Linux Cheat-Sheet

Linux-cheat-sheet

> Click :star:if you like the project. Pull Request are highly appreciated. Follow me @SudheerJonna for technical updates.

Table of Contents

User Information

1. **who** It is used to get information about currently logged in user on to system. If you don't provide any option or arguments, the command displays the following information for each logged-in user.

2. Login name of the user
3. User terminal

4. Date & Time of login
5. Remote host name of the user

```
bash    $ who    sudheer :0 2019-08-04 01:21 (:0)
```

6. **whoami:** It display the system's username

```
bash    $ whoami    sudheer
```

7. **id:** It display the user identification(the real and effective user and group IDs) information

```
bash    $ id    uid=1000(sj) gid=1000(sj)
groups=1000(sj),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),120(lpadmin),131(lxd
),132(sambashare)
```

8. **groups:** This command is used to display all the groups for which the user belongs to.

```
bash    $ group    sj: sj, adm, cdrom, sudo, dip, plugdev, lpadmin, lxd,
sambashare
```

9. **finger:** Used to check the information of any currently logged in users. i.e, It displays users login time, tty (name), idle time, home directory, shell name etc.

```
bash    $ finger    Login     Name       Tty      Idle  Login Time   Office
Office Phone    sj        sj        *:0             Aug 28 01:27 (:0)
```

This may not be available by default in many linux machines. In this case, you need to install it manually.

```
bash    $ sudo apt install finger
```

10. **users:** Displays usernames of all users currently logged on the system.

```
bash    $ users    sj
```

11. **grep:** It is a powerful pattern searching tool to find information about a specific user from the system accounts file: /etc/passwd.

```
bash     $ grep -i sj /etc/passwd     sj:x:1000:1000:sj,,,:/home/sj:/bin/bash
```

12. **W Command:** It(W) is a command-line utility that displays information about currently logged in users and what each user is doing.

```bash
w [OPTIONS] [USER]
```

Example:

```
w
```

```
18:45:04 up  2:09,  1 user,  load average: 0.09, 0.07, 0.02
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
sj       :0       :0               01:27   ?xdm?   1:14   0.01s /usr/lib/gdm3/g
```

13. **last or lastb:** Displays a list of last logged in users on the system. You can pass user names to display their login and hostname details.

    ```bash
last [options] [username...] [tty...]
```

    Example:

    ```
    last
sj       :0           :0               Fri Aug 28 01:27    gone - no logout
reboot   system boot  5.4.0-29-generic Fri Aug 28 01:27   still running
sj       :0           :0               Wed Jul 29 11:46 - crash (29+13:40)
reboot   system boot  5.4.0-29-generic Wed Jul 29 11:45   still running
sj       :0           :0               Thu May 14 21:04 - crash (75+14:41)
reboot   system boot  5.4.0-29-generic Thu May 14 21:03   still running

    wtmp begins Thu May 14 21:03:56 2020
```

14. **lastlog:** The `lastlog` command is used to find the details of a recent login of all users or of a given user.

    ```cmd
$ lastlog
```

    ```
    Username         Port     From             Latest
root                                      Never logged in
daemon                                    Never logged in
bin                                       Never logged in
sys                                       Never logged in
sync                                      Never logged in
games                                     Never logged in
man                                       Never logged in
lp                                        Never logged in
mail                                      Never logged in
news                                      Never logged in
```

[↑ Back to Top]

File and directory commands

1. **pwd** The pwd(Present Working Directory) command is used to print the name of the present/current working directory starting from the root.
   `bash $ pwd /home/sj/Desktop/Linux`
2. **ls**: The `ls` command is used to list files or directories. It also accepts some flags or options that changes how files or directories are listed in your terminal.

```bash
Syntax:
ls [flags] [directory]

Example:
$ ls
bin dev lib libx32 mnt

//Listing files & directories with time in a rever order
$ ls -ltr
drwxr-xr-x 2 sj sj 4096 May 14  2020 Videos
drwxr-xr-x 2 sj sj 4096 May 14  2020 Templates
drwxr-xr-x 2 sj sj 4096 May 14  2020 Public

//Home directory
$ ls ~
Desktop  Downloads  Pictures  Sudheer  test  test.txt
Documents  Music  Public  Templates  test1  Videos
```

Below are the list of possible options for `ls` command,

`cmd -a Show all (including hidden) -R Recursive list -r Reverse order -t Sort by last modified -S Sort by file size -l Long listing format -1 One file per line -m Comma-separated output -Q Quoted output`

3. **mkdir** The mkdir(make directory) command allows users to create directories or folders.

`bash $ mkdir ubuntu $ ls ubuntu`

The option '-p' is used to create multiple directories or parent directories at once.

`bash $ mkdir -p dir1/dir2/dir3 $ cd dir1/dir2/dir3 ~/Desktop/Linux/dir1/dir2/dir3$`

4. **rmdir**: The rmdir(remove directories) is used to remove *empty* directories. Can be used to delete multiple empty directories as well. Safer to use compared to `rm -r FolderName`. This command can also be forced to delete non-empty directories.

5. Remove empty directory:

```bash
rmdir FolderName
```

6. Remove multiple directories:

```bash
rmdir FolderName1 FolderName2 FolderName3
```

7. Remove non-empty directories:

```bash
rmdir FolderName1 --ignore-fail-on-non-empty
```

8. Remove entire directory tree. This command is similar to `rmdir a/b/c a/b a`:

```bash
rmdir -p a/b/c
```

9. **rm**: The rm(remove) command is used to remove objects such as files, directories, symbolic links etc from the file system.
   10. Remove file: The rm command is used to remove or delete a file
   ```bash
   rm file_name
   ```
   11. Remove file forcefully: The rm command with -f option is used for removal of file without prompting for confirmation.
   ```bash
   rm -f filename
   ```
   12. Remove directory: The rm command with -r option is used to remove the directory and its contents recursively.
   ```bash
   rm -r myDir
   ```
   13. Remove directory forcefully: The rm command with -rf option is used to forcefully remove directory recursively.
   ```bash
   rm -rf myDir
   ```
10. **touch**: The touch command is is used to create, change and modify timestamps of a file without any content.
   15. **Create a new file:** You can create a single file at a time using touch command. The file created is an empty file.
   ```bash
   touch file_name
   ```
   16. **Create multiple files:** You can create the multiple numbers of files at the same time.
   ```bash
   touch file1_name file2_name file3_name
   ```
   17. **Change access time:** The touch command with `a` option is used to change the access time of a file.
   ```bash
   touch -a file_name
   ```
   18. **Change modification time:** The touch command with `m` option is used to change the

modified time.

```
bash        touch -m file_name
```

19. **Use timestamp of other file:** The touch command with `r` option is used to get timestamp of another file.

```
bash        touch -r file2 file1
```

In the above example, we get the timestamp of file1 for file2.

20. **Create file with Specific time:** The touch command with 't' option is used to create a file with specified time.

```
bash        touch -t 1911010000 file_name
```

21. **cat**: The cat command is used to create single or multiple files, view contain of file, concatenate files and redirect output in terminal or files.

```
bash      $ cat [OPTION] [FILE]...
```

22. **Create a file:** Used to create a file with specific name, content and press exit using `CTRL + D`

```
bash        cat > file_name1.txt        Hello, How are you?
```

23. **View file contents:** You can view contents of a single or more files by mentioning the filenames.

```
bash        cat file_name1 file_name2
```

24. **More & Less options:** If a file having a large number of content that won't fit in the output terminal then `more` & `less` options can be used to indiate additional content.

```
bash        cat file_name1.txt | more        cat file_name1.txt | less
```

[⬆ Back to Top]

File permissions

Since Linux is a multi-user operating system, it is necessary to provide security to prevent people from accessing each other's confidential files.
So Linux divides authorization into 2 levels,

1. **Ownership:**
   Each file or directory has assigned with 3 types of owners
   i. **User:** Owner of the file who created it.
   ii. **Group:** Group of users with the same access permissions to the file or directory.
   iii. **Other:** Applies to all other users on the system
2. **Permissions:**
   Each file or directory has following permissions for the above 3 types of owners.

i. **Read:** Give you the authority to open and read a file and lists its content for a directory.

ii. **Write:** Give you the authority to modify the contents of a file and add, remove and rename files stored in the directory.

iii. **Execute:** Give you the authority to run the program in Unix/Linux.

The permissions are indicated with below characters,

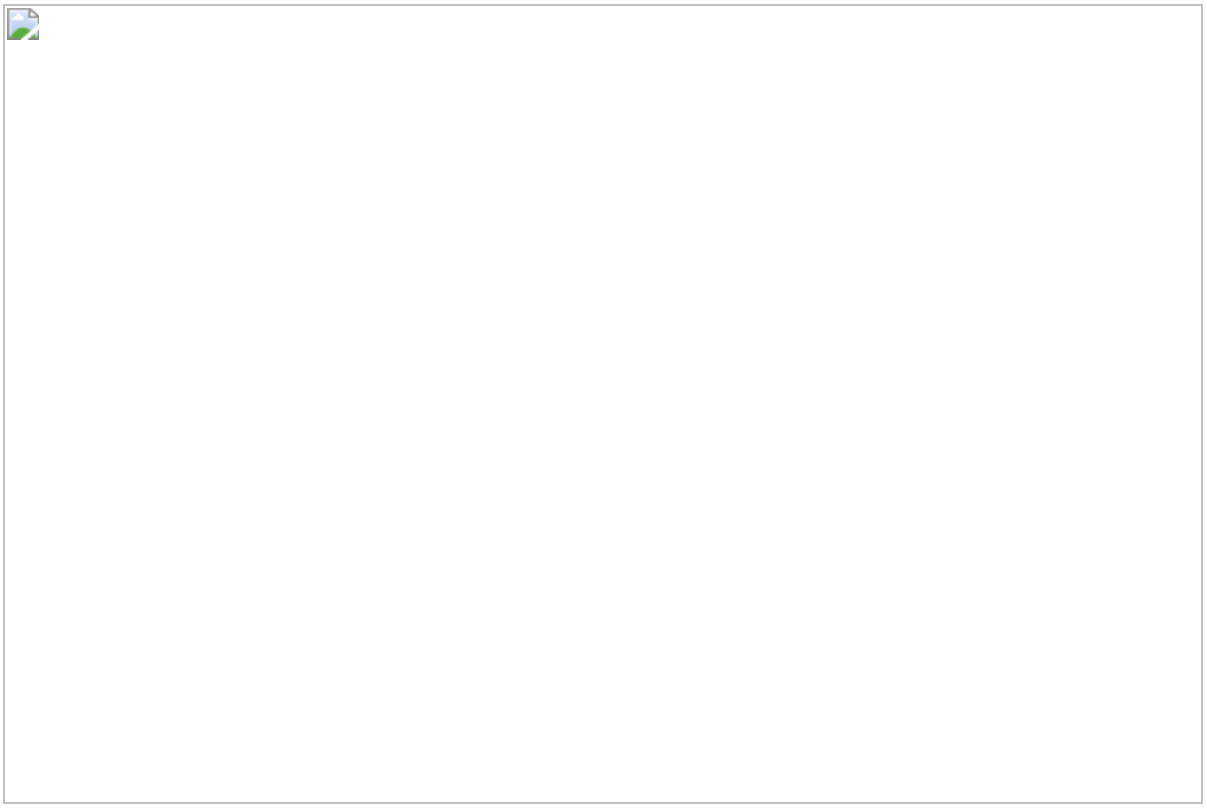
r = read permission

w = write permission

x = execute permission

- = no permission

The above authorization levels represented in a diagram



` | Pass custom headers to the request. | | `--data ` | HTTP POST data. | | `--data-urlencode ` | URL encode the POST data. | | `--form ` | Specify multipart/form-data POST data. | | `-d, --data-raw ` | Send data as-is without any encoding. | | `-i, --include` | Include HTTP headers in the output. | | `--url ` | Specify the URL to send the request to. | | `-o, --output ` | Write output to a file instead of stdout. | | `-O, --remote-name` | Save the file with the same name as in the URL. | | `-L, --location` | Follow redirects. | | `--user ` | Specify the user and password for server authentication.| | `-u, --user-agent ` | Set the user-agent string. | | `--cookie ` | Send cookies from string/file. | | `--

cookie-jar `  | Save cookies to a file after receiving them.  | | `--compressed`  | Request compressed response.  | | `-v, --verbose`  | Make the operation more talkative for debugging.  | | `-h, --help` | Display help information. | 1. Basic GET Request: Sends a simple GET request to the specified URL. ```cmd curl http://example.com ``` 2. Specify Request Method: Specifies a request method other than the default GET. ```cmd curl -X POST http://example.com ``` 3. Include Headers: Includes custom headers in the request. ```cmd curl -H "Content-Type: application/json" http://example.com ``` 4. Send POST Data: Sends data as POST request payload. ```cmd curl -X POST --data "key1=value1&key2=value2" http://example.com ``` 5. Save Output to File: Writes the output to a specified file. ```cmd curl -o output.html http://example.com ``` 6. Download File: Downloads a file and saves it with the same name as in the URL. ```cmd curl -O http://example.com/file.zip ``` **[⬆ Back to Top](#table-of-contents)**