Linux Fundamentals

How to check the current user?

```
whoami
```

How to check which users are currently logged in?

```
who
```

How to list all users on the system?

```
cat /etc/passwd
# it provides all users in the system with detailed information
```

```
username:x:uid:gid:comment:<home/path>:shell

username: name of the user
x: encrypted password
uid: user id
gid: group id
comment(optional): If given, will show information provided during user creat
home/path: User home directory
shell: Default shell used by the user
```

```
cat /etc/passwd | cut -d':' -f1
# or
cut -d':' -f1 /etc/passwd
# it provides all users name as a list

<<comment
-d':' -> specifies delimiter
-f1 -> extract the first field only
comment
```

How to change the user?

```
su - <username>
```

What is a login shell?

A login shell is the first shell session that a user starts upon logging into a system. It is initialized as part of the user's login process and is responsible for setting up the user environment by loading specific configuration files.

Difference between login and non-login shell?

The distinction between a **login shell** and a **non-login shell** lies in how they handle **environment initialization**.

A **login shell** initializes the user's full environment by reading specific configuration files. A **non-login shell** skips these files.

In non-login shells, user-specific environments are not applied.

How to check if the shell is a login shell or not?

```
shopt -q login_shell && echo "This is a login shell"
|| echo "This is not a login shell"
```

`shopt` stands for shell option.

How to switch to the login shell?

```
bash -l
su - <username>
```

How to create a user?

```
sudo useradd <username>
# creates just a new user

sudo useradd -m -s <shellname> -c <comment> <username>

<<comment
-m option creates a home directory for user inside /home
-s specifies the default shell for the user
comment

sudo adduser <username>
# alternative approach to create user
```

How to modify existing users?

```
sudo usermod -s <shellname> <username>
# change shell

sudo usermod -d /new/home/path -m <username>
# change current home path
```

```
# -m copies existing files to new home path

sudo usermod -l <newusername> <oldusername>
# change user name
```

How to delete a user?

```
sudo userdel -r <username>
# removes user with it's home directory
```

Can multiple users be deleted with userdel command?

It’s possible to delete multiple users with userdel command, it doesn’t delete the home directories along with the users.

How to add a new user to a group?

```
sudo usermod -aG <groupname> <username>
```

How to create a group?

```
sudo groupadd <groupname>
```

How to change the group name?

```
sudo groupmod -n <new_groupname> <old_groupname>
```

How to change the group ID?

```
sudo groupmod -g <new_gid> <groupname>
```

How to delete a group?

```
sudo groupdel groupname
```

How to remove a user from a group?

```
sudo gpasswd -d <username> <groupname>
```

How to change the owner of a file/directory?

```
sudo chown <username> <file_or_directory_name>
# or
sudo chown <username>:<groupname> <file_or_directory_name>
```

How to change group ownership only?

```
sudo chgrp groupname file_or_directory
# or
sudo chown :<groupname> <file_or_directory>
```

How to modify a file/directory?

```
sudo chmod u=rwx,g=rw,o=r <file_or_directory>
# r = read
# w = write
# x = executable

# or

sudo chmod 754 <file_or_directory>
<<comment
first number denotes user
second number denotes group
third number denotes others

4 = read
2 = write
1 = executable
comment
```

How to view permissions?

```
ls -l <file_or_directory>
```

How to modify file attributes?

File attributes can be modified with `chattr` which stands for change attribute.

Usage:

`chattr +i filename`: Make the file immutable.

`chattr -i filename`: Remove the immutable flag.

`chattr +a filename`: Make the file append-only. Only new data can be written at the end of the file

`chattr +d filename`: Prevent the file from being backed up using `dump`.

`chattr +s filename`: Enable secure deletion for the file. Secure deletion means data can't be recovered

`chattr +t dirname`: Can only be deleted by the owner.

List directory

```
ls [options] [directory_or_path]
# or
ls -l
# or
ls -la
```

How to check the current working directory?

```
pwd
```

How to change directory?

```
cd [path_or_directory]
<<comment
cd : change directory to current user's home dir
cd.. : moves up a directory
cd - : goes back to previous directoy
comment
```

mkdir command

The **mkdir** command lets you create one or multiple directories. The syntax looks like this:

```
mkdir [options] directory_name1 directory_name2
```

To create a folder in another location, specify the full path. Otherwise, this command will make the new item in your current working directory.

For example, enter the following to create **new_folder** in **/path/to/target_folder**:

```
mkdir path/to/target_folder/new_folder
```

By default, **mkdir** allows the current user to read, write, and execute files in the new folder. You can set custom privileges during the creation by adding the **-m** option. To learn more about permission management, read the **chmod** section below.

rm command

The **rm** command deletes files from a directory. You must have the write permission for the folder or use **sudo**. Here's the syntax:

```
rm [options] file1 file2
```

You can add the **-r** option to remove a folder and its contents, including subdirectories. Use the **-i** flag to display a confirmation message before the removal or **-f** to deactivate it completely.

Warning! Avoid using **-r** and **-f** unless necessary. Instead, add **-i** option to prevent accidental deletion.

Remove directory and subdirectory

```
rm -rf <directory_name>
```

cp command

Use the **cp** command to copy files from your current directory to another folder. The syntax looks like this:

```
cp file1 file2 [target_path]
```

You can also use **cp** to duplicate the content of one file to another using this syntax. If the target is in another location, specify the full path like so:

```
cp source_file /path/to/target_file
```

Additionally, **cp** lets you duplicate a directory and its content to another folder using the **-R** option:

```
cp -R /path/to/folder /target/path/to/folder_copy
```

mv command

The main usage of the **mv** command is to move a file or folder to another location. Here's the syntax:

```
mv file_or_directory [target_directory]
```

For example, we will move **file1.txt** from another location to the **/new/file/directory** path using this command:

```
mv /original/path/file1.txt the/target/path
```

You can also use the **mv** command to **rename files in your Linux system**. Here's an example:

```
mv old_name.txt new_name.txt
```

If you specify the full path, you can simultaneously rename files and move them to a new location like this example:

```
mv old/location/of/old_name.txt new/path/for/new_name.txt
```

How to create a file?

```
touch [options] [path_and_file_name]
```

How to check file type?

```
file [file_name]
```

If this command is used on a symbolic link, it will output the actual file connected to the shortcut. The **-k** option to print more detailed information about the item.

How to zip and unzip?

The **zip** command compresses one or multiple files into a **ZIP** archive, reducing their size.

```
zip [options] zip_file_name file1 file2
```

To **extract a compressed file** into your current working directory, use the **unzip** command like so:

```
unzip [options] zip_file_name
```

tar command

The **tar** command bundles multiple files or directories into an archive without compression. The syntax looks as follows:

```
tar [options] tar_file_name file1 file2
```

To create a new **TAR** file, you must add the **-c** option. Then, use the **-f** flag to specify the archive's name.

If you want to enable compression, add a specific option based on your preferred method. For example, the following will bundle **file1.txt** and **file2.txt** with the **gzip** compression:

```
tar -cfz archive.tar.gz fle1.txt file2.txt
```

Remember that the archive's file format will differ depending on the compression method. Regardless of the extension, you can unpack a **TAR** file using this syntax:

```
tar [options] tar_file_name
```

To extract in the current directory

```
tar –xvzf [archive name]
```

To extract in a specific directory

```
tar -xvzf [archive name] -C <destination/path>
```

cat command

The **concatenate** or **cat** command has various usages. The most basic one is printing the content of a file. Here's the syntax:

```
cat file_name
```

To print the content in reverse order, use **tac** instead. If you add the standard output operator symbol (**>**), the **cat** command will create a new file. For example, the following will make **file.txt**:

```
cat > file.txt
```

You can also use cat with the operator to combine the content of multiple files into a new item. In this command, **file1.txt** and **file2.txt** will merge into **target.txt**:

```
cat file1.txt file2.txt > target.txt
```

grep command

Global regular expression print or **grep** lets you search specific lines from a file using keywords. It is useful for filtering large data like logs. The syntax looks as follows:

```
grep [options] keyword [file]
```

You can also filter data from another utility by piping it to the **grep** command. For example, the following searches **file.txt** from the **ls** command's output:

```
ls | grep "file.txt"
```

sed command

Use the **sed** command to search and replace patterns in files quickly. The basic syntax looks like this:

```
sed [options] 'subcommand/new_pattern/target_pattern' input_file
```

You can replace a string in multiple files simultaneously by listing them. Here's an example of a sed command that changes **red** in **colors.txt** and **hue.txt** with **blue**:

```
sed 's/red/blue' colors.txt hue.txt
```

head command

Use the **head** command to print the first few entries of a file. The basic syntax is as follows:

```
head [options] file_name
```

You can also print the first few lines of another command's output by piping it like so:

```
command | head [options]
```

By default, the **head** will show the **first ten lines**. However, you can change this setting using the **-n** option followed by your desired number.

Meanwhile, use **-c** to print the first few entries based on the byte size instead of the line.

tail command

The **tail** command is the opposite of the **head**, allowing you to print the last few lines from files or another utility's output. Here are the syntaxes:

```
tail [options] file_name
```

```
command | tail [options]
```

The **tail** utility also has the same option as **head**. For example, we will extract the **last five lines** from the **ping** command's output:

```
ping -c 10 8.8.8.8 | tail -n 5
```

awk command

The **awk** command searches and manipulates **regular expression** patterns in a file. Here's the basic syntax:

```
awk '/regex pattern/{action}' input_file.txt
```

Although similar to **sed**, **awk** offers more operations beyond substitution, including printing, mathematical calculation, and deletion. It also lets you run a complex task with an **if** statement.

You can run multiple actions by listing them according to their execution order, separated by a semicolon (**;**). For example, this **awk** command calculates the average student score and print names that are above that threshold:

```
awk -F':' '{ total += $2; students[$1] = $2 } END { average = total / lengt
h(students); print "Average:", average; print "Above average:"; for (studen
t in students) if (students[student] > average) print student }' score.txt
```

sort command

Use the **sort** command to rearrange a file's content in a specific order. Its syntax looks as follows:

```
sort [options] [file_name]
```

Note that this utility doesn't modify the actual file and only prints the rearranged content as an output.

By default, the **sort** command uses the alphabetical order from **A** to **Z**, but you can add the **-r** option to reverse the order. You can also sort files numerically using the **-n** flag.

cut command

The **cut** command selects specific sections from a file and prints them as a Terminal output. The syntax looks like this:

```
cut options file
```

Unlike other Linux utilities, the **cut** command's options are mandatory for file sectioning. Here are some of the flags:

f – select a specific row field.

b – cuts the line by a specified byte size.

c – sections the line using a specified character.

d – separates lines based on delimiters.

You can combine multiple options for a more specific output. For example, this command extracts the **third** to **fifth** field from a comma-separated list:

```
cut -d',' -f3-5 list.txt
```

diff command

The **diff** command compares two files and prints their differences. Here's the syntax:

```
diff file_name1 file_name2
```

By default, the **diff** command only shows the differences between the two files. To print all the content and highlight the discrepancies, enable the context format using the **-c** option. You can also ignore case sensitivity by adding **-i**.

For example, run the following to show only the differences between **1.txt** and **2.txt**:

```
diff -c 1.txt 2.txt
```

tee command

The **tee** command outputs another command's results to both the Terminal and a file. It's helpful if you want to use the data for further processing or backups. Here's the syntax:

```
command | tee [options] file_name
```

If the specified file doesn't exist, **tee** will create it. Be careful when using this command since it will overwrite the existing content. To preserve and append existing data, add the **-a** option.

For example, we will save the ping command's output as new entries in the **test_network.txt** file:

```
ping 8.8.8.8 | tee -a test_network.txt
```

locate command

The **locate** command searches for a file and prints its location path.

```
locate [options] [keyword]
```

If you use the **-r** option to search files using regular expressions, omit the **[keyword]** argument. The **locate** command is case-sensitive by default, but you can turn off this behavior using the **-i** flag.

Note that the **locate** will look for files from its database. While this behavior speeds up the search process, you must wait for the list to refresh before finding newly created items.

Alternatively, enter the following to reload the data manually:

```
updatedb
```

find command

The **find** command searches for a file within a specific directory. Here's the syntax:

```
find [path] [options] expression
```

If you don't specify the path, the **find** command will search your current working directory. To find files using their name, add the **-name** option followed by the keyword.

You can specify the type of item you are looking for using the **-type** flag. The –**type f** option will search files only, while **-type d** will find directories. For example, we will check **file.txt** in **path/to/folder**:

```
find path/to/folder -type f -name "file"
```

Unlike locate, the **find** command searches through folders in real time. While it slows down the process, you can look for new items immediately without waiting for the system database to refresh.

What is a soft link and a hard link?

A **soft link** (or **symbolic link**) is a file that acts as a pointer to another file or directory. It's similar to a shortcut in Windows.

Characteristics of Soft Links:

- It is a separate file that contains the path to the original file.
- If the original file is deleted, the soft link becomes **broken** (a "dangling link").

- Can link to files and directories, even across different file systems.
- Doesn't increase the link count of the original file.

```
ln -s target_file link_na
```

Use Cases:

- Shortcut to frequently accessed files or directories.
- Linking to a file on a different disk or partition.

A **hard link** is another reference (or name) to the same physical file on the disk. Both the original file and the hard link share the same **inode number**.

Characteristics of Hard Links:

- It is not a separate file; it points directly to the data blocks of the original file.
- If the original file is deleted, the hard link still accesses the data because the file's data remains on the disk until all hard links are deleted.
- Cannot link directories (to prevent cyclic structures).
- Only works on the same file system (not across partitions).
- Increases the link count of the original file.

```
ln target_file link_name
```

Use Cases:

- Creating redundant references to important files.
- Sharing file content with different names while using minimal disk space.