# Linux Basic

File Structure

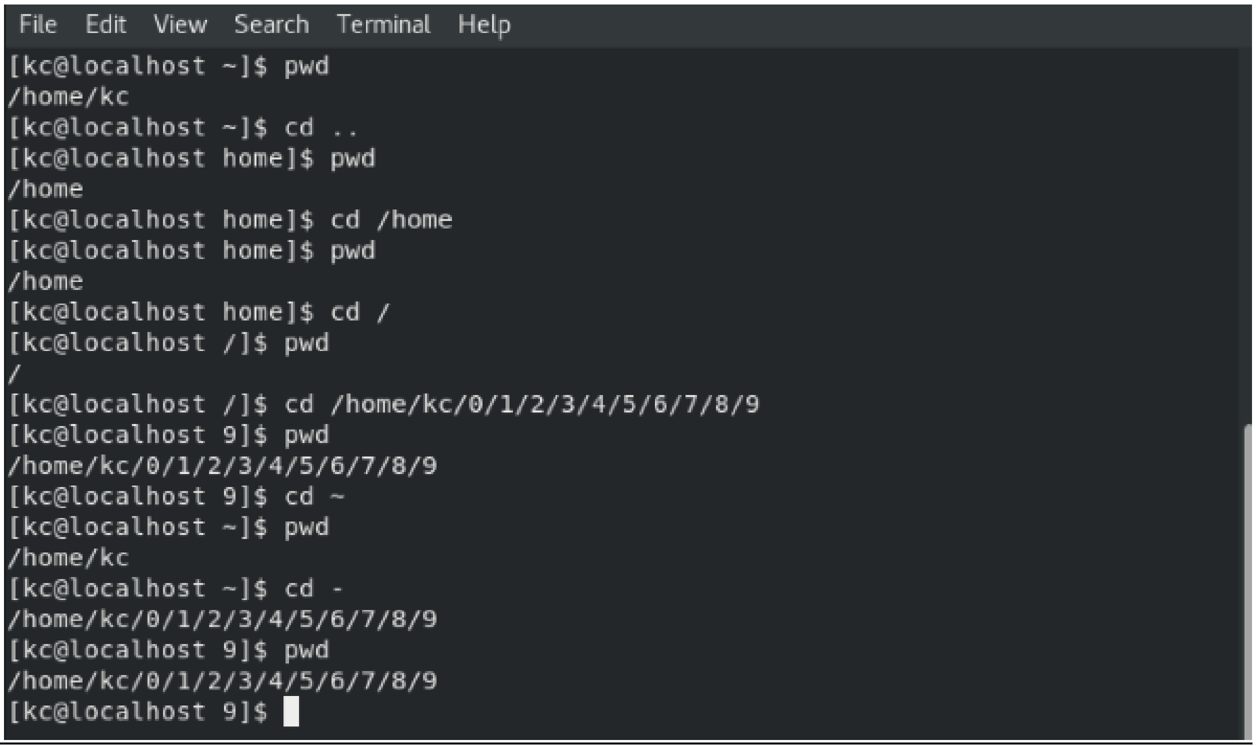
***************

| | |
|---|---|
| /bin | : User binaries (Common linux commnd need to use in single user mode) |
| /sbin | : System binaries (Commands under this is for system maintenance purpose used by system admins) |
| /etc | : Config files (etc: extended text configuration) |
| /dev | : Device files |
| /proc | : Process info (Contains running process info, Once system get reboot then all the related files will get clear and will create new info. because it retrives information from kernel and memory.) |
| /var | : Variable files (It contains logs) |
| /tmp | : Temporary files (Contains temporary files created by system and user, Once system get reboot then all the files will get clear) |
| /usr | : User programs (USR: Unix system resource. Contains binaries, libraries documenation files. Under it we have bin(Eg: less,awk) and sbin(Eg: cron,sshd) which contains binary files for user programs.) |
| /home | : Home directories |
| /boot | : Boot loader files(kernel files, grub files etc) |
| /lib | : System libraries(Contains library files that supports the binaries that are located in /bin and /sbin) |
| /opt | : Optional add-on apps(Contains add on installed application related files) |
| /mnt | : Mount directory (To mount any foreign device temporarily) |
| /media | : Removable device(To mount any removable device) |
| /srv | : Service data (Contains server specific service data) |

## Basic Symbols in Linux:

Before we get into commands, let's talk about important special characters. The dot (`.`) , dot-dot (`..`) , forward slash (`/`), and tilde (`~`), all have special functionality in the Linux filesystem:

- The dot (`.`) represents the current directory in the filesystem.
- The dot-dot (`..`) represents one level above the current directory.
- The forward slash (`/`) represents the "root" of the filesystem. (Every directory/file in the Linux filesystem is nested under the root `/` directory.)
- The tilde (`~`) represents the home directory of the currently logged in user.
- The dash (`-`) navigates back to the previous working directory, similar to how you can navigate to your user home directory with `~`. If you need to go back to our deeply nested directory `9` under your user home directory (this was my previous working directory), you would issue this command:



```
[kc@localhost ~]$ pwd
/home/kc
[kc@localhost ~]$ cd ..
[kc@localhost home]$ pwd
/home
[kc@localhost home]$ cd /home
[kc@localhost home]$ pwd
/home
[kc@localhost home]$ cd /
[kc@localhost /]$ pwd
/
[kc@localhost /]$ cd /home/kc/0/1/2/3/4/5/6/7/8/9
[kc@localhost 9]$ pwd
/home/kc/0/1/2/3/4/5/6/7/8/9
[kc@localhost 9]$ cd ~
[kc@localhost ~]$ pwd
/home/kc
[kc@localhost ~]$ cd -
/home/kc/0/1/2/3/4/5/6/7/8/9
[kc@localhost 9]$ pwd
/home/kc/0/1/2/3/4/5/6/7/8/9
[kc@localhost 9]$ 
```

## Some more basic symbols

| S:NO | Symbol | Explanation | Examples |
|---|---|---|---|
| 1 | / | The forward slash (/) represents the "root" of the filesystem. (Every directory/file in the Linux filesystem is nested under the root / directory.) / also use for directoty separation and path separation | / is a root directory<br>/home/user/samle/test.txt |
| 2 | ~ | is equal to the current user's home directlry. E.g: /home/someone/ | cd ~<br>ls ~ |
| 3 | * | A symbol which stands for "everything". Let's say you want to remove all the .jpg files from your Downloads folder which have their name starting with the "E" character, then you can use this symbol to represent all the other letters except E. See the example. | rm ~/Downloads/E*.jpg<br>ls /etc/*c<br>nano /var/log/nginx/* |
| 4 | & | Run a command in the background. It will return the PID of the newly running process to you and won't show you the output. | sudo apt update & |
| 5 | && | These symbols written together stand for "and". So if you want to run 2 commands together, you can use it. | sudo apt update && sudo apt upgrade |
| 6 | ¥ | Allows you to continue writing commands/Bash syntax in new line. | sudo ¥<br>apt ¥<br>update |
| 7 | .. | In many cases, especially in navigation, the two dots stand for the parent folder. | cd .. |
| 8 | . | In navigation or referring to files/folders, the dot stands for the current folder. | ls . |
| 9 | # | Everything after this symbol in the same line is considered to be a comment, so it won't be processed by the shell. | cd # This commands moves you somewhere. |
| 10 | \| | This is called "Piping", which is the process of redirecting the output of one command to the input of another command. Very useful and common in Linux/Unix-like systems. | cat /etc/profile \| grep bash |
| 11 | > | Take the output of a command and redirect it into a file (will overwrite the whole file). | ls ~ > output.txt |
| 12 | < | Read the contents of a file into the input of a command. | grep bash < /etc/profile |
| 13 | >> | Append a text or a command output into the last line of a file. | echo "First Line" > output.txt<br>echo "See this is the last line" >> output.txt |

- The dash (-) Also represent for file type as (-) rwx rwx rwx

# File Type

**(-)** : (Regular file) (Eg : text file,imagefile,binary...)

**d** : (Directory)

**c** : (Character device file) | (Character and block device files allow users and programs to communicate with hardware peripheral device.)server console is a character device file.Talks to devices in a character by character (1 byte at a time)

**b** : (Blockdevice file) | (Eg: storage LUN files)Talks to devices 1 block at a time ( 1 block = 512 bytes to 32KB)

**s** : (Local socket file used for communication between processes)

**p** : (Named Pipe)

**l** : (Symbolic link)

"file" command is used to identify the file type.

"stat" command is used to view details (Access, Modify,change time, size, inode, block ... )

"ln" command is used to make link between files.

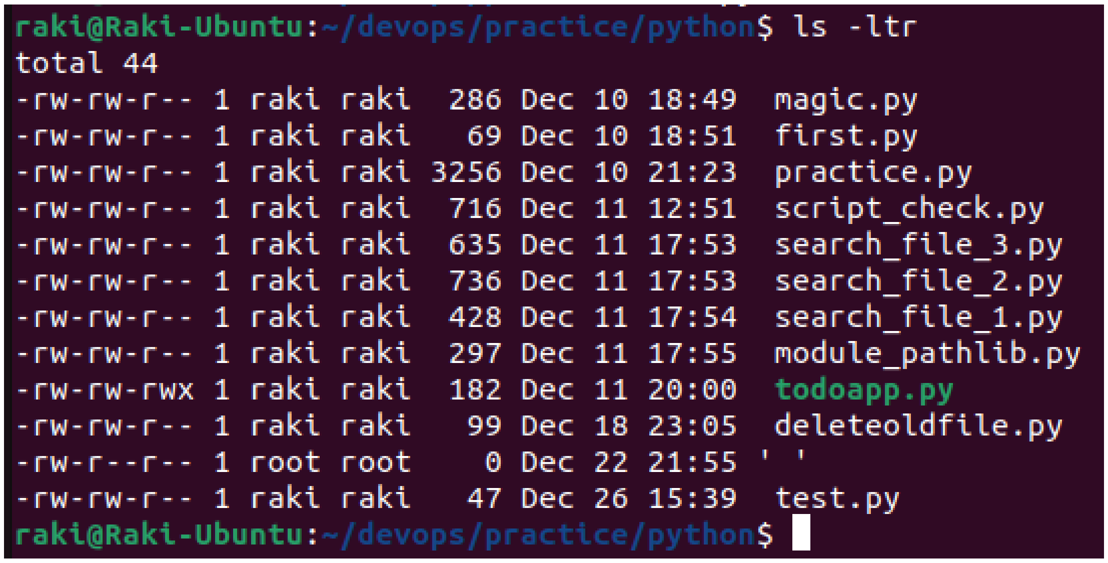
Syntax : ln <file> <hard ilnk> with -s option we can create soft link file .

Hard link : It acts like a mirror copy and share the same inode. When you delete hard link nothink will happen to the other file .Hard link can't create across the file system.Advantage of the hard link is it conumes 1 files space but if we delete any of the file then we have 1 more backup of the file, because both the files are sharing same indoe.

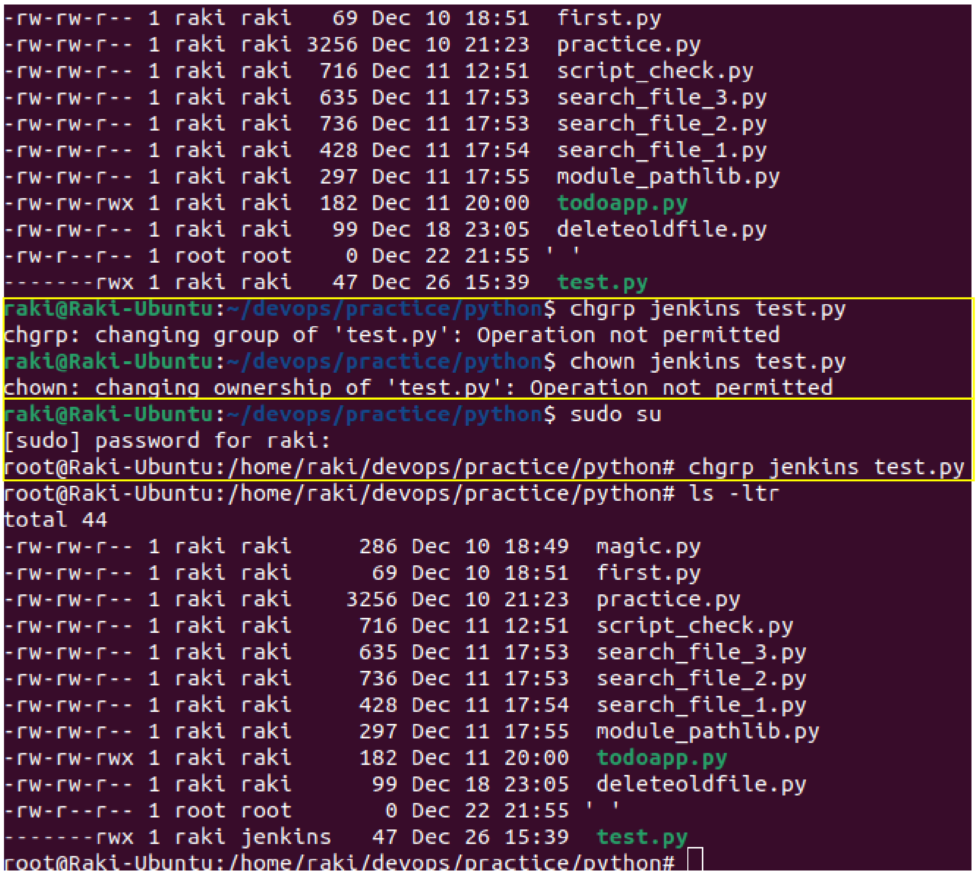
Soft link : It is actual link to the original file and it have a different inode. Soft link points to the original file so if original file is deleted then the soft link fails. If you delete soft link then nothing will happen.soft link can create across the file system because it have different indoe number.

dd command: It is used to take backup of any fs or disk and to check the disk. Eg:#dd if=/dev/sdc of=/tmp/sdc.bkp bs=1024 count=5

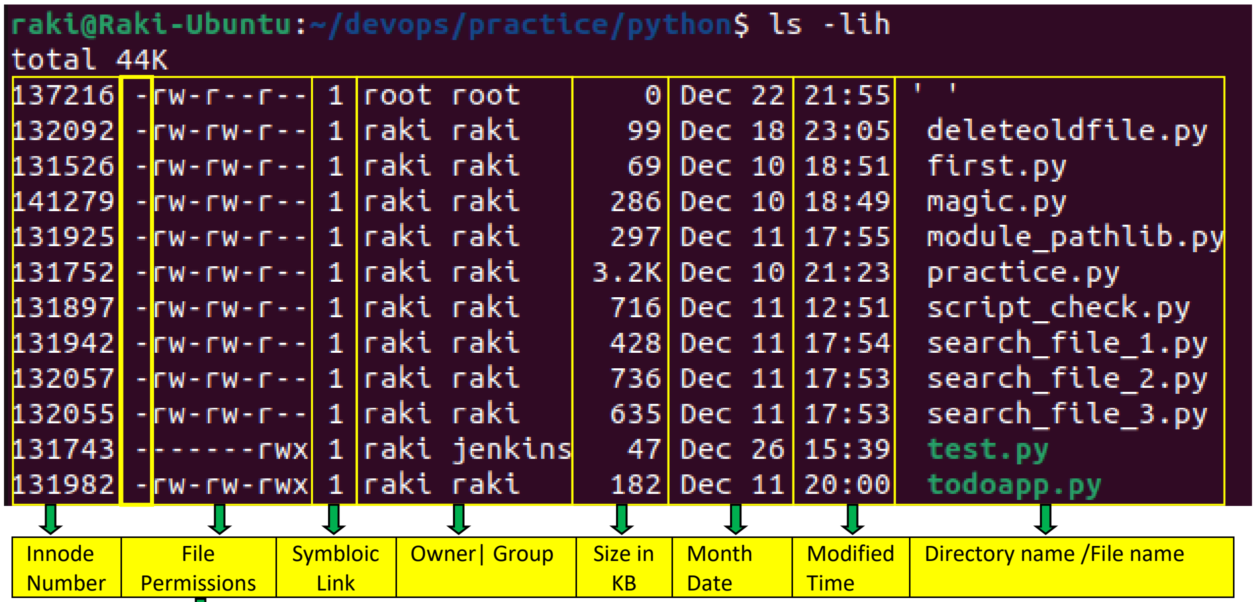
# File Permission



```
raki@Raki-Ubuntu:~/devops/practice/python$ ls -ltr
total 44
-rw-rw-r-- 1 raki raki  286 Dec 10 18:49  magic.py
-rw-rw-r-- 1 raki raki   69 Dec 10 18:51  first.py
-rw-rw-r-- 1 raki raki 3256 Dec 10 21:23  practice.py
-rw-rw-r-- 1 raki raki  716 Dec 11 12:51  script_check.py
-rw-rw-r-- 1 raki raki  635 Dec 11 17:53  search_file_3.py
-rw-rw-r-- 1 raki raki  736 Dec 11 17:53  search_file_2.py
-rw-rw-r-- 1 raki raki  428 Dec 11 17:54  search_file_1.py
-rw-rw-r-- 1 raki raki  297 Dec 11 17:55  module_pathlib.py
-rw-rw-rwx 1 raki raki  182 Dec 11 20:00  todoapp.py
-rw-rw-r-- 1 raki raki   99 Dec 18 23:05  deleteoldfile.py
-rw-r--r-- 1 root root    0 Dec 22 21:55 ' '
-rw-rw-r-- 1 raki raki   47 Dec 26 15:39  test.py
raki@Raki-Ubuntu:~/devops/practice/python$
```



```
-rw-rw-r-- 1 raki raki   69 Dec 10 18:51  first.py
-rw-rw-r-- 1 raki raki 3256 Dec 10 21:23  practice.py
-rw-rw-r-- 1 raki raki  716 Dec 11 12:51  script_check.py
-rw-rw-r-- 1 raki raki  635 Dec 11 17:53  search_file_3.py
-rw-rw-r-- 1 raki raki  736 Dec 11 17:53  search_file_2.py
-rw-rw-r-- 1 raki raki  428 Dec 11 17:54  search_file_1.py
-rw-rw-r-- 1 raki raki  297 Dec 11 17:55  module_pathlib.py
-rw-rw-rwx 1 raki raki  182 Dec 11 20:00  todoapp.py
-rw-rw-r-- 1 raki raki   99 Dec 18 23:05  deleteoldfile.py
-rw-r--r-- 1 root root    0 Dec 22 21:55 ' '
-------rwx 1 raki raki   47 Dec 26 15:39  test.py
raki@Raki-Ubuntu:~/devops/practice/python$ chgrp jenkins test.py
chgrp: changing group of 'test.py': Operation not permitted
raki@Raki-Ubuntu:~/devops/practice/python$ chown jenkins test.py
chown: changing ownership of 'test.py': Operation not permitted
raki@Raki-Ubuntu:~/devops/practice/python$ sudo su
[sudo] password for raki:
root@Raki-Ubuntu:/home/raki/devops/practice/python# chgrp jenkins test.py
root@Raki-Ubuntu:/home/raki/devops/practice/python# ls -ltr
total 44
-rw-rw-r-- 1 raki raki     286 Dec 10 18:49  magic.py
-rw-rw-r-- 1 raki raki      69 Dec 10 18:51  first.py
-rw-rw-r-- 1 raki raki    3256 Dec 10 21:23  practice.py
-rw-rw-r-- 1 raki raki     716 Dec 11 12:51  script_check.py
-rw-rw-r-- 1 raki raki     635 Dec 11 17:53  search_file_3.py
-rw-rw-r-- 1 raki raki     736 Dec 11 17:53  search_file_2.py
-rw-rw-r-- 1 raki raki     428 Dec 11 17:54  search_file_1.py
-rw-rw-r-- 1 raki raki     297 Dec 11 17:55  module_pathlib.py
-rw-rw-rwx 1 raki raki     182 Dec 11 20:00  todoapp.py
-rw-rw-r-- 1 raki raki      99 Dec 18 23:05  deleteoldfile.py
-rw-r--r-- 1 root root       0 Dec 22 21:55 ' '
-------rwx 1 raki jenkins   47 Dec 26 15:39  test.py
root@Raki-Ubuntu:/home/raki/devops/practice/python#
```



```
raki@Raki-Ubuntu:~/devops/practice/python$ ls -lih
total 44K
137216 -rw-r--r-- 1 raki root      0 Dec 22 21:55 ' '
132092 -rw-rw-r-- 1 raki raki     99 Dec 18 23:05  deleteoldfile.py
131526 -rw-rw-r-- 1 raki raki     69 Dec 10 18:51  first.py
141279 -rw-rw-r-- 1 raki raki    286 Dec 10 18:49  magic.py
131925 -rw-rw-r-- 1 raki raki    297 Dec 11 17:55  module_pathlib.py
131752 -rw-rw-r-- 1 raki raki   3.2K Dec 10 21:23  practice.py
131897 -rw-rw-r-- 1 raki raki    716 Dec 11 12:51  script_check.py
131942 -rw-rw-r-- 1 raki raki    428 Dec 11 17:54  search_file_1.py
132057 -rw-rw-r-- 1 raki raki    736 Dec 11 17:53  search_file_2.py
132055 -rw-rw-r-- 1 raki raki    635 Dec 11 17:53  search_file_3.py
131743 -------rwx 1 raki jenkins  47 Dec 26 15:39  test.py
131982 -rw-rw-rwx 1 raki raki    182 Dec 11 20:00  todoapp.py
```

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| -/d | r | w | - | r | - | x | r | - | - |
| File type | Owner Permissions | | | Group Permissions | | | Other user Permission- Not in Group | | |
| | 4 +2 + 0 = 6 | | | 4 + 0 + 1 = 5 | | | 4 + 0 + 0 = 4 | | |

| Symbolic | Mode | Absolute Mode |
|---|---|---|
| r | -read | 4 |
| w | -write | 2 |
| x | -execute | 1 |
| (-) | Null | 0 |

- Highest permission is –7—(4+2+1)
- Maximum permission - 777, but effective 666 incase of a file for security reason in linux, no user will get execute permission.
- For directory case effective permission is 755
- Lowest permission is –000 not recomended
- Minimum permission – effective permission 644 incase of a file * [ ]
- For directory case default permission is execute permission.

Each of the three permissions are assigned to three defined categories of users. The categories are:

- owner — The owner of the file or application.
- "chown" is used to change the ownership permission of a file or directory.
- group — The group that owns the file or application.
- "chgrp" is used to change the gropu permission of a file or directory.
- others — All users with access to the system. (outised the users are in a group)
- "chmod" is used to change the other users permissions of a file or directory.

# Special Permissions

SUID (Set User Id): If SUID is set on an executable file and a normal user execute it then the process will have same rights as the owner of the file being execute instead of the normal user.(Eg: password command)

```
[root@node2 ~]# ls -ltr /usr/bin/passwd      ## passwd command is suid set so any non-root user is able to change there own password , as its execute as root .
```

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| - | rws | r-x | r-x | . | 1 | root | root | 30768 | Feb 17  2012   /usr/bin/passwd |

```
[root@node2 tmp]# ls -ltr test
```

```
-rw-r--r--. 1 root root 0 Feb 16 11:26 test   ## 1st field is for file type, next 3*3 is for access permissions (user,group and other), link count, owner of the file, group owner, size, date and file name.
```

```
[root@node2 tmp]# chmod u+s test           ## suid applied on user permission so u+s.
```

```
[root@node2 tmp]# ls -ltr test
```

```
-rwSr--r--. 1 root root 0 Feb 16 11:26 test   ## "S" denotes suid applied on the file .
```

```
[root@node2 tmp]# chmod u+x test
```

```
[root@node2 tmp]# ls -ltr test
```

```
-rwsr--r--. 1 root root 0 Feb 16 11:26 test   ## If user have execute right and suid both then instead of "S"(Capital) it will show "s".
```

```
[root@node2 tmp]# chmod 4744 test          ## This is absolute method the first field(hear 4) is for special permission(hear suid) and remain are for user,group and other .
```

4-SUID

2-SGID

1-STICKY BIT

SGID(Set Group Id) : If SGID is set on any directory, all subdirectory and files created inside will get same group ownership as the main directory, it doesn't matter who is creating.

Owner of the file will be the file creator and group owner will inherit from the directory.

```
[root@node2 tmp]# mkdir test1
```

```
[root@node2 tmp]# ls -ld test1
```

```
drwxr-xr-x. 2 root root 4096 Feb 16 11:57 test1
```

```
[root@node2 tmp]# chmod g+s test1          ## sgid applied on group permission so g+s(We can use g+2 also).
```

```
[root@node2 tmp]# ls -ld test1

drwxr-sr-x. 2 root root 4096 Feb 16 11:57 test1   ## Here is "s" is on group execute filed.
```

STICKY BIT : it is used on folders in order to avoid deletion of a folder and its content by other users though they having write permissions on the folder contents. Except owner and root user, No one can delete other users data in this folder(Where sticky bit is set). though other users have full permissions.

```
[root@node2 tmp]# chmod o+t test1            ## It applid on other permission so o+t.

[root@node2 tmp]# ls -ld test1

drwxrwxrwt. 2 root root 4096 Feb 16 11:57 test1   ## Hear "t" is for sticky bit . As execute permission is there for other so its
showing "t" instead of "T".
```

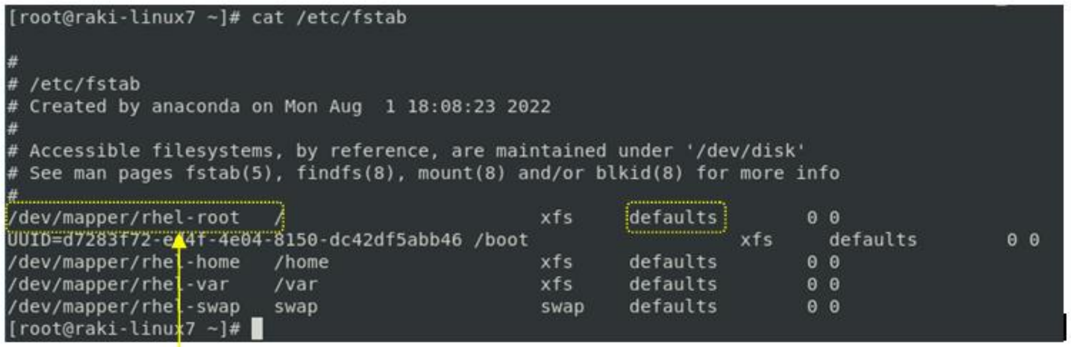
# Controlling access to files with ACLs (Access control list)

**************************************************************

Standard file permissions are satisfying when files are used by only a single ownner and a single designated group. However, If we want to give access to a user or group which not listing on default file permission, then ACL will come in use.

With ACL, you can grant permission to multiple users and groups, identified by user name, group name, UID, GID. using the same permission flags used with regular file permission: read, write and execute.

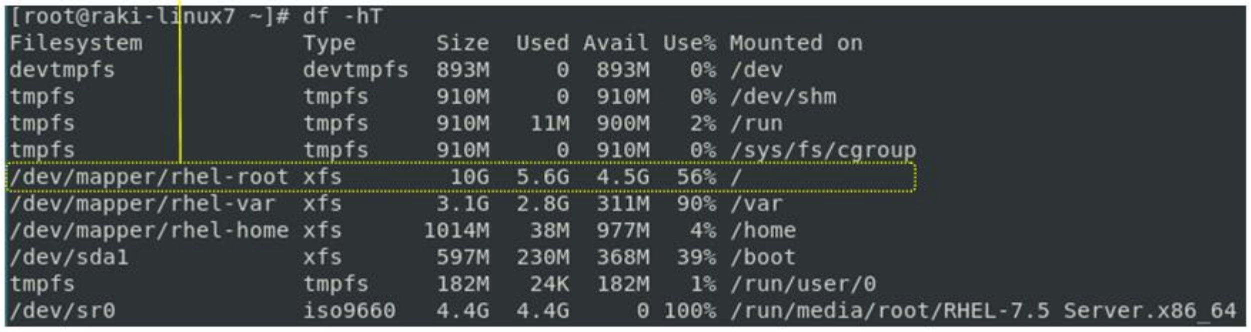
To check ACL is enable in our file system or not type cat /etc/fstab, where if mounted file system is defaults means ACL is enable in our file system.



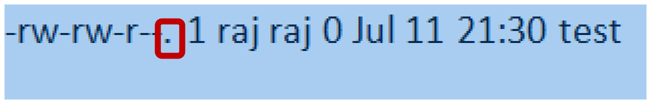
```
[root@raki-linux7 ~]# cat /etc/fstab

#
# /etc/fstab
# Created by anaconda on Mon Aug  1 18:08:23 2022
#
# Accessible filesystems, by reference, are maintained under '/dev/disk'
# See man pages fstab(5), findfs(8), mount(8) and/or blkid(8) for more info
#
/dev/mapper/rhel-root   /                       xfs     defaults        0 0
UUID=d7283f72-e44f-4e04-8150-dc42df5abb46 /boot                   xfs     defaults        0 0
/dev/mapper/rhel-home   /home                   xfs     defaults        0 0
/dev/mapper/rhel-var    /var                    xfs     defaults        0 0
/dev/mapper/rhel-swap   swap                    swap    defaults        0 0
[root@raki-linux7 ~]#
```

To check the file system → df –hT



```
[root@raki-linux7 ~]# df -hT
Filesystem            Type      Size  Used Avail Use% Mounted on
devtmpfs              devtmpfs  893M     0  893M   0% /dev
tmpfs                 tmpfs     910M     0  910M   0% /dev/shm
tmpfs                 tmpfs     910M   11M  900M   2% /run
tmpfs                 tmpfs     910M     0  910M   0% /sys/fs/cgroup
/dev/mapper/rhel-root xfs        10G  5.6G  4.5G  56% /
/dev/mapper/rhel-var  xfs       3.1G  2.8G  311M  90% /var
/dev/mapper/rhel-home xfs      1014M   38M  977M   4% /home
/dev/sda1             xfs       597M  230M  368M  39% /boot
tmpfs                 tmpfs     182M   24K  182M   1% /run/user/0
/dev/sr0              iso9660   4.4G  4.4G     0 100% /run/media/root/RHEL-7.5 Server.x86_64
```



```
-rw-rw-r--. 1 raj raj 0 Jul 11 21:30 test
```

This red marked (.) is denote for special permission and ACL.

[raj@master ~]$ touch test

[raj@master ~]$ ls -ltr test                                    ## test file owner is raj user and group is raj.

-rw-rw-r--. 1 raj raj 0 Jul 11 21:30 test

[raj@master ~]$ getfacl test

# file: test

# owner: raj

# group: raj

user::rw-

group::rw-

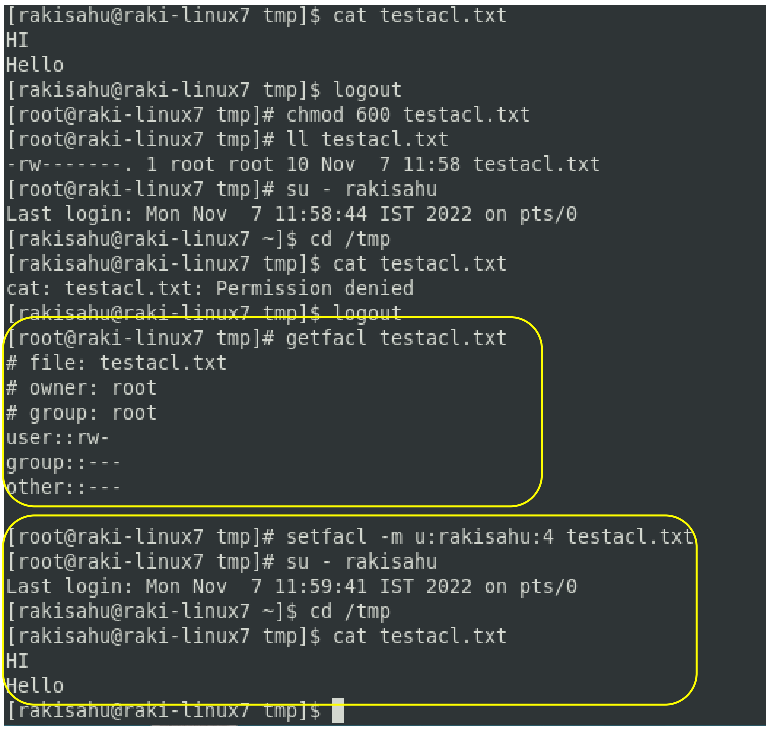
other::r--

[raj@master ~]$ setfacl -m u:vagrant:rw test          ## Hear we have set read and write permission for *vagrant* user, who dont have access to the file before.

"m" for modify  "u,g,o are for user, group and other" "r,w,x is for read, write and execute".

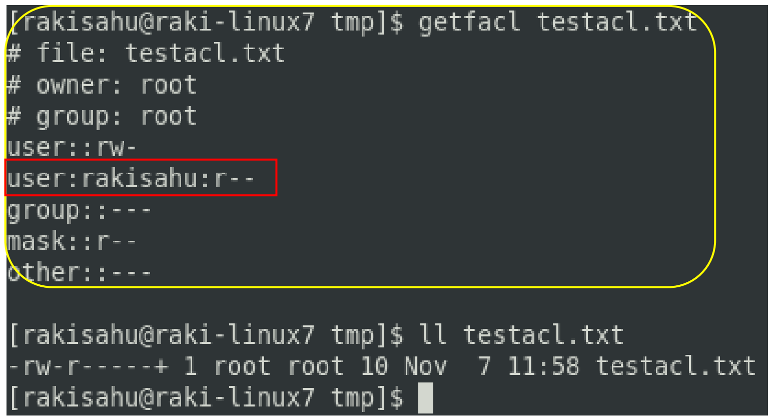
[raj@master ~]$ ls -ltr test

-rw-rw-r--+ 1 raj raj 0 Jul 11 21:30 test     ## If ACL set to any file, then "+" sign will come on "ls -ltr" output.



```
[rakisahu@raki-linux7 tmp]$ cat testacl.txt
HI
Hello
[rakisahu@raki-linux7 tmp]$ logout
[root@raki-linux7 tmp]# chmod 600 testacl.txt
[root@raki-linux7 tmp]# ll testacl.txt
-rw-------. 1 root root 10 Nov  7 11:58 testacl.txt
[root@raki-linux7 tmp]# su - rakisahu
Last login: Mon Nov  7 11:58:44 IST 2022 on pts/0
[rakisahu@raki-linux7 ~]$ cd /tmp
[rakisahu@raki-linux7 tmp]$ cat testacl.txt
cat: testacl.txt: Permission denied
[rakisahu@raki-linux7 tmp]$ logout
[root@raki-linux7 tmp]# getfacl testacl.txt
# file: testacl.txt
# owner: root
# group: root
user::rw-
group::---
other::---

[root@raki-linux7 tmp]# setfacl -m u:rakisahu:4 testacl.txt
[root@raki-linux7 tmp]# su - rakisahu
Last login: Mon Nov  7 11:59:41 IST 2022 on pts/0
[rakisahu@raki-linux7 ~]$ cd /tmp
[rakisahu@raki-linux7 tmp]$ cat testacl.txt
HI
Hello
[rakisahu@raki-linux7 tmp]$
```



```
[rakisahu@raki-linux7 tmp]$ getfacl testacl.txt
# file: testacl.txt
# owner: root
# group: root
user::rw-
user:rakisahu:r--
group::---
mask::r--
other::---

[rakisahu@raki-linux7 tmp]$ ll testacl.txt
-rw-r-----+ 1 root root 10 Nov  7 11:58 testacl.txt
[rakisahu@raki-linux7 tmp]$
```

[raj@master ~]$ getfacl test

# file: test

# owner: raj

# group: raj

user::rw-

user:vagrant:rw- ## Now vagrant user have read and write permission to this file.

group::rw-

mask::rw- ## Before ACL set on this file mask was not their. As we set rw permission for vagrant user, so mask set as rw. we can change the mask value too.

other::r--

[raj@master ~]$ setfacl -m u:vagrant:rwx test

[raj@master ~]$ getfacl test

# file: test

# owner: raj

# group: raj

user::rw-

user:vagrant:rwx

group::rw-

mask::rwx

other::r--

[raj@master ~]$ setfacl -m mask:r test ## Hear we are going to change the mask value as read.

[raj@master ~]$ getfacl test

```
# file: test

# owner: raj

# group: raj

user::rw-

user:vagrant:rwx  #effective:r--

group::rw-     #effective:r--

mask::r--

other::r--
```

If we will change the mask value to lower of what additional user, group and other have then only the lower value will effect.

On above vagrant user have full permission but as the mask is set only for read, so he can do only read. Same applicable for group members and others too.

```
[raj@master ~]$ mkdir testdir

[raj@master ~]$ getfacl testdir/

# file: testdir/

# owner: raj

# group: raj

user::rwx

group::rwx

other::r-x
```

[raj@master ~]$ setfacl -d -m u:vagrant:rw testdir/                ## Hear we have set default acl with -d to a directory. So that all the files or directory created under it will inherit the ACL permission.

```
[raj@master ~]$ getfacl testdir/

# file: testdir/

# owner: raj

# group: raj

user::rwx

group::rwx

other::r-x

default:user::rwx

default:user:vagrant:rw-
```

```
default:group::rwx

default:mask::rwx

default:other::r-x

[raj@master ~]$ cd testdir/

[raj@master testdir]$ touch test1

[raj@master testdir]$ getfacl test1

# file: test1

# owner: raj

# group: raj

user::rw-

user:vagrant:rw-

group::rwx     #effective:rw-

mask::rw-

other::r--
```

After creation of test1 file under the directory by default vagrant user also got access, as the parent directory is set with ACL.

```
[raj@master ~]$ setfacl -x vagrant test    ## To remove ACL permission for a user.

[raj@master ~]$ setfacl -b test                              ## To remove complete ACL from a file.
```

To Be Continued…….