LINUX BASIC COMMANDS

linuxjourney.com

Command line / Terminal

pwd: (present working directory)

syntax: pwd

Shows which directory you are currently in.

cd: (change directory)

syntax: cd </directory/location>

Change the current working directory to another specified directory.
Each time you interact with your command prompt, you are working within a directory.
There are two different ways to specify a path, with absolute and relative paths:

- Absolute path: This is the path from the root directory. The root directory is commonly shown as a slash. Every time your path starts with / it means you are starting from the root directory. For example, /home/pete/Desktop. (if the path starts with a slash (/) it is the absolute path to the directory)

- Relative path: This is the path from where you are currently in filesystem. If I was in location /home/pete/Documents and wanted to get to a directory inside Documents called taxes, I don't have to specify the whole path from root like /home/pete/Documents/taxes, I can just go to taxes/ instead.

When used without any argument, cd will take you to your home directory.

cd shortcuts:

- .. (parent directory). Takes you to the directory above your current.
- ../../ to move two levels up (move two parent directories up...or more..etc)
- ../<directory> move one level up and enter a directory in that level
- ~ (home directory). This directory defaults to your home directory
- ~/<directory> navigate to a directory inside your home directory
- - (previous directory). This will take you to the previous directory you were just at.

```
$ cd ..
$ cd ../../
$ cd ../Downloads
$ cd ~
$ cd ~/Documents
$ cd -
```

Directories with space in their names:

If the directory you want to change to has spaces in its name, you should either surround the path with quotes or use the backslash (\) character to escape the space:

```
cd 'dir with spaces'
cd dir\ with\ spaces
```

Command line / Terminal

ls: (list directories contents)

syntax: ls <options>

The ls command will list directories and files in the current directory by default, however you can specify which path you want to list the directories of.

- ls
- ls /deirectory

Also note that not all files in a directory will be visible, Filenames that start with . are hidden, you can view them however with the ls command and pass the -a flag to it (a for all)

- ls -a

show a detailed list of files in a long format. This will show you detailed information, starting from the left: file permissions, number of links, owner name, owner group, file size, timestamp of last modification, and file/directory name.

- ls -l
- ls -la (to show all hidden and in long format)

touch: (create files)

syntax: touch <filename.file>

Touch allows you to the create new empty files.
Touch is also used to change timestamps on existing files and directories

- touch filename.file
- touch /directory/location/filename.file

You can use touch to create more than one empty file at a time:

- touch file1 file2 Directory/file3

You can auto-generate filenames:

- touch filename{1..10}.txt
- touch filename{1..10}.txt Downloads/filename{1..5}.txt

file: (determine file type)

syntax: file <file> or file <options> <file>

file is used to recognize the type of a file. It is very useful when you need to find out the filetype of a file which does not have a file extension (filename extension) or with a file extension thats different from the actual filetype. In Linux, filenames aren't required to represent the contents of the file

You can also pass the multiple files at once to get type of given files:

- file doc text2

find the type of all files in a particular directory: (use * wildcard)

- file Downloads/*
- file *.txt
- file *

Command line / Terminal

cat: (concatenate file(s) and print to the standard output)

syntax: cat <file(s)> or cat <options> <file(s)>

cat command allows us to view contents of a file, create single or multiple files, concatenate files and redirect output in terminal or files.

cat the contents of a file into another file: (will create a new file if it doesn't already exist)

- cat file1 > file2
- cat file1 >> file2
- cat file1 | tee file2
- cat file1 | tee -a file2
- cat file1 | sudo tee file2
- cat file1 | sudo tee -a file2

The > is a redirection operator that allows us the change where standard output goes. It allows us to send the output of echo Hello World to a file instead of the screen. If the file does not already exist it will create it for us. However, if it does exist it will overwrite it.

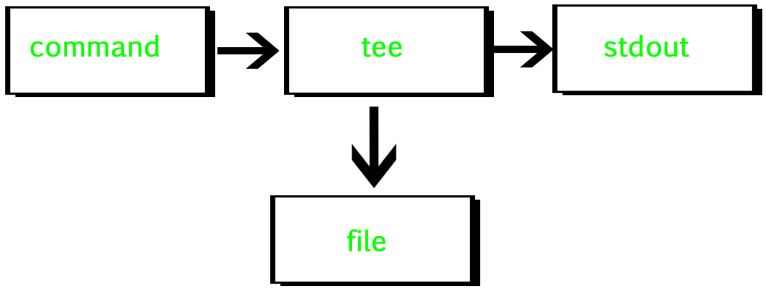
The >> redirection operator will append to the end of the file. if the file doesn't already exist it will create it for us. However, if it does exist it will append the output to the end of the file.

The | (pipe) allows us to get the stdout of a command and make that the stdin to another process. takes the output of a command and makes it the input of another command.

The tee command reads standard input (stdin) and writes it to both standard output (stdout) and one or more files. tee is usually part of a pipeline, and any number of commands can precede or follow it.

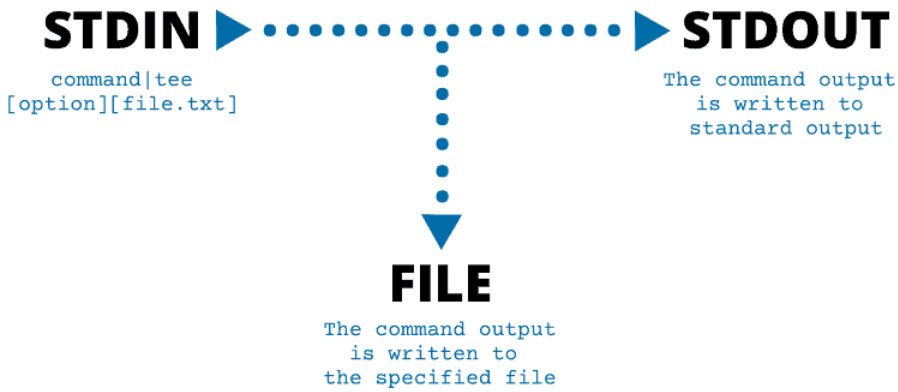
View contents of a file(s):

- cat file1
- cat file1 file2



Redirecting Multiple Files to a Single File:
(combining files)

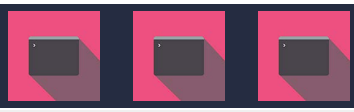
- cat file file1 file2 > file3
- cat file file1 file2 >> file3
- cat file file1 file2 | tee file3
- cat file file1 file2 | tee -a file3



Note: you can also use the command **echo to write to a file:

echo "text" > file
echo "text" >> file

Command line / Terminal



less: (view files/very large files)

syntax: less <file> or less <options> <file>

The text is displayed in a paged manner, so you can navigate through a text file page by page.
Use the following command to navigate through less:

- q - Used to quit out of less and go back to your shell.
- Page up, Page down, Up and Down - Navigate using the arrow keys and page keys.
- g - Moves to beginning of the text file.
- G - Moves to the end of the text file.
- /search - You can search for specific text inside the text document. Prefacing the words you want to search with /
- h - If you need help about how to use less while you're in less, use help.
- +F - monitor files in real time
- <#>g - move to the # line of the output... e.g. 6g moves to the 6th line
- you can view multiple files with less <file1> <file2> etc...

history: (display the history of the commands that you have previously entered)

syntax: history or history <options>

This program keeps track of everything that we type into the command line.
Allows you to review and repeat your previous commands.

Show a numbered list of all entered commands:

- history

Run the previous command:

- !!
- sudo !!

Running a command from the history list:

- !<#> ...e.g. !37 .. will run the 37th command

Delete a command from history:

- history -d <#>

Delete entire command history

- history -c

Search for command in history with grep:

- history | grep <search-command>

Search for a command in history interactively:

- in terminal ,press CTRL+R
  you can modify the desired command found in interactive mode by pressing left/right arrow keys

Command line / Terminal

cp: (copy)

syntax: cp <source> <destination> or cp <options> <source> <destination>

copy files and directories.
* if the destination doesn't exist, then first it creates it and content is copied to it, but if you copy a file over to a directory that has the same filenames, the file will be overwritten with whatever you are copying over. if you have a file that you don't want to get accidentally overwritten. You can use the -i flag (interactive) to prompt you before overwriting a file

Copy a file to another directory:

- cp file /destinsation/location/

Copy multiple files to a destination:
- cp file1 file2 file3 directory

You can copy multiple files and directories as well as use wildcards. A wildcard is a character that can be substituted for a pattern based selection, giving you more flexibility with searches. You can use wildcards in every command for more flexibility.

- * the wildcard of wildcards, it's used to represent all single characters or any string.
- ? used to represent one character
- [] used to represent any character within the brackets

e.g. $ cp *.jpg /home/pete/Pictures $ cp * /home/place/

Copy a directory to another location (-r flag):

- cp -r directory/ /other/directory/location

If you want to be prompted before overwriting a file, use the -i (interactive) option:

- cp -i mycoolfile /home/pete/Pictures

You can combine flags:

- cp -ir directory /other/directory
- cp -ir directory/* /other/directory

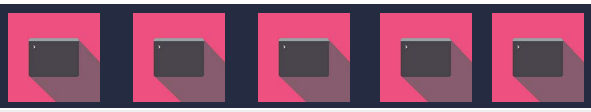
Update (-u flag), copy only when the SOURCE file is newer than the destination file or when the destination file is missing:

- cp -u source_file destination
- cp -iru source/directory/* destination/location

Archive (-a flag), copies files/directories and preserves ownership, time-stamps attributes.

- cp -a source destination
- cp -a source/* destination

Command line / Terminal



mv: (move)

syntax: mv <source> <destination>

Used for moving files, directories or/and also renaming them. if you mv a file or directory it will overwrite anything in the same directory with the same name, So you can use the -i flag to prompt you before overwriting anything.

move a file or/and directory to a different location:

- mv file_1 /dir/location/

Rename a file/directory:

- mv name_old name_new
- mv /place/name_old /dir/location/name_new

Prompt you for confirmation before overwiting anything named the same (-i flag):

- mv -i /place/* /other/place/
- mv -i file1 /location/

Do not overwrite anything (-n flag):

- mv -n TEST dir1/TEST

mkdir: (make directory)

syntax: mkdir <dir_name> or mkdir <options> <dir_name>

Used for creating directories. By defuault creates a directory in the current working directory, unless specified to create it in another directory location.

Create a directory:

- mkdir dir_name
- mkdir /location//dir_name

Create multiple directories at once:

- mkdir dir1 dir2 dir3

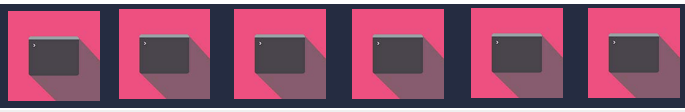
Create subdirectories/parent directories, within a newly created directory (-p flag):

- mkdir -p books/hemmingway/favorites

Create a directory and set the directories permissions simultaneously (-m flag):

- mkdir -m 777 dir_name
- mkdir -m a= rwx dir_name

Command line / Terminal



rm: (remove)

syntax: rm <file> or rm <options> <file>

rm command is used to remove or delete the files or directories. By default, the rm command will not remove the directories; you need to use the -r flag to delete directories.

Delete a file:

- rm file1

Delete a directory and all its contents (-r flag):

- rm -r dir1

Remove multiple files:

- rm file1 file2 file3

Prompt before every removal (-i flag):

- rm -i file
- rm -ir dir
- rm -ir /location/*

Force deletion of files or directories, whether they are write protected or not (-f flag):

- rm -f file
- rm -rf /location/*
- rm -rfi dir/*

find: (find files)

syntax: find [where to start searching from] [-options] [what to find]

Find files and directories in the system, and can perform subsequent operations on them. It supports searching by file, folder, name, creation date, modification date, owner and permissions. By using the '-exec' other UNIX commands can be executed on files or folders found.

find a file

- find /home -name file22

Command line / Terminal

locate: (find/search files)

syntax: locate <file_name>

locate command in Linux is used to find the files by name. locate works better and faster than find command counterpart because instead of searching the file system when a file search is initiated, it would look through a database.

By default, locate command does not check whether the files found in the database still exist and it never reports files created after the most recent update of the relevant database. If it doesn't find a file, it doesn't necessarily mean that the file does not exist on the system. The locate command works on an index i.e. a database of file locations. When you use the command, it refers to this index instead of searching the entire filesystem. This is the reason why locate is super quick at finding files in Linux.

Using locate command is dead simple. You just have to specify the file name:

- locate <filename>

perform a case-insensitive search (-i flag):

- locate -i <filename>

using **updatedb** to create your own index for locate command:

updatedb; updates file name databases used by GNU locate. The file name databases contain lists of files that were in particular directory trees when the databases were last updated. The file name of the default database is determined when locate and updatedb are configured and installed. The frequency with which the databases are updated and the directories for which they contain entries depend on how often updatedb is run, and with which arguments.

create an index of a specified directory (and its subdirectories and files), you can use the updatedb command like this:

- updatedb --localpaths='<desired_directory>' --output=<location_of_indexfile_name>

  e.g. updatedb --localpaths='/mnt/XTGO-EXT4' --output=/mnt/XTGO-EXT4/dbfile

where:

--localpaths='path1 path2...'
Non-network directories to put in the database. Default is /.

--output=dbfile
The database file to build. Default is system-dependent. In Debian GNU/Linux, the default is /var/cache/locate/locatedb.

Once you have created this index file, you can ask locate command to explicitly use this index:

- locate -d index_file file_name

Command line / Terminal

help: (help)

syntax: <command> --help

the help command displays information about commands.

e.g: Display a brief help description of the command cp.

- cp --help

man: (manual pages)

syntax: man <command>

man command in Linux is used to display the user manual of commands (commands that have a manual page available for it, most do). It provides a detailed view of the command and its usage.

e.g.: Display manual pages for mv command:

- man mv

whatis: (one line manual page description of a command)

syntax: whatis <command>

the “whatis” command is used to offer a one-line overview of command, option, or a flag
e.g. get a whatis description of mkdir command

- whatis mkdir

alias: (create a custom reference to another command)

syntax: alias <custom_name>='<custom-command>'

The shell alias is simply a way to reference another command. It can be used to avoid repetitive long typing of commands among other things.

There are many advantages of writing alias and keeping them in user property files like .bashrc and .profile file in users home directories. Some of the advantages of alias as listed below:

1) Time-saving: We can save a good amount of time by shortening the length of a command so that we dont need to type as many characters.
2) We dont need to remember bigger command, we just create a short name for it (alias) and use it.
3) We can run multiple commands (using command chaining ;) with just single alias.
4) We can use functions to make our alias more powerful, though we do not use alias command directly.

How to define an alias:

- alias a='cmd'
- alias a='cmd1;cmd2'

How to define an alias permanently:

The above-mentioned method will not keep your alias across reboots and logouts we have to make these alias available all the time. To do that we have to edit our shell profiles files(~/.bashrc for BASH shell) and give alias entries in it as shown below.

- $vi ~/.bashrc
  Give entries as below to bottom of ~/.bashrc file
- alias ll='ls -l'

Unset an alias (unalias):

- unalias <alias_name>

list currently defined aliases:

- alias

Command line / Terminal

apropos: (searches the manual pages names and descriptions for a keyword or regular expression.)

syntax: apropos <keyword> or apropos <options> <keyword>

Each manual page has a short description available within it. apropos searches the descriptions and names for instances of keyword.

examples:

- apropos compress
- apropos grub

env: (enviroment variables)

In Linux based systems environment variables are a set of dynamic named values, stored within the system that are used by applications launched in shells or subshells. In simple words, an environment variable is a variable with a name and an associated value.

Environment variables allow you to customize how the system works and the behavior of the applications on the system. For example, the environment variable can store information about the default text editor or browser, the path to executable files, or the system locale and keyboard layout settings.etc...

There are several commands available that allow you to list and set environment variables in Linux:

- env – The command allows you to run another program in a custom environment without modifying the current one. When used without an argument it will print a list of the current environment variables.
- printenv – The command prints all or the specified environment variables.
- set – The command sets or unsets shell variables. When used without an argument it will print a list of all variables including environment and shell variables, and shell functions.
- unset – The command deletes shell and environment variables.
- export – The command sets environment variables.

List enviroment variables:

The most used command to displays the environment variables is printenv, but you can also use echo $<env_name>.

For example, to display the value of the HOME environment variable with printenv you would run:

- printenv HOME

To display the current user in the terminal session ($USER) enviroment variable using echo:

- echo $USER

To display the directories of $PATH, PATH enviroment variable:

- echo $PATH
- printenv PATH

To see enviroment variables for current terminal session:

- env
- printenv

File Permissions

Linux File/Directory Permissions:

In Linux, file permissions, attributes, and ownership; control the access level that the system processes and users have to files. This ensures that only authorized users and processes can access specific files and directories.

For effective security, Linux divides authorization into 2 levels.

1. Ownership
2. Permission

The basic Linux permissions model works by associating each system file with an owner and a group and assigning permission access rights for three different classes of users.

Ownership of Linux Files:

Every file and directory on your Unix/Linux system is assigned 3 types of owner, given below:

- **user** (the file owner): user is the owner of the file. By default, the person who created a file becomes its owner. Hence, a user is also sometimes called owner. The ownership can be changed as well (chown command).
- **group** (the group members): A user- group can contain multiple users. All users belonging to a group will have the same Linux group permissions access to the file. Every user is part of a certain group(s). A group consists of several users and this is one way to manage users in a multi-user environment. The group ownership can be changes as well (chgrp command).
- **others** (everybody else): Any other user who has access to a file. This person has neither created the file, nor he belongs to a usergroup who could own the file. Practically, it means everybody else. Hence, when you set the permission for others, it is also referred as set permissions for the world.

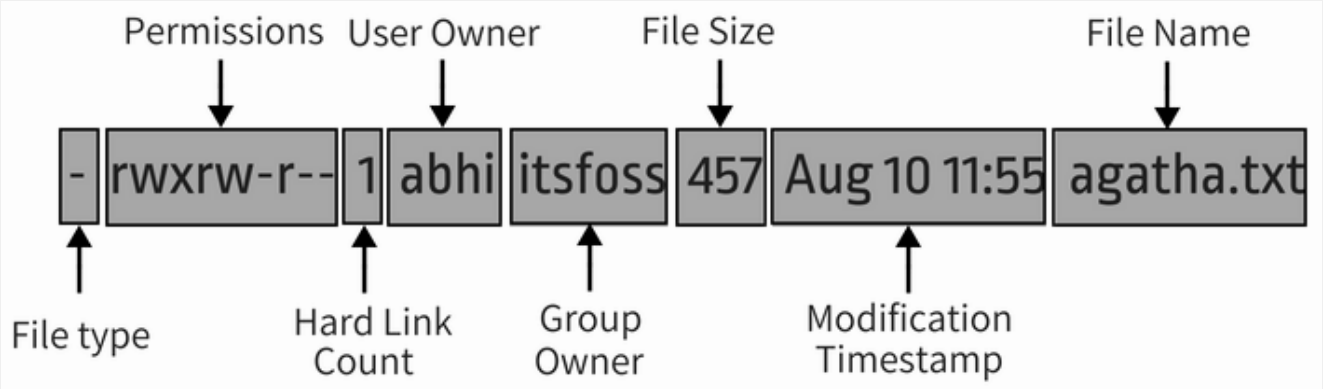
Permissions:

Every file and directory in your UNIX/Linux system has following 3 permissions defined for all the 3 owners discussed above.

- **read:** the read permission gives you the authority to open and read a file. Read permission on a directory gives you the ability to lists its content copy the files from directory.
- **write:** the write permission gives you the authority to modify the contents of a file. The write permission on a directory gives you the authority to add, remove and rename files stored in the directory. Consider a scenario where you have to write permission on file but do not have write permission on the directory where the file is stored. You will be able to modify the file contents. But you will not be able to rename, move or remove the file from the directory.
- **execute:** The execute permission: In Unix/Linux, you cannot run a program unless the execute permission is set. If the execute permission is not set, you might still be able to see/modify the program code (provided read & write permissions are set), but not run it.

Viewing file permisions:

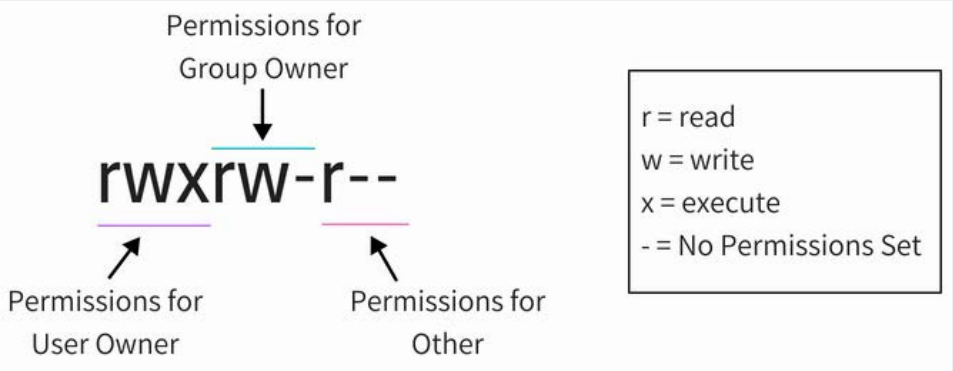
You can view the permissions of file(s)/directory(ies) by using the; **ls -l** or **ll** ;comands.



- File type: Denotes the type of file. d means directory, – means regular file, l means a symbolic link.
- Permissions: This field shows the permission set on a file.
- Hard link count: Shows if the file has hard links. Default count is one.
- User: The user who owns the files.
- Group: The group that has access to this file. Only one group can be the owner of a file at a time.
- File size: Size of the file in bytes.
- Modification time: The date and time the file was last modified.
- Filename: Obviously, the name of the file or directory.

File Permissions

In the previous example, you see the file permissions in three triplets of three characters each. The first triplet shows the owner permissions, the second one group permissions, and the last triplet shows everybody else permissions.



Permissions are always in the order of read, write and execute i.e., rwx. And then these permissions are set for all three kind of owners in the order of User, Group and Other.

Changing file/directory permissions (chmod command):

We can use the **chmod** command which stands for 'change mode', for changing the permissions on a file/directory in Linux. There are two ways to use the chmod command:

1. **Absolute mode**
2. **Symbolic mode**

I will only cover absolute mode in detail because its my preffered method.

Absolute (Numeric) Mode:

In the absolute mode, permissions are represented in numeric form (octal system to be precise).
file permissions are represented in a three-digit octal number.

- r (read) = 4
- w (write) = 2
- x (execute) = 1
- – (no permission) = 0

The table below gives numbers for all for permissions types.

Number	Permission Type	Symbol
0	No Permission	---
1	Execute	--x
2	Write	-w-
3	Execute + Write	-wx
4	Read	r--
5	Read + Execute	r-x
6	Read +Write	rw-
7	Read + Write +Execute	rwx

Using chmod to change permissions of a file/directory:

syntax: chmod <numeric_value#> file

- **chmod 777 file**

File Permissions

Changing ownership and group (chown command):

To change the ownership of a file/directory, you can use the chown command.

change the user ownership of a file using chown command:

- chown <username> <file>

change the user ownership as well as group of a file using chown command:
If you add a colon and groupname after the user you can set both the user and group at the same time.

- chown <username>:<group_name> <file>

change the user ownership of directories, their subdirectories and files; (-R option) operate on files and directories recursively.

- chown -R <username> <dir>
- chown -R <username>:<groupname> <dir>

Change group ownership (chgrp command):

To change the group-owner only, use the chgrp command.

change only the group owner of a file/directory, using chgrp command:

- chgrp <group> file

Block devices/Storage devices

Linux block devices, also referred to as storage devices:

lsblk: (list all block devices details or a specific block devices details)
syntax: lsblk or lsblk <specific_device>

lsblk can be used with options (lsblk <option>):

- **lsblk -f :**

Outputs info about filesystems. This option is equivalent to -o NAME,FSTYPE,LABEL,UUID,FSAVAIL,FSUSE%,MOUNTPOINT.

- **lsblk -f <block_device>:**

Outputs info about a specific filesystem. This option is equivalent to -o NAME,FSTYPE,LABEL,UUID,FSAVAIL,FSUSE%,MOUNTPOINT.

lsblk usage examples:

- lsblk /dev/sdX
- lsblk
- lsblk -f
- lsblk -f /dev/sdX

blkid: (block id)

syntax: blkid or blkid | grep sdX

locate/print block device attributes. In order to use blkid for a specific device we pipe (|) and grep the desired device.

e.g.

obtain blkid for /dev/sda:

- blkid | grep sda

fdisk :

syntax: fdisk -l or fdisk -l <device>

Display or manipulate a disk partition table. you can use fdisk to format block devices and to display block devices partition table information among other things.

List information for all block devices (-l option):

- fdisk -l

List information for a specific block device (-l option):

- fdisk -l <device>
- fdisk -l /dev/sdX

Block devices/Storage devices

udevadm info :

syntax : udevadm info <block_device>

udevadm info : Query the udev database for device information.

Basic usage example:

- udevadm info /dev/sdX

persistent block device naming:

There are four different schemes for persistent naming: **by-label, by-uuid**, **by-id** and **by-path**. For those using disks with GUID Partition Table (GPT), two additional schemes can be used **by-partlabel** and **by-partuuid**.

You can also use static device names by using Udev.
The directories in **/dev/disk/** are created and destroyed dynamically, depending on whether there are devices in them.

In order to list a devices, Persistent block device name:

- ll /dev/disk/by-label
- ll /dev/disk/by-uuid/
- ll /dev/disk/by-partlabel/
- ll /dev/disk/by-partuuid/

- ll /dev/disk/by-label | grep sdX
- ll /dev/disk/by-uuid/ | grep sdX
- ll /dev/disk/by-partlabel/ | grep sdX
- ll /dev/disk/by-partuuid/ | grep sdX

- ll /dev/disk/by-id/
- ll /dev/disk/by-path/

NOTE: **by-id creates a unique name depending on the hardware serial number, **by-path** depending on the shortest physical path (according to sysfs). Both contain strings to indicate which subsystem they belong to (i.e. pci- for by-path, and ata- for by-id), so they are linked to the hardware controlling the device. This implies different levels of persistence: the **by-path** will already **change** when the device is plugged into a different port of the controller, the **by-id will change** when the device is plugged into a port of a hardware controller subject to another subsystem.
Thus, **both are not suitable to achieve persistent naming tolerant to hardware changes.(by-id & by-path)**
However, both provide important information to find a particular device in a large hardware infrastructure. For example, if you do not manually assign persistent labels (by-label or by-partlabel) and keep a directory with hardware port usage, by-id and by-path can be used to find a particular device.