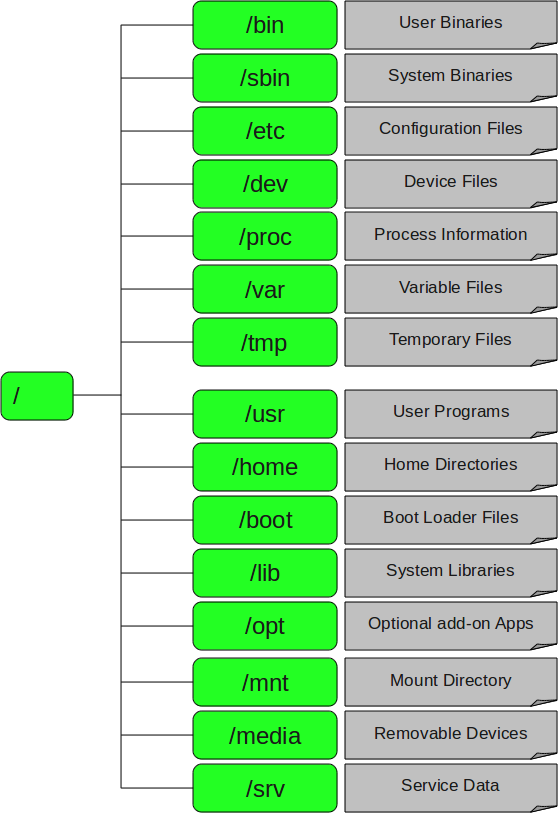
In kernel mode, the executing code has unrestricted access to the underlying hardware, meaning it has **full access to all resources of the computer**. It can execute any CPU instruction and reference any memory address. Kernel mode is generally reserved for the lowest-level, most trusted functions of 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 / on Linux) and inside that folder you can add other folders (say var, tmp, etc.) 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 var/log/syslog 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 ] ...

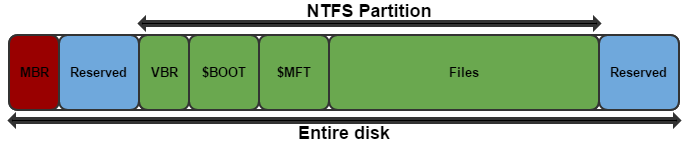
Each file on the disk is stored in at least one of these blocks and the OS must know which blocks belong to each file. For example, a short text file might need only one block and the OS may assign block 1 to it. If you download a video from the Internet, it might need more than one block and may be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

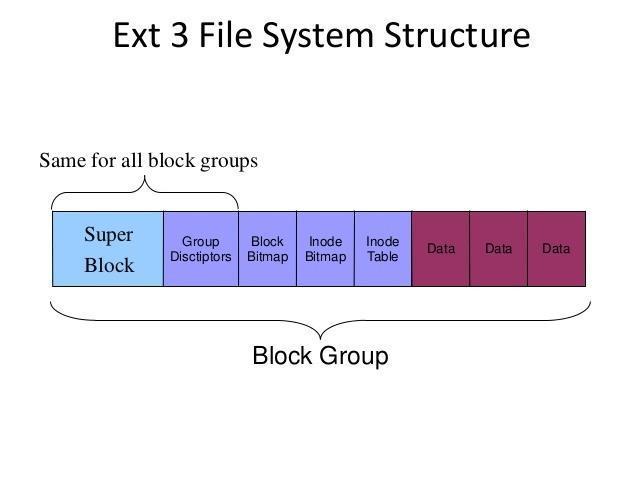
But in reality, thing might not that simple. For example, if the OS assign blocks 2, 3 and 5 for the video file because the block 4 is used by another file, what happens if you delete a file? You will have to use that space again, but how do you deal with the fact that you now have a gap? If the file 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 the computer can use any kind of storage device (hard drive, CD-ROM, network share, etc.), the OS must make it accessible through the computer's file system. This process is called *mounting*.

In particular, **mounting is 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 (/).

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, 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/>

# File

## File Descriptor vs. File Pointer

File descriptors and file pointers serve different purposes in the context of I/O operations.

### File Descriptor

An integer value presenting an open file in the OS, returned by open() system call.

1. Low/Kernel level handler for interacting with files.
2. Doesn't include buffering and such features.
3. More efficient for system-level operations. System calls using file descriptors are typically faster since they make direct calls to the OS kernel.
4. Used for a wide range of purposes beyond regular file I/O, such IPC via sockets or pipes.

### File Pointer

A pointer to a C structure returned by fopen() library function, which is used to identifying a file, wrapping the file descriptor, buffering functionality and all other functionalities needed for I/O operation.

1. High-level interface provided by standard libraries.
2. Encapsulates file descriptors and includes additional information like buffering, error indication, EOF detection, etc.
3. More convenient for I/O operations, but introduce overhead due to features like buffering, etc.

# Permission

## Privileged User vs Non-Privileged User

### Privileged User

* **Root or administrator access**: Privileged users, often referred to as root or administrators, have full control over the system. They can execute any command, access all files, installing software, and make system-wide changes.
* **Greater risk**: Because they have unrestricted access, actions taken by privileged users can significantly impact system stability and security. If compromised, a malicious actor can do extensive damage.

### Non-Privileged User

* **Limited access**: Non-privileged users, or regular users, have restricted permissions. They cannot access / modify files they don't own, or execute commands that require elevated privileges (like installing software or changing configurations). They can only perform tasks permitted by the system administrator, typically through mechanisms such as sudo.
* **Safer for the System**: Since they can't make system-level changes, they pose less risk to system integrity. Any weakening of security or stability usually occurs in their own user space.

## Ambient Capabilities

### What Are Ambient Capabilities?

Ambient capabilities are a way to **allow non-privileged processes to perform some specific privileged operations** without giving them full root access. Here are some reasons why they are useful:

* **Fine-grained control**: Ambient capabilities allow you to grant specific permissions that a process needs to function properly, rather than giving it all the privileges associated with being a root user. This follows the principle of least privilege, enhancing security.
* **Running processes as non-privileged users**: You can run processes as non-privileged users but still enable them to perform certain privileged operations (like binding to low-numbered ports, accessing specific hardware, etc.) by assigning the necessary capabilities.
* **Enhanced security**: By limiting the capabilities available to a process, you reduce the potential attack surface. If an attacker compromises a process, they will have limited privileges and thus less ability to exploit the system.
* **Retaining capabilities across user changes**: Using ambient capabilities along with SecureBits (e.g., keep-caps), allows for capabilities to persist even when the process changes its user context, making it easier to manage permissions in complex scenarios.
* **Simplifying privilege management**: Instead of needing to set up specific permissions in various ways (like using sudoers files or group memberships), you can directly assign the necessary capabilities to processes, making management simpler.

### Common Ambient Capabilities

|  |  |
| --- | --- |
| **Ambient capabilities** | **Description** |
| CAP\_SYS\_ADMIN | One of the most powerful capabilities, allowing a program to perform various system administration tasks, such as mounting filesystems, managing kernel parameters, and executing certain system calls. |
| CAP\_NET\_BIND\_SERVICE | Allows a process to bind to a network port below 1024, which is typically reserved for privileged users.  This is often used by web servers to listen on standard HTTP (port 80) or HTTPS (port 443) ports. |
| CAP\_DAC\_OVERRIDE | Grants the ability to bypass file owner and permission checks. This enables a process to read and write to files regardless of the file's ownership and permission settings. |
| CAP\_SYS\_PTRACE | Enables a process to use the ptrace system call, which is necessary for debugging other processes. It allows for examining or controlling the execution of another process. |
| CAP\_CHOWN | Allows a process to change the ownership of files, which is typically restricted to privileged users |
| CAP\_SETUID | Grants the ability to change the user ID of the current process.  This is particularly useful for programs that need to drop privileges after gaining access to elevated rights. |
| CAP\_SETGID | Similar to CAP\_SETUID, but it allows changing the group ID of the current process. |
| CAP\_SYS\_BOOT | Permits the process to reboot the system, typically limited to privileged users. |
| CAP\_SYS\_RESOURCE | Allows a process to override resource limits set for the user, such as increasing limits on file size and the number of processes. |
| CAP\_SYS\_NICE | Grants the capability to change the priority of processes, allowing a process to adjust its own scheduling priority or that of others. |

# 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 A 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

Depending on the scenarios, a proces in Linux can be categorized into following types:

### Foreground vs Backgroud

#### 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.

### Parent vs Child

#### Parent 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

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 vs Orphaned

#### Zombie Processes

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 Processes

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.

### Service vs Daemon

#### Service Processes

These are special types of **background processes that are not primarily intended to be used interactively by users**. They can be started, stopped, and managed independently of other processes.

In older Linux distributions, the old SysVinit init system manage each services through an init script located in the /etc/init.d/ directory.

In modern Linux distributions, the modern systemd init system manages services through a configuration file. These files have the .service extension and are stored in the /etc/systemd/system/ directory or in /lib/systemd/system/ directory.

For more details about systemd, check [this session](#_Systemd).

#### Daemon Processes

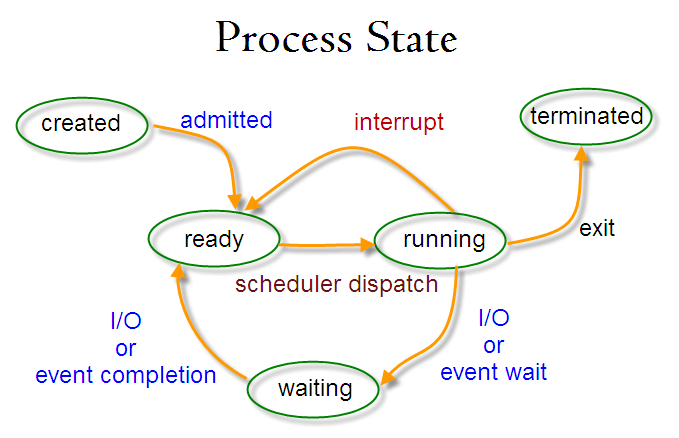
These are special types of **service processes that are designed to start at system startup** and keep running forever (until died).

In addition, it's a common convention to name daemon executables with a "d" at the end. For example, ftp daemon is named ftpd, ssh server daemon is named sshd, etc.

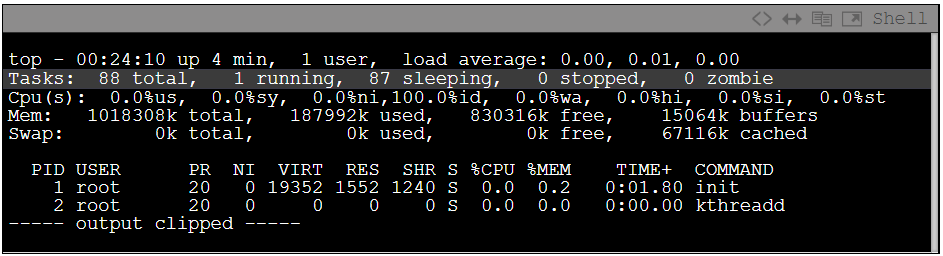
Note: The term "service" and "daemon" are usually called interchangeably. So, we should not pay too much attention to the name, but can always call service for simple.

## 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.



* **Orphan** –

## 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?

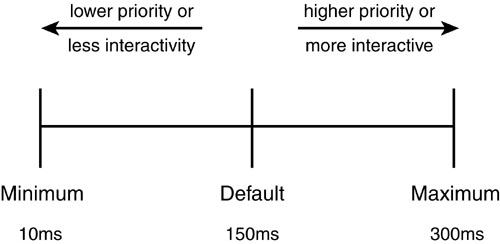
**The time a process runs before it is preempted is called the *timeslice* of the process**; in other words, how long a task can run until it is preempted.

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.

# SystemD

Official website: <https://systemd.io/>

## What Is SystemD?

systemd is a **system and service manager** that runs as PID 1 and starts the rest of the system. The suite provides:

* Aggressive parallelization capabilities
* Uses socket and D-Bus activation for starting services
* Interface with the bootloader to receive performance data and other information, and pass control information
* Offers on-demand starting of daemons
* Keeps track of processes using Linux control groups
* Maintains mount and automount points
* Supports SysV and LSB init scripts and works as a replacement for sysvinit.

## ServiceD Commands

SystemD provides a tool called systemctl to control unit behavior. Following are the most commonly used controls:

|  |  |
| --- | --- |
| **Command** | **Description** |
| systemctl start <unit-name> | Start a unit |
| systemctl stop <unit-name> | Stop a unit |
| systemctl restart <unit-name> | Restart a unit |
| systemctl status <unit-name> | Check the status of a unit.  Along with the status, this command also displays journal log of the specific unit since the last running. So it’s very convenient compared to using journalctl.  If the unit is running, the status will be active. Otherwise, the status will be inactive. |
| systemctl is-active <unit-name> | Check the active state of a unit.  If the unit is *active*, the command will exit with a status of 0.  If the unit is *inactive*, *activating*, *deactivating*, or in a failed state, the command will exit with a status of 3. |
| systemctl enable <unit-name> | Enable a unit to start automatically at boot |
| systemctl disable <unit-name> | Disable a unit to start automatically at boot |
| systemctl reload <unit-name> | Reload a unit configuration without restarting it. |
| systemctl daemon-reload | Apply changes to unit files without rebooting the system.  (If the modified unit file belongs to a running service, restart the service. Otherwise, you'll get console warning from systemd) |
| systemctl show <name> -p <option-name-in-unit-file> | Display value of an option in the unit file |
| systemctl list-unit-files | List unit files |
| systemctl list-dependencies <unit-file> | List the dependency of a specific unit file |

Ref: <https://linuxhandbook.com/systemctl-commands/>

**Other usefull commands:**

|  |  |
| --- | --- |
| **Command** | **Description** |
| systemd-delta | Displays changes made to the original unit files, helping to identify overrides and modifications. |
| systemd-analyze | Provides insights into the boot process, including time taken by services and the overall boot time. |
| systemd-ask-password | Prompts the user for a password securely, typically used in scripts or services requiring user input. |
| systemd-tty-ask-password-agent | Securely request passwords from the user interactively. |
| systemd-cat | Directs standard input to the systemd journal, allowing you to log output from commands or scripts. |
| systemd-cgls | Displays the processes running in a specific cgroup, allowing you to visualize resource allocation and process management. |
| systemd-cgtop | Displays real-time statistics for all cgroups, including CPU, memory, and I/O usage. |
| systemd-detect-virt | Identifies if the system is running in a virtualized environment and provides details about the virtualization type. |
| systemd-escape | Prepares strings by escaping special characters, making them safe for use in unit file names or properties. |
| systemd-hwdb | Queries or updates the hardware database used for matching devices with their properties. |
| systemd-inhibit | Prevents specified operations from occurring, often used by applications that need to ensure tasks are completed before shutdown. |
| systemd-machine-id-setup | Sets up the machine ID file, which is used to uniquely identify the system across reboots. |
| systemd-mount | Manages filesystem mounts using systemd, often used in conjunction with unit files. |
| systemd-umount | Unmounts filesystems managed by systemd, similar to the standard umount command. |
| systemd-notify | Sends messages from services to systemd, indicating readiness or other status updates. |
| systemd-path | Displays various paths used by systemd, including configuration files, runtime directories, and more. |
| systemd-resolve | Queries DNS servers and displays information about DNS resolution. |
| systemd-run | Creates and starts a transient service unit on-the-fly.  It's particularly useful for running commands or scripts as systemd services without needing to create a permanent service unit file.  Once the command finishes, the unit is automatically removed.  E.g.:  $ sudo systemd-run echo "Hello, World!"  $ sudo systemd-run --unit=my\_script\_service /path/to/script.sh  $ sudo systemd-run --property=MemoryLimit=100M /path/to/your/command |
| systemd-socket-activate | Starts services automatically when a socket is accessed, optimizing resource usage. |
| systemd-stdio-bridge | Allows programs that do not use the journal directly to log messages to it. |
| systemd-sysusers | Reads a configuration file to create user and group entries during system initialization. |
| systemd-tmpfiles | Reads configuration files to create or clean up temporary files and directories based on defined rules. |

Ref: <https://www.man7.org/linux/man-pages/man1/systemd-delta.1.html>

## ServiceD Unit File

A unit refers to any resource that the system knows how to operate on and manage. This is the primary object that systemd know how to deal with.

### Unit File Types

Systemd categories units according to the type of resource they describe. The easiest way to determine the type of a unit is with its type suffix, which is appended to the end of the resource name. The following list describes the types of units available to systemd:

* .service: A service unit describes how to manage a service or application on the server. This will include how to start or stop the service, under which circumstances it should be automatically started, and the dependency and ordering information for related software.
* .socket: A socket unit file describes a network or IPC socket, or a FIFO buffer that systemd uses for socket-based activation. These always have an associated .service file that will be started when activity is seen on the socket that this unit defines.
* .device: A unit that describes a device that has been designated as needing systemd management by udev or the sysfs filesystem. Not all devices will have .device files. Some scenarios where .device units may be necessary are for ordering, mounting, and accessing the devices.
* .mount: This unit defines a mountpoint on the system to be managed by systemd. These are named after the mount path, with slashes changed to dashes. Entries within /etc/fstab can have units created automatically.
* .automount: An .automount unit configures a mountpoint that will be automatically mounted. These must be named after the mount point they refer to and must have a matching .mount unit to define the specifics of the mount.
* .swap: This unit describes swap space on the system. The name of these units must reflect the device or file path of the space.
* .target: A target unit is used to provide synchronization points for other units when booting up or changing states. They also can be used to bring the system to a new state. Other units specify their relation to targets to become tied to the target’s operations.
* .path: This unit defines a path that can be used for path-based activation. By default, a .service unit of the same base name will be started when the path reaches the specified state. This uses inotify to monitor the path for changes.
* .timer: A .timer unit defines a timer that will be managed by systemd, similar to a cron job for delayed or scheduled activation. A matching unit will be started when the timer is reached.
* .snapshot: A .snapshot unit is created automatically by the systemctl snapshot command. It allows you to reconstruct the current state of the system after making changes. Snapshots do not survive across sessions and are used to roll back temporary states.
* .slice: A .slice unit is associated with Linux Control Group nodes, allowing resources to be restricted or assigned to any processes associated with the slice. The name reflects its hierarchical position within the cgroup tree. Units are placed in certain slices by default depending on their type.
* .scope: Scope units are created automatically by systemd from information received from its bus interfaces. These are used to manage sets of system processes that are created externally.

### Unit File Location

Unit files in systemd are in one of the following directories. Files in a directory with higher priority override files with the same name in directories of lower priority.

|  |  |  |
| --- | --- | --- |
| **Directory** | **Description** | **Priority** |
| /usr/lib/systemd/system/ | Unit files distributed with **installed RPM packages**. | 3rd |
| /run/systemd/system/ | Unit files **created at run time**.  This directory takes precedence over the directory with installed service unit files.  This directory can be used to change the system’s unit behavior for the duration of the session. All changes made in this directory will be lost when the server is rebooted. | 2nd |
| /etc/systemd/system/ | Unit files created by using the systemctl enable command, and unit files added for **extending a service**.  This directory takes precedence over the directory with runtime unit files.  If you need to modify the system’s copy of a unit file, putting a replacement in this directory is the safest and most flexible way. | 1st |

### Unit File Structure

Unit files have a similar syntax to .ini file or .cfg file. They consist of following sections:

|  |  |
| --- | --- |
| **Section** | **Description** |
| [Unit] | Contains **generic** options that are not dependent on the type of the unit. These options provide **unit description**, specify the unit’s behavior, and set dependencies to other units. |
| [Unit type] | Contains type-specific directives, these are grouped under a section named after the unit type. For example, service unit files contain the [Service] section; socket unit files contain the [Socket] section; etc. |
| [Install] | Contains information about unit installation used by systemctl enable and disable commands. |

**Note**: You can comment out any line in the unit file with one or more #.

#### [Unit] Section

Here's a breakdown of the common options you'll find in the [Unit] section of unit file:

|  |  |
| --- | --- |
| **Option** | **Description** |
| Description | Human-readable description of the unit.  This text is displayed, for example, in the output of the systemctl status command. |
| Documentation | Documentation for the service. |
| Wants/Requires | Dependencies on other units that need to be started before this service.  The units listed here are activated together with the unit.  Differences between Wants and RequiredBy:   * Wants establishes an optional dependency. The dependency is wanted by the unit, but the unit can still function if the dependency has not be started. * Requires establishes a mandatory dependency. The dependency is required by the unit, and the unit will fail to start if the dependency has not be started. |
| Before/After | Units that should be started after / before this unit.  Unlike Requires, After does not explicitly activate the specified units. |
| Conflicts | Units that conflict with this service and cannot be started together.  This is opposite to Requires. |

For the full list: <https://www.man7.org/linux/man-pages/man5/systemd.unit.5.html>

#### [Install] Section

Here's a breakdown of the common options you'll find in the [Install] section of unit file:

|  |  |
| --- | --- |
| **Option** | **Description** |
| WantedBy/RequiredBy | Target that this service should be associated with. It determines when the unit should be started or stopped in relation to the system’s boot process or other system states.   * default.target: Start the unit when the system reaches the default target during the boot process or when the target is explicitly activated. * multi-user.target: Start the unit when the system reaches the multi-user target. * network-online.target: Start the unit when the network is online.   SystemD will create a symbolic link in one of above target directory for each unit file.  Differences between WantedBy and RequiredBy:   * WantedBy establishes an optional dependency. The unit is wanted by the target, but the target can still function if the unit is not available. * RequiredBy establishes a mandatory dependency. The unit is required by the target, and the target will fail to start if the unit is not available. |
| Also | List of units to be installed or uninstalled along with the unit. |
| DefaultInstance | Default instance for which the unit is enabled. See [Managing multiple instances of a service](#_Manage_Multiple_Instances). |
| Alias | Provides a space-separated list of additional names for the unit. Most systemctl commands, excluding systemctl enable, can use aliases instead of the actual unit name. |

For the full list: <https://www.freedesktop.org/software/systemd/man/latest/systemd.unit.html#%5BInstall%5D%20Section%20Options>

#### [Service] Section

Here's a breakdown of the common options you'll find in a the [Service] section of a .service file:

|  |  |
| --- | --- |
| **Option** | **Description** |
| Type | How the service is started. Common types include:   * simple: The service is expected to be a single, standalone process. systemd will consider the service to be running as long as the main process is alive. systemd does not track additional processes or perform any complex process management for the service. * forking: The service is expected to spawn a child process that becomes the main process of the service into the background. The parent process exits when the startup is complete. systemd will track the main process and consider the service running until the main process terminates.   In other words, when using forking, systemd expects the main process to fork and exit, while the child process continues running. If the child process sends notifications, systemd might not have the correct PID to associate those notifications with.   * oneshot: The service is expected to execute a single task or command and then exit. systemd considers the service to be running until the task is completed. * dbus: The service is a D-Bus service. systemd will consider the service running as long as the corresponding D-Bus name is registered. * notify: The service will send a notification to systemd to indicate that it is ready. systemd considers the service running after receiving the notification (via the sd\_notify(READY=1) function) * idle: similar to simple, the actual execution of the service binary is delayed until all jobs are finished, which avoids mixing the status output with shell output of services. |
| ExecStart/ExecStop | Command, script or executable to start or stop the service. |
| ExecStartPre/ExecStartPost | Command, script or executable to run before or after ExecStart. |
| ExecCondition | Optional commands that are executed before the commands in ExecStartPre.  Syntax is the same as for ExecStart, except that multiple command lines are allowed and the commands are executed one after the other. |
| WorkingDirectory | Working directory for the service. |
| Environment | Environment variables for the service. |
| ConditionPathExists | Path or file that should exist for the service to be started. |
| User/Group | User and group under which the service runs. |
| Restart | Restart behavior for the service when its process exits, is killed, or a timeout is reached:   * always: restart the service if it exits for any reason. * on-failure: restart the service only if it exits with a non-zero exit status. * on-abnormal: restart the service if it is terminated by a signal (including on core dump, excluding the aforementioned four signals). * on-success: restart the service if it exits successfully. * on-abort: restart the service if it exits due to an uncaught signal not specified as a clean exit status. * on-watchdog: restart the service if it's watchdog timeout expires. * no: do not restart the sevice for any reason.   When the death of the process is a result of systemd operation (e.g. service stop), the service will not be restarted.  The service process may be the main service process, but it may also be one of the processes specified with ExecStartPre, ExecStartPost, ExecStop, ExecStopPost, or ExecReload. |
| RestartSec | Delay between restart attempts.  The default value of RestartSec is 0.1s if it’s not specified. |
| TimeoutStartSec | The time to wait for start-up. If a daemon service does not signal start-up completion (via sd\_notify(READY=1)) within the configured time, the service will be considered failed and will be shut down again.  This option takes a unit-less value in seconds (e.g. 10, meaning 10 seconds), or a time span value (e.g. 10s, 5min20s). Or pass infinity to disable the timeout logic.  The default value of TimeoutStartSec is 90s if it’s not specified. |
| WatchdogSec | Watchdog timeout for a service.  The watchdog is activated when the start-up is completed. The service must call sd\_notify(WATCHDOG=1) regularly.  If WatchdogSec is not specified, then watchdog functionality is disabled by default. |
| ExecReload | Command, script or executable to reload the service configuration. |
| StandardOutput | File path which stores standard output stream (stdout) of the service.  It can be set to one of the following options:   * null: Discards the output. * journal: Sends the output to the systemd journal, which can be accessed using the journalctl command. * console: Sends the output to the console where the service is running. * syslog: Sends the output to the system log using syslog. * kmsg: Sends the output to the kernel log. * file:<file-path>: Writes the output to the specified file. This will create a new file. Make sure you create the directory already. * append:<file-path>: Writes the output to the specified file. This will append to the existing file. |
| StandardError | File path which stores standard error stream (stderr) of the service.  It can be set to the same options as StandardOutput mentioned above. |

For the full list: <https://www.man7.org/linux/man-pages/man5/systemd.service.5.html>

#### [Socket] Section

For the full list: <https://www.man7.org/linux/man-pages/man5/systemd.socket.5.html>

#### Etc.

### Creating Custom Unit File

<https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/9/html/using_systemd_unit_files_to_customize_and_optimize_your_system/assembly_working-with-systemd-unit-files_working-with-systemd#proc_creating-custom-unit-files_assembly_working-with-systemd-unit-files>

Examples:

/lib/systemd/system # cat its.service

[Unit]

Description=its stack and application

After=cv2x.service

ConditionPathExists=/data/v2x/enable-its.conf

[Service]

Type=simple

Restart=always

RestartSec=3

ExecStart=/usr/bin/its

StandardOutput=file:/path/to/output.log

[Install]

WantedBy=multi-user.target

### Extending Default Unit Configuration

<https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/9/html/using_systemd_unit_files_to_customize_and_optimize_your_system/assembly_working-with-systemd-unit-files_working-with-systemd#proc_extending-the-default-unit-configuration_assembly_working-with-systemd-unit-files>

### Overriding Default Unit Configuration

<https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/9/html/using_systemd_unit_files_to_customize_and_optimize_your_system/assembly_working-with-systemd-unit-files_working-with-systemd#proc_overriding-the-default-unit-configuration_assembly_working-with-systemd-unit-files>

### Managing Multiple Instances of a Unit

<https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/9/html/using_systemd_unit_files_to_customize_and_optimize_your_system/assembly_working-with-systemd-unit-files_working-with-systemd#con_working-with-instantiated-units_assembly_working-with-systemd-unit-files>

## SystemD Log

systemd interacts with journald, a system logging service in systemd. Check [this document](Linux%20Logging.docx) for more details.

## Managing Services With SystemD

### Service State Notification

sd\_notify() may be **called by a service to notify the service manager about state changes**. It can be used to send arbitrary information, encoded in an environment-block-like string. Most importantly, it can be used for start-up or reload completion notifications.

Common parameters:

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| READY=1 | Tells systemd that service startup is finished, or the service finished re-loading its configuration.  This is only used by systemd if the unit file has Type=notify or Type=notify-reload set. |
| WATCHDOG=1 | Tells systemd to update the watchdog timestamp.  This is the **keep-alive** ping that services need to issue in regular intervals if WatchdogSec is enabled for it. |

Full list: <https://www.man7.org/linux/man-pages/man3/sd_notify.3.html>

## QAs

**"System has not been booted with systemd as init system (PID 1). Can't operate."**

* Your Linux system is not using systemd as the init system.
* Explain:

The init system is the first process that gets executed when a Linux system boots up and is responsible for initializing the system and managing services.

To determine the init system in use on your Linux system, you can try running the following command:

$ ps -p 1 -o comm=

It shows the name of the process with PID 1, which is typically the init system. If it doesn't show "systemd", it means that your system is using a different init system.

If you specifically need to use systemd and your system is not currently configured to use it, you may consider upgrading to a newer Linux distribution or adjusting the boot configuration to use systemd as the init system. However, be cautious when making changes to the init system, as it can have significant impacts on the system's behavior and stability.

# Iptables Command

## What Is iptables?

iptables is a command-line utility for **configuring the built-in Linux kernel firewall**. It enables administrators to define chained rules that control incoming and outgoing network traffic.

The rules provide a robust security mechanism, defining which network packets can pass through and which should be blocked. These protect Linux systems from data breaches, unauthorized access, and other network security threats.

## How Does iptables Work?

iptables uses **rules** to determine what to do with a network packet. The utility consists of the following components:

* **Tables**: Tables are **files that group similar rules**. A table consists of several rule chains.
* **Chains**: A chain is a **string of rules**. When a packet is received, iptables finds the appropriate table and filters it through the rule chain until it finds a match.
* **Rules**: A rule is a statement that defines the **conditions for matching a packet**, which is then sent to a target.
* **Targets**: A target is a **decision of what to do with a packet**. The packet is either accepted, dropped, or rejected.

## Command Syntax

$ iptables [options] [chain] [rule-specification] [target]

Here is the details:

|  |  |  |  |
| --- | --- | --- | --- |
| **Option** | **Usage** | **Example** | **Explanation** |
| -A, --append [chain] | Append a rule to a chain | iptables -A INPUT -p tcp --dport 22 -j ACCEPT | Allows incoming TCP traffic on port 22 (SSH) in the INPUT chain. |
| -D, --delete [chain] | Delete a rule from a chain | iptables -D INPUT -s 192.168.1.100 -j DROP | Removes a rule that drops traffic from the specified IP address in the INPUT chain. |
| -I, --insert [chain] [rulenum] | Insert a rule at a specific position | iptables -I INPUT 1 -p tcp --dport 80 -j ACCEPT | Inserts a rule at position 1 in the INPUT chain to allow incoming HTTP traffic on port 80. |
| -R, --replace [chain] [rulenum] | Replace a rule in a chain | iptables -R INPUT 1 -p tcp --dport 443 -j ACCEPT | Replaces the first rule in the INPUT chain with a new rule to allow HTTPS traffic on port 443. |
| -L, --list [chain] | List all rules in a chain | iptables -L | Lists all rules in the default filter table's INPUT chain. |
| -F, --flush [chain] | Flush (delete all rules in) a chain | iptables -F | Removes all rules from all chains in the default filter table. |
| -Z, --zero [chain] | Zero packet and byte counters | iptables -Z INPUT | Resets the packet and byte counters for the INPUT chain to zero. |
| -N, --new-chain [chain] | Create a new user-defined chain | iptables -N mychain | Creates a new chain named "mychain" for custom rules. |
| -X, --delete-chain [chain] | Delete a user-defined chain | iptables -X mychain | Deletes the user-defined chain "mychain". |
| -P, --policy [chain] [target] | Set the policy for a chain | iptables -P INPUT DROP | Sets the default policy for the INPUT chain to DROP, blocking all incoming traffic unless allowed by specific rules. |
| -S, --list-rules [chain] | List rules in a format for iptables-restore | iptables -S | Lists the rules in a format that can be used to restore them later with iptables-restore. |
| -i, --in-interface [interface] | Specify the incoming network interface for the rule. | iptables -A INPUT -i eth0 -j ACCEPT | Allows all incoming traffic on the eth0 interface. |
| -o, --out-interface [interface] | Specifies the outgoing network interface for the rule. | iptables -A OUTPUT -o eth1 -j ACCEPT | Allows all outgoing traffic on the eth1 interface. |
| -j, --jump [target] | Specify the target for the rule | iptables -A INPUT -p tcp --dport 22 -j ACCEPT | Indicates that packets matching the rule should be *accepted* (allowed) instead of *dropped* or *rejected*. |
| -g, --goto [chain] | Jump to a specified chain | iptables -A INPUT -g mychain | Directs matching packets to the "mychain" for further processing. |
| -p, --protocol [protocol] | Specify the protocol  (e.g., tcp, udp) | iptables -A INPUT -p tcp -j ACCEPT | Specifies that the rule only applies to TCP packets. |
| -s, --source  [address[/mask]] | Match packets from the source address | iptables -A INPUT -s 192.168.1.0/24 -j ACCEPT | Allows incoming traffic from the entire subnet 192.168.1.0/24. |
| -d, --destination  [address[/mask]] | Match packets destined for the address | iptables -A OUTPUT -d 10.0.0.1 -j ACCEPT | Allows outgoing traffic to the specific IP address 10.0.0.1. |
| --sport [port] | Match packets from the source port | iptables -A INPUT -p tcp --sport 8080 -j ACCEPT | Allows incoming TCP traffic from the source port 8080. |
| --dport [port] | Match packets destined for the destination port | iptables -A INPUT -p tcp --dport 80 -j ACCEPT | Allows incoming TCP traffic destined for port 80 (HTTP). |
| -m, --match [module] | Specify a match module | iptables -A INPUT -m state --state NEW -j ACCEPT | Uses the "state" module to match only new connections. |
| -m state --state [state] | Match packets based on connection state | iptables -A INPUT -m state --state ESTABLISHED -j ACCEPT | Allows incoming packets that are part of established connections. |
| -m conntrack --ctstate [state] | Match based on connection tracking state | iptables -A INPUT -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT | Allows packets that are related to or part of established connections. |
| -j LOG --log-prefix "[prefix]" | Log matching packets to syslog | iptables -A INPUT -j LOG --log-prefix "INPUT DROP: " | Logs packets that match the rule to the system log (e.g. dmsg, /var/log/syslog) with a specified prefix. |
| -m limit | Match packets based on rate limit | iptables -A INPUT -m limit --limit 5/min -j LOG | Logs packets at a maximum rate of 5 per minute. |
| --limit [rate] | Specify maximum rate of matching packets | iptables -A INPUT -m limit --limit 10/min -j ACCEPT | Allows a maximum of 10 connections per minute to be accepted. |
| --limit-burst [n] | Specify number of packets allowed in a burst | iptables -A INPUT -m limit --limit-burst 5 -j ACCEPT | Allows an initial burst of up to 5 packets before enforcing the rate limit. |
| -t nat | Specify the NAT table | iptables -t nat -A POSTROUTING -j MASQUERADE | Applies the rule to the NAT table, enabling masquerading for outgoing packets. |
| -A PREROUTING | Add a rule to the PREROUTING chain | iptables -t nat -A PREROUTING -p tcp --dport 80 -j DNAT --to-destination 192.168.1.2 | Redirects incoming HTTP traffic to a specific internal IP address. |
| -A POSTROUTING | Add a rule to the POSTROUTING chain | iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE | Enables NAT for outgoing packets on the specified interface (eth0). |
| -A OUTPUT | Add a rule to the OUTPUT chain | iptables -A OUTPUT -p tcp --dport 53 -j ACCEPT | Allows outgoing DNS queries (UDP/TCP on port 53). |
| -n | Numeric output (do not resolve hostnames) | iptables -L -n | Lists the rules without resolving IP addresses to hostnames, showing numeric values only. |
| -v | Verbose output | iptables -L -v | Provides detailed information about each rule, including packet and byte counts. |

### Chains

|  |  |
| --- | --- |
| **Chain** | **Description** |
| INPUT | Used for packets that are destined for the local system.  Rules in this chain handle **incoming packets**. |
| OUTPUT | Used for packets that are generated by the local system.  Rules in this chain handle **outgoing packets**. |
| FORWARD | Used for packets that are being routed through the system (not destined for the local system).  Rules in this chain handle **forwarded packets**. |
| PREROUTING | A NAT chain used to alter packets as they arrive before routing decisions are made.  Useful for DNAT (Destination Network Address Translation). |
| POSTROUTING | A NAT chain used to alter packets as they leave the system after routing decisions have been made.  Useful for SNAT (Source Network Address Translation). |
| OUTPUT (NAT) | A NAT chain used for packets generated by the local system that are subject to NAT. |
| User-defined Chains | Custom chains created by users for specific rule management. These chains can be referenced by name in rules. |

### Protocols

|  |  |
| --- | --- |
| **Protocol** | **Description** |
| tcp | Transmission Control Protocol, used for reliable communication.  This protocol can be one of following sub-types:   |  |  |  | | --- | --- | --- | | **Sub-Type** | **Defaul Port** | **Description** | | HTTP | 80 | Hypertext Transfer Protocol, used for web traffic. | | HTTPS | 443 | HTTP Secure, a secure version of HTTP using SSL/TLS. | | FTP | 21 | File Transfer Protocol, used for transferring files. | | FTPS | 990 | FTP Secure, FTP with SSL/TLS for secure file transfer. | | SFTP | 22 | SSH File Transfer Protocol, secure file transfer over SSH. | | SMTP | 25 | Simple Mail Transfer Protocol, used for sending emails. | | POP3 | 110 | Post Office Protocol v3, used for retrieving emails. | | IMAP | 143 | Internet Message Access Protocol, used for accessing emails. | | Telnet | 23 | Used for unencrypted text communication over the network. | | SSH | 22 | Secure Shell, used for secure remote login and command execution. | |
| udp | User Datagram Protocol, used for connectionless communication.  This protocol can be one of following sub-types:   |  |  |  | | --- | --- | --- | | **Sub-Type** | **Defaul Port** | **Description** | | DNS | 53 | Domain Name System, used for resolving domain names to IP addresses. | | DHCP | 67 (server)  68 (client) | Dynamic Host Configuration Protocol, used for IP address allocation. | | RTP | 5004 | Real-time Transport Protocol, used for audio and video streaming. | | SNMP | 161 | Simple Network Management Protocol, used for network management. | |
| icmp | Internet Control Message Protocol, used for network diagnostics (e.g., ping).  This protocol can be one of following sub-types:   |  |  | | --- | --- | | **Sub-Type** | **Description** | | Echo Request | Used by the ping command to test network connectivity. | | Echo Reply | Response to an Echo Request, confirming connectivity. | | Destination Unreachable | Indicates that a destination is unreachable. | | Time Exceeded | Indicates that the time to live (TTL) for a packet has expired. | |
| igmp | Internet Group Management Protocol, used for managing multicast groups. |
| esp | Encapsulating Security Payload, part of the IPsec protocol for encryption and authentication. |
| ah | Authentication Header, also part of the IPsec protocol for authentication. |
| sctp | Stream Control Transmission Protocol, used for message-oriented communication. |
| all | Matches all protocols, used for broad rules that apply to any protocol. |

### Targets

|  |  |
| --- | --- |
| Target | Description |
| ACCEPT | **Allows** the packet to pass through the firewall. |
| DROP | Silently drops (**block**) the packet without sending a response. |
| REJECT | Drops (**block**) the packet and sends an error response to the sender. |
| LOG | Logs the packet information to the system log (usually /var/log/messages). |
| RETURN | Stops processing the current chain and returns to the previous chain. |
| DNAT | Alters the destination address of the packet (used in NAT). |
| SNAT | Alters the source address of the packet (used in NAT). |
| MASQUERADE | A form of SNAT for dynamically assigned IP addresses, typically used for internet sharing. |
| MARK | Marks the packet for use in later rules (useful for routing decisions). |
| QUEUE | Passes the packet to user space for processing by a program (requires libnetfilter\_queue). |
| RETURN | Stops processing in the current chain and returns control to the calling chain. |

### States

|  |  |
| --- | --- |
| **State** | **Description** |
| NEW | The packet is starting a new connection. |
| ESTABLISHED | The packet is part of an existing connection that has seen traffic in both directions. |
| RELATED | The packet is related to an existing connection (e.g., FTP data connection related to an FTP control connection). |
| INVALID | The packet is not associated with any known connection and is considered invalid. |
| UNTRACKED | The packet should not be tracked by the connection tracking system. |

## Notes

When working with firewalls, **take care not to lock yourself out of your own server by blocking SSH traffic (port 22, by default)**. If you lose access due to your firewall settings, you may need to connect to it via a web-based console to fix your access. Once you are connected via the console, you can change your firewall rules to allow SSH access (or allow all traffic). If your saved firewall rules allow SSH access, another method is to reboot your server.

## Tips

### Save Your Changes

iptables does not persist rules when the system reboots. All the changes apply only until the first restart. To save the rules, see the commands below:

* Debian-based systems: $ sudo apt install iptables-persistent, $ sudo netfilter-persistent save
* RedHat-based systems: $ sudo service iptables save

On the next restart, iptables will automatically reload the firewall rules.

### Order of Rule Matters

The order of rules in iptables is crucial because **iptables processes rules in a top-down manner**. Here are some key points to keep in mind regarding rule order:

1. **Sequential Processing**: Once a match is found, the corresponding action is taken, and no further rules are checked.
2. **Specificity**: More specific rules should generally be placed before more general ones. For example, if you have a rule allowing traffic from a specific IP address, it should come before a broader rule that allows traffic from all IPs.
3. **Default Policy**: If no rules match a packet, the default policy of the chain (ACCEPT, DROP, etc.) will apply. This means you might inadvertently block or allow traffic if the relevant rules are placed after the default policy.
4. **Logging**: If you're using logging rules, consider their placement. If a packet matches a logging rule, it will be logged and then processed according to the next matching rule.
5. **Testing and Debugging**: When testing new rules, the order can affect the results. If something isn't working as expected, check the order of your rules.

# Package Generator (Pktgen)

Pktgen is a package generator which helps **create network packet on kernel space**. A kind of packet generator in high speed, comparing a packet generator in user space.

## Installation

Following steps are how to build source code and install DPDK and pktgen:

Clone source code DPDK:

$ git clone http://dpdk.org/git/dpdk

Clone source code pktgen:

$ git clone http://dpdk.org/git/apps/pktgen-dpdk

Install kernel headers to allow DPDK to build its kernel modules:

$ uname -r # Find kernel version of your Linux

$ sudo apt install linux-headers-3.5.0-32-generic

# Replace "3.5.0-32-generic" with your kernel version

Install following packages:

$ sudo apt update

$ sudo apt install libpcap-dev # For testing without a real NIC or for low-speed packet capture

$ sudo apt install -y meson # For building DPDK and pktgen source code (used version: 0.59.1. Version < 47.0 will cause "ERROR: Unknown Type feature")

$ sudo apt install -y ninja-build # For building DPDK and pktgen source code

$ sudo apt install -y libnuma-dev # For building pktgen source code

Build source code DPDK:

$ cd dpdk

$ meson build

$ sudo ninja -C build

$ sudo ninja -C build install

Build source code Pktgen:

$ cd ptkgen-dpdk

$ make rebuild

**Note**: There can be other missing libraries while building DPDK and Pktgen source code, so install them if errors occur.

Refs:

<https://pktgen-dpdk.readthedocs.io/en/latest/getting_started.html>

<https://stackoverflow.com/a/66990993/14835442>

<https://suryanshpradhan.wordpress.com/2020/01/04/how-to-fix-meson-build-error-unknown-type-feature-on-ubuntu-18-04/>

## Usage

Following steps are how to run pktgen to generate package:

- Load pktgen:

$ sudo modprobe pktgen

- If you want to unload pktgen, run:

$ sudo rmmod pktgen

After loading the module, pktgen will create directory /proc/net/pktgen, with following files:

1. kpktgend\_X (X maps each CPU core of your machine)

2. pgctrl: Control the module

The module will create files for each interface that you are sending packets

For example, if you are using eth0 and eth1, you will see 2 files with the same name in /proc/net/pktgen.

To generate packages, you need to handle those files.

You can do it manually or creating shell scripts.

Here a simple sample script: <https://github.com/torvalds/linux/tree/master/samples/pktgen>

- Run the above sample script with parameters. For example:

user@user-virtual-machine:~/pkt/samples$ ./pktgen\_sample01\_simple.sh -i eth0 -m 00:0c:29:07:a3:75 -d 192.168.86.139

[sudo] password for user:

Running... ctrl^C to stop

Done

Result device: eth0

Params: count 100000 min\_pkt\_size: 60 max\_pkt\_size: 60

frags: 0 delay: 0 clone\_skb: 0 ifname: eth0

flows: 0 flowlen: 0

queue\_map\_min: 0 queue\_map\_max: 0

dst\_min: 192.168.86.139 dst\_max: 192.168.86.139

src\_min: src\_max:

src\_mac: 00:0c:29:91:01:33 dst\_mac: 00:0c:29:07:a3:75

udp\_src\_min: 9 udp\_src\_max: 109 udp\_dst\_min: 9 udp\_dst\_max: 9

src\_mac\_count: 0 dst\_mac\_count: 0

Flags: UDPSRC\_RND NO\_TIMESTAMP

Current:

pkts-sofar: 100000 errors: 0

started: 1713053451215us stopped: 1713054971845us idle: 13327us

seq\_num: 100001 cur\_dst\_mac\_offset: 0 cur\_src\_mac\_offset: 0

cur\_saddr: 192.168.159.128 cur\_daddr: 192.168.86.139

cur\_udp\_dst: 9 cur\_udp\_src: 30

cur\_queue\_map: 0

flows: 0

Result: OK: 1520630(c1507302+d13327) usec, 100000 (60byte,0frags)

65762pps 31Mb/sec (31565760bps) errors: 0

It seems that packaged are generated with pktgen.

However, when trying to capture packages with tcpdump, there is NO package captured:

user@user-virtual-machine:~$ sudo tcpdump -i eth0 -nn -A "dst 192.168.86.139"

tcpdump: verbose output suppressed, use -v or -vv for full protocol decode

listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes

^C

0 packets captured

0 packets received by filter

0 packets dropped by kernel

Also, when checking statistics files in /proc/net/pktgen, all of them are 0 bytes in size (nothing is recorded)

user@user-virtual-machine:/proc/net/pktgen$ ls -l

total 0

-rw------- 1 root root 0 9月 6 20:21 kpktgend\_0

-rw------- 1 root root 0 9月 6 20:21 kpktgend\_1

-rw------- 1 root root 0 9月 6 20:21 kpktgend\_2

-rw------- 1 root root 0 9月 6 20:21 kpktgend\_3

-rw------- 1 root root 0 9月 6 20:21 pgctrl

Refs:

<https://www.kernel.org/doc/Documentation/networking/pktgen.tx>

<http://code.dpdk.org/pktgen-dpdk/pktgen-20.11.2/source/README.md>

<https://juliofaracco.wordpress.com/2015/06/14/pktgen-a-kernel-space-traffic-generator-for-testing-network-throughput>

ISSUES:

1. Don't clear whether steps in "2. Run pktgen to generate package" are correct or wrong?

# Discretionary Access Control (DAC)

## Linux Access Attributes

Standard Linux systems use a set of access attributes that are part of every file system resource. These attributes govern the access permissions for a given file system resource. These permissions include Read, Write, and eXecute, or (RWX).

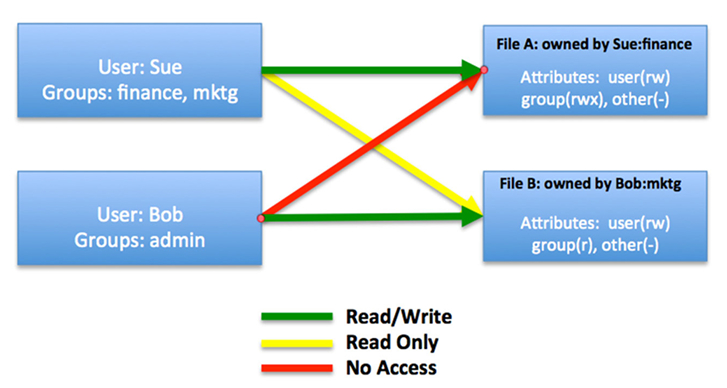
Attributes exist in three categories of system user: User, Groups, and Other.

|  |  |
| --- | --- |
| **System User Type** | **Description** |
| User | Refers to a single individual login account on a system |
| Group | Linux and UNIX systems contain many Groups to help partition access to system resources. For example, user Judy might belong to two groups Marketing and Administrators. |
| Other | Refers to any other users on the system beyond the current user or any defined groups. |

## What Is DAC?

DAC grants the owner of the resource the authority to **decide who gets access to those resources**. It is suitable for protection from accidental access violations. DAC security policy **centers on users** by answering 4 questions:

1. What file system resources can this user read, write, or execute? (User)
2. What file system resources can this group of users read, write, or execute? (Group)
3. What file system resources on the system can everyone else read, write, or execute? (Other)
4. Who can execute programs (also file system resources) on this system? (Execute permissions)



# Linux Security Modules (LSM)

## What Is LSM?

An LSM can **deny a process access to important kernel objects**. The types of objects include files, inodes (metadata about a file or directory, like file size, permissions, ownership, timestamps), task structures, credentials, and interprocess communication objects.

The main use case of an LSM is to implement **Mandatory Access Control** (MAC) policies. These help protect your system from being hacked when an attacker exploits flaws in one of the running programs.

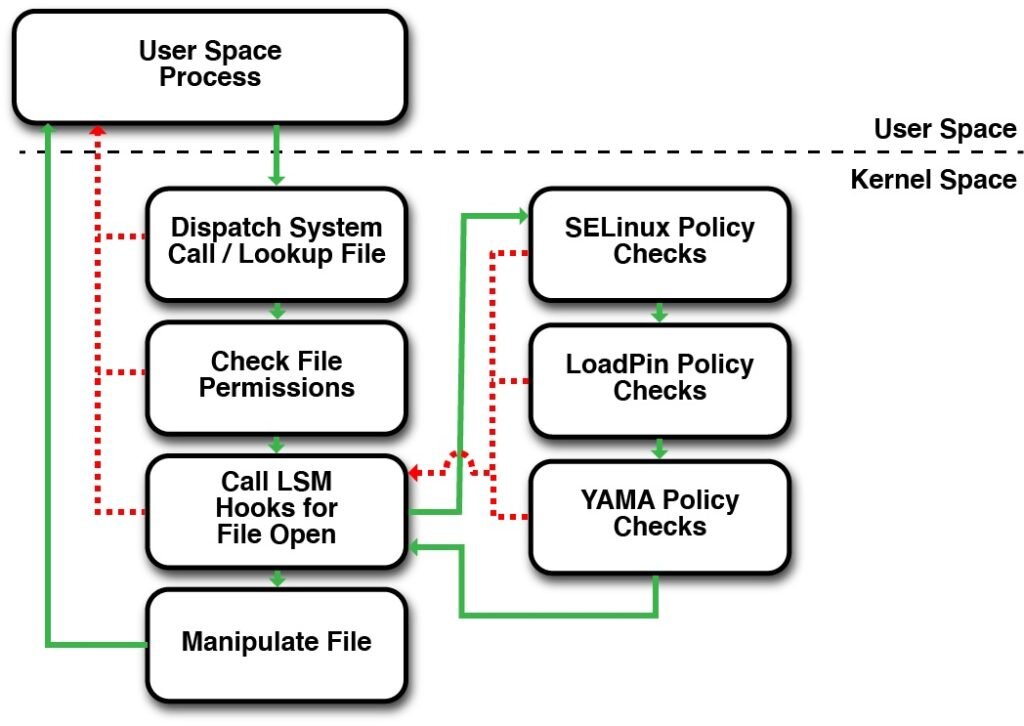
## How Many LSMs?

In kernel version 5.4, there are eight LSMs — *SELinux*, *SMACK*, *AppArmor*, *TOMOYO*, *Yama*, *LoadPin*, *SafeSetID*, and *Lockdown*.

Why there is such a variety of LSM? Well, some solve a common problem in unique ways while others address specific problems.

## How LSMs Help Protect System?

The current LSM framework allows a user to compile into the kernel **multiple LSMs that can then be stacked and used simultaneously**. The diagram below shows a call flow for an [open()](http://man7.org/linux/man-pages/man2/open.2.html) where three LSMs have registed hooks:



An LSM's place in an open system call

1. A process in user space calls open() on a file path.
2. The system call is dispatched and the path string is used to obtain a kernel file object. If the parameters are incorrect, an error is returned.
3. The [Discretionary Access Control](#_Discretionary_Access_Control) (DCA) file permissions are checked. Does the current user have permission to open the requested file? If not, the system call is terminated and an error is returned to the user.
4. If DACs are satisfied, the LSM framework calls each of the file\_open hooks for the enabled LSMs. The system call is terminated and an error returned to the user if a single LSM hook returns an error.
5. Finally, if all the security checks pass, the file is opened for the process, and a new file descriptor is returned to the process in user space.

## SELinux

SELinux (Security Enhanced Linux) is the default MAC implementation on Read Hat Linux distributions. It is known for being powerful and complex.

It's attribute-based which means the security identifiers for files are stored in **extended file attributes** in the file system. For example, you can use ls -Z to see the security context on /bin/bash. The four fields are separated by colons and are user:role:type:level.

$ ls -Z /bin/bash

-rwxr-xr-x. root root **system\_u:object\_r:shell\_exec\_t:s0** /bin/bash

The classic DAC permissions indicate that all users on the system are allowed to read and execute bash. But with SELinux, a security administrator can further specify the subjects that are allowed to execute or read files of type shell\_exec\_t in policy files. It seems reasonable that a security engineer may not want to permit a web server to execute a shell in case the web server is vulnerable to a remote code execution attack.

For more detailed info on how to use it, check [Red Hat’s SELinux User’s and Administrator’s Guide](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/selinux_users_and_administrators_guide/index).

## SMACK

### What Is SMACK?

SMACK (Simplified Mandatory Access Control) is debuted in 2008, specially designed for embedded systems. It's similar to SELinux but with a simpler approach to access control. It’s still a MAC system, meaning it is governed by a central policy and not by system users.

### Concepts

SMACK consists of three components: a SMACK-enabled kernel, SMACK utilities, and the configuration data policy.

There are 4 terms in SMACK that are especially important:

|  |  |
| --- | --- |
| **Subject** | An **active entity** on the computer system.  Examples: An app which is in turn the basic unit of execution. A running process. |
| **Object** | A **passive entity** on the computer system.  Examples: Apps, files, libs, IPC, etc. |
| **Access** | Any attempt by a **subject to put information into or get information from an object**. |
| **Label** | Data that identifies the MAC **characteristics of a subject or an object**. |

SMACK works **based on labels attached to objects**. Every task on a SMACK-based system is assigned a label. Special labels are assigned to system tasks such as init. Several special labels have specific meaning, such as the asterisk as a wild card character.

SMACK uses traditional UNIX/Linux access permissions such as read, write, and execute.

### Rules

A SMACK rule has the format: [subject-label] [object-label] [access(es)].

SMACK rules are very straightforward:

* Any access requested by a task labeled \* is denied.
* A read or execute access requested by a task labeled ^ is permitted.
* A read or execute access requested on an object labeled \_ is permitted.
* Any access requested on an object labeled \* is permitted.
* Any access requested by a task on an object with the same label is permitted.
* Any access requested that is explicitly defined in the loaded rule set is permitted.

Following is an example of SMACK rule config file:

$ cat /sys/fs/smackfs/load2

App::its Lib::mizaru rwxl # [subject-label] [object-label] [access(es)]

App::its Lib::v2xstack rwxl

App::its Log::tel rwxa

App::its App::v2xmain rw

App::its App::dlt rw

App::its System::dlt rwxt

App::its System::v2x rwx

App::its \_ rwxa

\_ App::health rwx

\_ App::app rwx

\_ Privileged::data rwx

Access legend:

|  |  |  |
| --- | --- | --- |
| **Notation** | **Name** | **Description** |
| -r | Read |  |
| -w | Write |  |
| -x | Execute |  |
| -a | Append |  |
| -t | Transmute | Affects how labels are assigned to newly created files within directories. When a directory has the transmute bit set, files created in that directory inherit the label of the directory rather than the creating process's label. |
| -l | Lock | Controls whether a subject can get a file lock on an object. File locks are used to manage concurrent access to files. |

Let's analyze the above example, focusing only on App::its rules – what the its application can access:

* Has read/write/execute/lock access to Lib::mizaru and Lib::v2xstack libraries .
* Has full access (read/write/execute/append) to Log::tel.
* Has read/write access to App::v2xmain and App::dlt.
* Has read/write/execute/transmit access to System::dlt.
* Has read/write/execute access to System::v2x.
* Has full access to the default label \_.

### Commands

|  |  |
| --- | --- |
| **Command** | **Description** |
| smack\_admin | Enter SMACK admin mode.  SMACK mode can be *enforced* or *permissive*. |
| smack-profile -p status | Check the SMACK mode. |
| smack-profile -p enable | Enter *permissive* mode – the mode you have all the permission with app, files |
| smack-profile -p disable | Exit permissive mode.  By default, *enforced* mode is enabled. So exiting permissive mode means switching to enforced mode. |

### Log

Log /data/logger/audit/audit.log shows what kind of permission is denied when running an app. Here you can find info about SMACK.

For example:

/ # cat /data/logger/audit/audit.log | grep smack

type=AVC msg=audit(1744891569.776:37): lsm=SMACK fn=smack\_inode\_permission action=denied **subject="App::v2xmain"** **object="App::ehsm"** requested=rw pid=6208 comm="PKIConnWk" name="sem.mHSM\_sem" dev="tmpfs" ino=21667

## AppArmor

## TOMOYO

## LoadPin

## Yama

## SafeSetID

## Lockdown

For intro of all above LSMs, check <https://www.starlab.io/blog/a-brief-tour-of-linux-security-modules>.

## Comparison Table

