Linux: Overview of Commands and Permissions

Created By - Abhinav Badola

Audience

- Linux Beginners
- Linux Intermediate Users
- People who want to develop better understanding of working of commands, permissions

Objective

- Develop a familiarity with the linux commands.
- Develop understanding about the semantics of linux commands
- Develop understanding about the various details provided by `ls -l`
- Understanding permissions on Linux

Materials

- Linux Command Structure
- Generic linux commands
- ls
- Permissions

Linux Commands

Types of Commands

- Internal Commands (Shell Built In Commands)
  - Built in commands provided by the shell
  - Not dependent on additional source/package
  - Bash Built In Commands: http://www.gnu.org/software/bash/manual/bashref.html#Shell-Builtin-Commands
  - Eg - `cd`, `umask`, `test`, (etc.)
- External Commands
  - Utilities that help perform a particular task
  - Not part of the core shell, but mostly all commands are provided with standard linux distro
  - Eg - `ls`, `cat`, `head`, `tail`, `grep` (etc.)

Command Structure

- <command> <options/switches> <source> <target>
  - Examples -
    - `ls -d  test_directory`
    - `cp -rf source_file    source_file_copy`
    - `mv     abc.mp3        bcd.mp3`
- If the <source> is omitted, then the current working directory is generally assumed to be source.
  - Example -
    - `ls -ltr     # this assumes that <source> = current working directory`
- <target> and <options/switches> are optional

Command Options

- Single character options
  - Need to start with - (single hyphen)
    - `ls -l -r       # we have applied 2 options to ls command`
  - Can be chained together
    - `ls -lrt        # expands into   =>   ls -l -r -t`
- Multi-character options
  - Generally need to be preceded by -- (double hyphen)
    - `ls --color`
  - There are some exceptions, when it is preceded by - (single hyphen)
    - `find . -name “*.log”`

ls (list directory contents)

- ls is the most used command in linux and provides a lot of information, that is essential in day-to-day debugging issues.
- We will analyze ls in proper depth, because it will cover a lot of essential topics required for better understanding of how things work in linux.
- Options
  - `-l      => list in long format`
  - `-a      => include all/hidden files`
  - `-r      => display in reverse order`
  - `-R      => recursively (go inside all subdirectories)`
  - `-h      => display size in human readable format, used with -l`
  - `-d      => display the details of the directory, instead of its content`
  - `-t      => sort according to last modification timestamp`
  - `--color => show color for different file types`

ls (list directory contents)

```
$ mkdir sample_directory           # so let us make a sample directory, using mkdir command
$ cd sample_directory              # get inside that directory, using cd command
$ touch test_file                  # create a file, using touch command
$ mkdir test_directory             # create a directory, for testing purpose
$ ln -s test_file link_file        # create a link file, for testing purpose
$ cd ..                            # go back to previous directory. Remember relative paths?

$ ls sample_directory/             # oops, it listed the contents insteads, we wanted to know about directory itself
test_directory test_file link_file

$ ls -d sample_directory/          # now it is telling about directory, but not the details
sample_directory/

$ ls -ld sample_directory/         # finally we got something right, the details of the directory
drwxr-xr-x  4 abhinav  staff  170 Mar 10 00:41 sample_directory/
```

Lets understand what `ls -l` provides, and how to exploit its information...

ls -l (long-list contents)

```
$ ls -ld sample_directory          # so what does the below info tell ?
drwxr-xr-x  3 abhinav  staff  170 Mar 10 00:41 sample_directory
$ ls -l sample_directory
lrwxr-xr-x  1 abhinav  staff   9 Mar 10 01:29 link_file -> test_file
drwxr-xr-x  2 abhinav  staff  68 Mar 10 01:14 test_directory
-rw-r--r--  1 abhinav  staff   0 Mar 10 00:40 test_file
```

Let us divide the highlighted line into groups and analyze each in detail :

- `d                => file type`
- `rwxr-xr-x        => permissions on file`
- `4                => link count of the file`
- `abhinav          => user/owner of the file`
- `staff            => group of the file`
- `170              => size of the file`
- `Mar 10 00:41     => last modification timestamp`
- `sample_directory => name of the file`

File Types In Linux

```
$ ls -l sample_directory
lrwxr-xr-x  1 abhinav  staff   9 Mar 10 01:29 link_file -> test_file
drwxr-xr-x  2 abhinav  staff  68 Mar 10 01:14 test_directory
-rw-r--r--  1 abhinav  staff   0 Mar 10 00:40 test_file
```

- In Linux, everything is a file.
- Your hard-disk, cpu, ram, directory, network, everything the OS interacts with, is a file.
- Directory is a special kind of file, which contains information about other files (that it stores within, which in turn may again be file of any type)
- This makes it necessary to classify files in various types to understand which class of operations are applicable on a particular file. Eg - recursive option only makes sense, when we are applying it on a directory.
- In most of our further slides, we will use the word **file/files** to represent all type.

File Types In Linux

There are 7 types of files in Linux, and the **first character** in `` `ls -l` `` denotes that information -

- Normal Files
    - `d => directory: a file containing information about other files`
    - `- => normal/regular file: text file, ASCII, compiled, binary data`
    - `l => soft-link: a file pointing to another file`
- Device Files
    - `c => char device: communicates via sending/receiving one character at a time. Eg - sound card, serial ports, etc.`
    - `b => block device: communicates via sending/receiving blocks of data. Eg - hard-disk, usb camera, etc.`
- System/Networking Files
    - `s => socket file`
    - `P => pipe file`

Size Of The File

```
$ ls -l sample_directory
lrwxr-xr-x  1 abhinav  staff   9 Mar 10 01:29 link_file -> test_file
drwxr-xr-x  2 abhinav  staff  68 Mar 10 01:14 test_directory
-rw-r--r--  1 abhinav  staff   0 Mar 10 00:40 test_file
```

- In Linux everything is a file, hence the size shown by `ls` command is also the size of the file.
- This means that w.r.t. directory, it is the size of the directory(as a file) and **not** the size of underlying/contained items(which is generally the intended purpose).
- Similarly, the link file’s size is the size of the file itself(i.e. *link_file*), and not the file it is pointing to(*test_file*).

Modification Timestamp

```
$ ls -l sample_directory
lrwxr-xr-x  1 abhinav  staff   9 Mar 10 01:29 link_file -> test_file
drwxr-xr-x  2 abhinav  staff  68 Mar 10 01:14 test_directory
-rw-r--r--  1 abhinav  staff   0 Mar 10 00:40 test_file
```

- Linux stores three timestamps for a file -
  - `mtime` — updated when the file contents change. This is the "default" file time in most cases. This is the timestamp shown by `ls -l`
  - `ctime` — updated when the file *or* its metadata (owner, permissions) change. Way to see `ls -l --time=ctime`
  - `atime` — updated when the file is read/modified. Ways to see `ls -l --time=atime` or `ls -lu`
- Creation time of a file is generally not stored by linux filesystem architecture.
- Also, note that access times are by inode, not by filename, so if you have hard-links, reading from one will update all names that refer to the same file.

Link Count

```
$ ls -l sample_directory
lrwxr-xr-x  1 abhinav  staff   9 Mar 10 01:29 link_file -> test_file
drwxr-xr-x  2 abhinav  staff  68 Mar 10 01:14 test_directory
-rw-r--r--  1 abhinav  staff   0 Mar 10 00:40 test_file
```

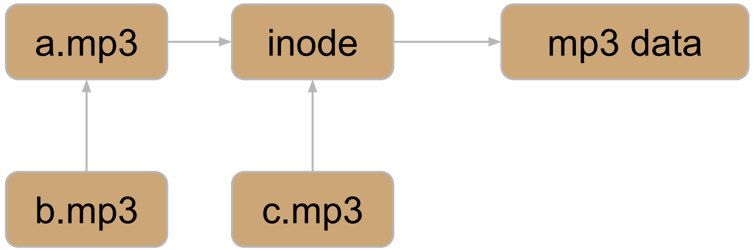
So first we need to understand hard-link and soft-link.

```
$ touch a.mp3                # create a file a.mp3
$ ln -s a.mp3 b.mp3          # create a softlink to a.mp3, named b.mp3
$ ln a.mp3 c.mp3             # create a hardlink to a.mp3, named c.mp3
```

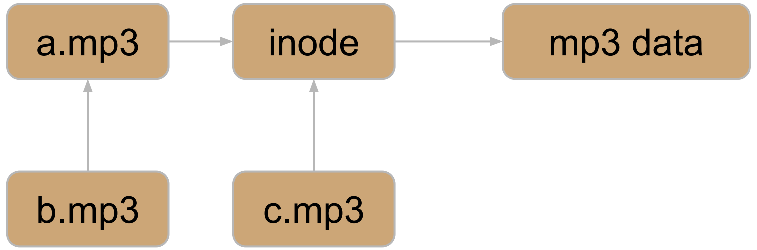
Linux provides a filename(***a.mp3***) to access the **data**(binary/text data stored in hard disk/storage) of *a.mp3*, via an index block(**inode**).

<u>`The links created map as:`</u>

```
b.mp3 -> points to a.mp3
c.mp3 -> points to the inode of a.mp3
```



Link Count



On the basis of this diagram, we conclude:

- If we delete ***a.mp3***:
  - *b.mp3* would become useless
  - *c.mp3* would still be valid, since it was pointing to underlying inode
- When should the operating system delete the ‘**mp3 data**’?
  - When *b.mp3* is deleted? => **doesn’t matter**, since a.mp3 would still be valid
  - When *a.mp3* is deleted? => **no**, since c.mp3 is valid
  - When both *a.mp3* and *c.mp3* are deleted? => **yes**, because all links that could have provided access to the data, are deleted.
- So that means the operating system **needs to maintain a count of how many hard-links** are pointing to a data.
- This count is the **link count** shown by `ls -l`
- As soon as link count goes to 0, the data is marked as deleted by the linux OS.

Link Count - facts

- Hard-link count is generally referred to as **link count** only.
- Soft-links do not participate in the link count, since their existence won't affect file lifetime.
- . (dot) used in relative paths, is a hard-link to current working directory.
- .. (double dot) used in relative path, is a hard-link to parent directory.
- Empty directory has a link count of 2. (think why?)
- The more the number of subdirectories, higher is the link-count of a directory. (think why?)

Linux Security

Authentication and Authorization

- **Authentication** is the process of validating one’s identity.
  - If a person goes to bank to withdraw some money, the bank verifies his/her identity by matching their signatures, or by matching with a photo kept in their record.
  - In linux, a user is authenticated by the combination of their username and passwords.

- **Authorization** is the list of permissible actions a user is allowed to do.
  - Authorization can only take place once a user has been authenticated.
  - Once a person has been authenticated by the bank teller, the teller then checks whether the person is authorized to withdraw money from the mentioned account. Eg - In case of a minor, the minor is not authorized to withdraw money unless accompanied by an authenticated guardian.
  - In linux, authorization is implemented by using the concept of permissions.

- It is important to understand users and permissions in linux, for better debugging.

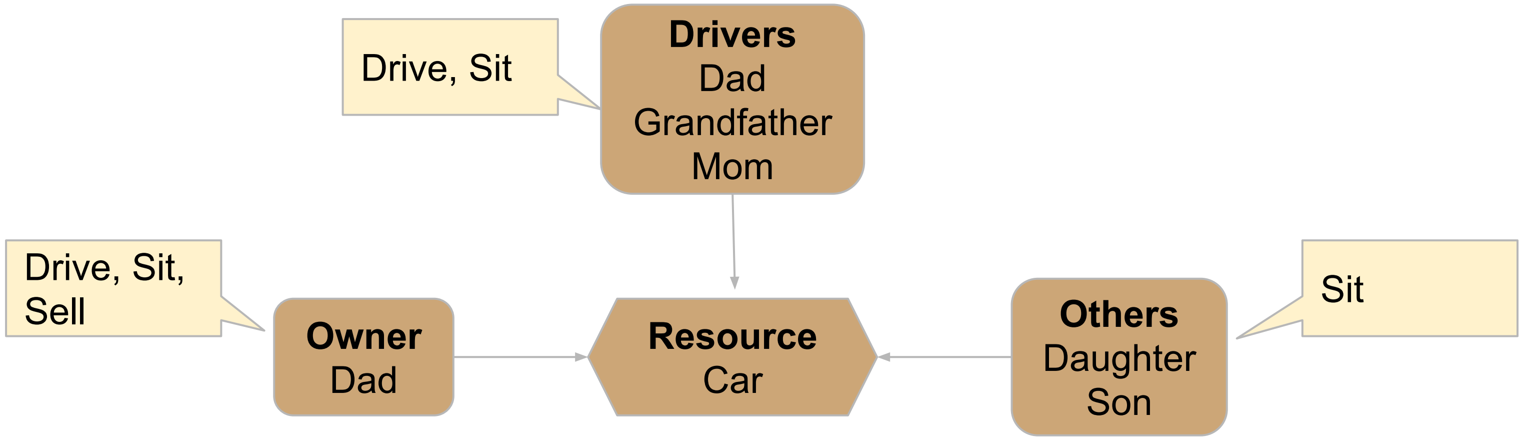
User(Owner) / Group / Others / All

Consider a case -

- There are 5 family members in a house: Grandfather(age:70), Dad(age:40), Mom(age:38), Daughter(age:9), Son (age:4).
- Now they buy a **Car**(resource), the bill is generated against Dad’s name. So legally now, Dad becomes the **owner** of the car.
- Dad has complete authority over the car. He can drive it and can also sell it(whenever he wishes to). Note that only Dad can sell the car, since he is the legal owner.
- Now Grandfather and Mom also have a driving licence and should also be allowed to drive car.
- Members ‘below the age of 18’ or ‘without driving licence’ should not be allowed to drive the car.
- All members should be allowed to sit in the car.
- How would they handle it ?

User(Owner) / Group / Others / All

- As per the problem, only Dad is permitted to drive/operate the car. Since he is the **owner**.
- How to allow Grandfather and Mom to drive, but not the **other** members?
- So the family creates a **group**: ***Drivers***, and make Grandfather and Mom a member of it.
- Anyone who is not the **owner** or is not a member of **Drivers** group, is only allowed to sit in the car.



User(Owner) / Group / Others / All

Observations

- There can be only one **owner** and the owner has complete authority over the resource.
- A resource can only be assigned to one **group**. By having multiple groups, we cannot guarantee that the permissions are restricted. (think why?)
- People can be added or removed from group, whenever required. This makes management of permission really easy.
- Permissions -
    ```
    ○ Owner           : Drive, Sell, Sit
    ○ Group(Drivers)  : Drive, Sit
    ○ Others          : Sit
    ```
- So anyone, who is authenticated, other than owner or group member, is called **others**.
- It is really important to note that **others** need to be **authenticated** users.
  A robber will neither be the owner nor a member of Drivers group as well.
  By definition a robber is **others**. Should you let him sit?

User(Owner) / Group / Others / All

```
$ ls -l sample_directory
lrwxr-xr-x  1 abhinav  staff   9 Mar 10 01:29 link_file -> test_file
drwxr-xr-x  2 abhinav  staff  68 Mar 10 01:14 test_directory
-rw-r--r--  1 abhinav  staff   0 Mar 10 00:40 test_file
```

- In the world of linux permissions to a resource are granted via user, group, and others.
- If a resource is being managed by a group '**G**' and a user '**X**' needs access to the resource. Then **X** needs to be a member of **G**, to access it.
- In the above shown example -
  - (**u**) user/owner => abhinav
  - (**g**) group   => staff
  - (**o**) others   => any authenticated user who is neither '*abhinav*' nor a member of '*staff*'
  - (**a**) all   => u + g + o

Permissions

Permissions

```
$ ls -l sample_directory
lrwxr-xr-x  1 abhinav  staff   9 Mar 10 01:29 link_file -> test_file
drwxr-xr-x  2 abhinav  staff  68 Mar 10 01:14 test_directory
-rw-r--r--  1 abhinav  staff   0 Mar 10 00:40 test_file
```

- **Permissions is the relationship between a resource and its intended users.**
- Permissions for a resource need to be set for **all** users (owner/group/others).
- Permissions stop on first match.
  If a user is found to be owner then linux doesn't check further in group or other.
- Permissions flow from left to right. (user/group/others)
  If a user tries to access a resource, permissions are checked in the following order-
  - Is user the owner? If yes, allow owner privileges
  - Is user in the group? If yes, allow group privileges
  - Else the permissions of others is applicable.

Permissions

```
$ ls -l sample_directory
lrwxr-xr-x  1 abhinav  staff   9 Mar 10 01:29 link_file -> test_file
drwxr-xr-x  2 abhinav  staff  68 Mar 10 01:14 test_directory
-rw-r--r--  1 abhinav  staff   0 Mar 10 00:40 test_file
```

- **Permissions is the relationship between a resource and its intended users.**
- Permissions for a resource need to be set for **all** users (owner/group/others).
- Permissions stop on first match.
  If a user is found to be owner then linux doesn't check further in group or other.
- Permissions flow from left to right. (user/group/others)
  If a user tries to access a resource, permissions are checked in the following order-
  - Is user the owner? If yes, allow owner privileges
  - Is user in the group? If yes, allow group privileges
  - Else the permissions of others is applicable.

Permissions

- There are 4 kind of permissions in linux:
  - read => 2^2 => 4 => r
  - write => 2^1 => 2 => w
  - execute => 2^0 => 1 => x
  - none => 0 => 0 => -

- Permissions can be represented by digits (0-7) as well:
  - `0 =>           0 => ---`
  - `1 =>           1 => --x`
  - `2 =>       2     => -w-`
  - `3 =>       2 + 1 => -wx`
  - `4 => 4           => r--`
  - `5 => 4 +       1 => r-x`
  - `6 => 4 + 2       => rw-`
  - `7 => 4 + 2 + 1 => rwx`

Permissions

```
$ ls -l test_file
-rw-r--r--  1 abhinav  staff   0 Mar 10 00:40 test_file

rw-   r--   r--
```

- The permissions need to be further subdivided in 3 groups
    - The first (leftmost) group represents user/owner's permissions
    - The second(middle) group represent group's permissions
    - The third(rightmost) group represent others' permissions

- So in the above example of *test_file*, it means
    - *abhinav* can *read* and *write* to the test file, but not *execute*
    - *staff* can only *read*
    - *others* can only *read*

Permissions

```
$ ls -l sample_directory
lrwxr-xr-x  1 abhinav  staff   9 Mar 10 01:29 link_file -> test_file
drwxr-xr-x  2 abhinav  staff  68 Mar 10 01:14 test_directory
-rw-r--r--  1 abhinav  staff   0 Mar 10 00:40 test_file
```

- By default, the normal files are not given execute permission. (`-rw-r--r--`)
  This is by design and enhances safety in linux. (think why?)
- By default, the directories require both read and execute for all users. (`drwxr-xr-x`)
  This is also by design, since `cd` command requires execute permission to enter inside a directory.
- Permissions are primarily controlled via group, since -
  - A user can be in multiple groups
  - A group is designed to hold multiple users

Permissions - chmod

- chmod command is used to modify permissions of a file.
- chmod has two mechanisms of working
  - `Numerical mode => chmod 755  test_file`
  - `Textual   mode => chmod o+rx test_file`

- Permissions are assigned from **right to left**, i.e. -
  - The last digit goes to others, then group, then the owner is assigned the leftover
  - `chmod  75   test_file   # Before => rwx rw- r-- ; After => --- rwx r-x`
  - As can be seen, missing a digit results in owner getting no privileges, since allocation is from right to left.

- Permissions can be modified using chmod command. Eg -
  - `chmod 755   test_file   # Before => rw- r-- r-- ; After => rwx r-x r-x`
  - `chmod 000   test_file   # Before => rwx r-x r-x ; After => --- --- ---`
  - `chmod a+rx  test_file   # Before => --- --- --- ; After => r-x r-x r-x`
  - `chmod g-x   test_file   # Before => rwx r-x r-x ; After => rwx r-- r-x`
  - `chmod u+rwx test_file   # Before => r-- r-- r-- ; After => rwx r-- r--`

Permissions (other useful commands)

- chown - Change owner or group of a file

```
chown                    torvalds test_file # make torvalds the owner of file
chown --from=abhinav torvalds test_file # make torvalds the owner of file only if
                                        # previous owner was abhinav (recommended use)

chown                    :friends test_file # make friends the group of file
chown --from=:staff  :friends test_file # make friends the group of file only if
                                        # previous group was staff (recommended use)

chown           torvalds:friends test_file # make torvalds the owner and friends the group
                                        # respectively of file
```

- chgrp - Change group of a file

```
chgrp                     friends test_file # make friends the group of file
chgrp -Rh                 friends test_file # make friends the group of file
```

Some Useful Tips

- How to know what all permissions a linux user has?
  - Check for the **groups** a user is a part of. The permissions that are granted to a group implicitly are also applicable to the concerned user. Eg -

  -
    ```
    groups abhinav                # groups is the command to know about groups of a user
    staff sharepoint everyone localaccounts admin _developer
    ```
  - This concludes that user ***abhinav*** has all the privileges that are granted to the groups (*staff, sharepoint, everyone, localaccounts, admin, _developer*).

- What is the default group assigned to a file ?
  - It is the **primary group** of the user(who is creating the file)
  - All groups other than primary group, are called **secondary groups** w.r.t. a particular user
  - Primary group can be identified by

    -
      ```
      groups abhinav          # first group shown by groups command output, i.e. staff
      staff sharepoint everyone localaccounts admin _developer
      ```

    -
      ```
      id abhinav              # gid shown by id command output, i.e. gid=20(staff)
      uid=501(abhinav) gid=20(staff) groups=20(staff),701(sharepoint),12(everyone),
      61(localaccounts),80(admin),204(_developer)
      ```

Some Useful Tips

- How to find the latest file in a particular directory ?
  - Sort the files in a directory according to time.
  - `ls -ltr        # list the content of a directory sorted by time in reverse order`
    `               # so the latest files should be at the last and oldest at the top`

- What are the default permissions with which a file is created ?
  - The answer lies in the value of **umask**, which is generally set to **022**
  - umask is the value which is subtracted from the default global permissions
  - Default global permissions for regular file => **666**
  - Default global permissions for directory => **777**, since directory needs execute permission (remember?)
  - `umask          # this command displays the current value of umask`
  - If **`umask`** is set to **`022,`** then the default permissions would become :
    - `Regular file => 666 - 022(umask) => 644 => rw- r-- r--`
    - `Directory    => 777 - 022(umask) => 755 => rwx r-x r-x`
  - `umask  011     # command to set custom value of umask variable`

Thank You