Lesson 01

To manage a Linux remote machine like a EC2 Instances, system administrators commonly use a command-line interface (CLI). While some Linux distributions offer a graphical user interface (GUI), commands are more efficient and flexible with a CLI.

For example, you can replace specific entries in multiple files using a single command, which takes time in a GUI. However, using bash shell can be tricky since Linux has many commands for various tasks.

Basic Unix Commands

This section will explore basic Linux commands for file and directory management.

> **Tip**

> To run Linux commands in a remote server, connect via SSH using PuTTY or Terminal. Alternatively, users can leverage our built-in Browser terminal to run the commands directly from their web browsers.

Navigation (`pwd`, `cd`, `ls`)

1. pwd command

The **pwd** command prints your current working directory's path, like **/home/directory/path**. Here's the command syntax:

```
pwd [option]
```

It supports two options. The **-L** or –**-logical** option prints environment variable content, including symbolic links Meanwhile, **-P** or **–physical** outputs the current directory's actual path.

```
rajath@rhelpro ~ (0.02s)
pwd
/home/rajath
```

2. cd command

Use the **cd** command to navigate the Linux files and directories. To use it, run this syntax with sudo privileges:

```
cd /directory/folder/path
```

```
rajath@rhelpro ~ (0.024s)
cd test/app/
```

Depending on your current location, it requires either the full path or the directory name. For example, omit **/username** from **/username/directory/folder** if you are already within it.

Omitting the arguments will take you to the home folder. Here are some navigation shortcuts:

- **cd ~[username]** – goes to another user's home directory.
- **cd ..** – moves one directory up.
- **cd-** – switches to the previous directory.

3. ls command

The **ls** command lists files and directories in your system. Here's the syntax:

```
ls [/directory/folder/path]
```

```
rajath@rhelpro ~ (0.024s)
ls test/
app  hello.sh
```

If you remove the path, the **ls** command will show the current working directory's content. You can modify the command using these options:

- **-R** – lists all the files in the subdirectories.
- **-a** – shows all files, including hidden ones.
- **-lh** – converts sizes to readable formats, such as **MB**, **GB**, and **TB**.

File Operations (`cat`, `cp`, `mv`, `rm`, `touch`)

4. cat command

Concatenate or **cat** is one of the most used Linux commands. It lists, combines, and writes file content to the standard output. Here's the syntax:

```
cat filename.txt
```

There are various ways to use the **cat** command:

- **cat > filen.txt** – creates a new file.
- **cat file1.txt file2.txt > file3.txt** – merges **file1.txt** with **file2.txt** and stores the output in **filename3.txt**.
- **tac file.txt** – displays content in reverse order.

5. cp command

Use the **cp** command to copy files or directories, including their content, from your current location to another. It has various use cases, such as:

- Copying one file from the current directory to another folder. Specify the file name and target path:

```
cp filename.txt /home/username/Documents
```

- Duplicating multiple files to a directory. Enter the file names and the destination path:

```
cp filename1.txt filename2.txt filename3.txt /home/username/Documents
```

- Copying a file's content to another within the same directory. Enter the source and the destination file:

```
cp filename1.txt filename2.txt
```

- Duplicating an entire directory. Pass the **-R** flag followed by the source and destination directory:

```
cp -R /home/username/Documents /home/username/Documents_backup
```

6. mv command

The **mv** or **move** command is used for two essential tasks in handling files in Terminal – moving files between locations and renaming them.

The basic **mv** command syntax is as follows:

```
mv option SOURCE...DIRECTORY
```

SOURCE refers to the file's origin directory, while **DESTINATION** is its target path.

The **option** is an additional parameter for modifying the command's output. Here are several popular **mv** options:

- **-f** – shows no message before overwriting a file.
- **-i** – displays warning messages before overwriting a file.
- **-u** – only moves a file if it is new or doesn't exist in the destination.
- **-v** – explains what the command does.

To rename a file, use the following **mv** command syntax. Note that this only works if you are in the same directory as the file:

```
mv oldnamefile1 newnamefile1
```

If there is a file called **file1.txt** which you want to rename to **file2.txt**, enter the following:

```
mv file1.txt file2.txt
```

If you are not in the file's location, you must change the current working directory using the **cd** command. For example:

```
cd /home/user/docs/files
mv file1.txt file2.txt
```

The **rename** command gives you more flexibility in modifying the files. Many Linux configurations include this command by default. If your system doesn't have one, install it via Terminal.

Depending on your server operating system, the installation command for **rename** differs. For Debian, Ubuntu, Linux Mint, and their derivatives, use the following command:

```
sudo apt install rename
```

Meanwhile, use this command if you are operating CentOS 7 or RHEL:

```
sudo yum install rename
```

For Arch Linux, run this command:

```
yay perl-rename ## or yaourt -S perl-rename
```

7. rm command

Use the rm command to permanently delete files within a directory. Here's the general syntax:

```
rm [filename1] [filename2] [filename3]
```

Adjust the number of files in the command according to your needs. If you encounter an error, ensure you have the **write** permission in the directory.

To modify the command, add the following options:

- **-i** – prompts a confirmation before deletion.
- **-f** – allows file removal without a confirmation.
- **-r** – deletes files and directories recursively.

> ⚠ **Warning**
>
> Use the **rm** command with caution since deletion is irreversible. Avoid using the **-r** and **-f** options since they may wipe all your files. Always add the **-i** option to avoid accidental deletion.

8. touch command

The **touch** command lets you create an empty file in a specific directory path. Here's the syntax:

```
touch [option] /home/directory/path/file.txt
```

If you omit the path, the command will create the item in the current folder. You can also use **touch** to generate and modify a timestamp in the Linux command line.

Check out the table below for all the touch command options, also known as flags:

Flag	Use Case
-a	Change the access time
-m	Change the modification time
-c	Prevent creating a new file
-h	Change the symbolic link timestamp
-h	Change the timestamp for symbolic links
-t	Modify the timestamp. In this case, follows the date-time format
-d=	Change the timestamp based on the date string
-r=	Change the timestamp based on the reference file
-v or –version	Display the touch command version
–help	Display the help menu

File Timestamps

In Linux, every file and folder has a timestamp that shows when a file's content or attributes were modified. There are three types of timestamps:

- **Access time (atime)** – last time a file was read.
- **Modification time (mtime)** – last time a file's content was modified. Like access time, it is also part of the file status metadata.
- **Changed time (ctime)** – last time a file's metadata was changed. For example, permissions.

As a result, the Linux touch command is mainly used to manipulate file or folder access and modification time.

Keep in mind that there is no way to set or change **ctime** manually. Since **atime** and **mtime** are part of a file's status metadata, changing **atime** or **mtime** of the file results in **ctime**, which is automatically set to the current time.

Linux touch Command Examples

Here are a few useful examples of the touch command on a Linux system.

Using touch to Create a File

If you use the touch command without any options, it will simply create a new empty file. If the file already exists, the touch command will update the access and modification times to the current time without changing the file contents.

```
touch hello.sh
```

```
rajath@rhelpro ~/test (0.027s)
ls -l

total 4
drwxr-xr-x 2 rajath rajath   6 Aug  4 16:36 app
-rwxr-xr-x 1 rajath rajath 185 Aug  4 23:56 hello.sh
```

Using touch to Create Multiple Files

It is also possible to create multiple files using a single touch command. To do that, specify the names of the files with spaces between them. It would look like this in the command line:

```
touch file_name1.txt file_name2.txt file_name3.txt
```

You can auto-generate file names using curl braces while creating multiple files, like in the following example:

```
touch file_name{1..3}.txt
```

The above touch command will create three files named **file_name1.txt**,**file_name2.txt,** and **file_name3.txt**.

Using touch to Change Access Time

To change the access time of a file to the current time, use the **a** option followed by the file name with the touch command like in the following example:

```
touch -a file_name.txt
```

Then, check the access time with the following command:

```
ls -lu file_name.txt
```

```
rajath@rhelpro ~/test (0.026s)
ls -lu file_name.txt

-rw-r--r-- 1 rajath rajath 0 Aug  5 03:30 file_name.txt
```

Using touch to Change Modification Time

The **m** option, along with the touch command, changes the modification time of a file to the current time:

```
touch -m file_name.txt
```

Using touch to Change Access and Modification Time

To change both access time and modification time with a single command, use the options **a** and **m** together:

```
touch -am file_name.txt
```

Now, check the date with both of these commands:

```
ls -l file_name.txt
ls -lu file_name.txt
```

Using touch to Change Access Time Without Creating a New File

In some situations, you want to change an existing file's access and modification time to the current time without actually creating a new one. To do that, use the **c** option followed by the file name with the touch command.

```
touch -c new_file.txt
```

Using touch to Set Specific Access and Modification Time

It is also possible to set a file's access and modification time to a particular date by using the **t** option followed by a date-time. It would look like this:

```
touch -t 202203081047.30 file_name.txt
```

Make sure to check whether the date changed with the following command:

```
ls -lu file_name.txt
```

Remember that the date-time format must follow the **CCYYMMDDhhmm.ss** style:

- **CC** – the first two digits of the year
- **YY** – the second two digits of the year
- **MM** – the month of the year (01-12)
- **DD** – the day of the month (01-31)
- **hh** – the hour of the day (00-23)
- **mm** – the minute of the hour (00-59)
- **ss** – the second of the minute (00-59)

Directory Operations (`mkdir`, `rmdir`)

9. rmdir command

To permanently remove a directory in Linux, use either the **rmdir** or **rm**command. The **rmdir** or **rm -d** command is for removing empty directories, while the **rm -r** command deletes non-empty directories.

Before removing a directory, you must know its name. To discover files and directories, use the **ls** command. To know the current working directory, use the **pwd** command.

The options you use with these commands also determine how they work. Here's a quick recap of **rm** command options:

Command and Option	Description
rm -d	Remove an empty directory using the **rm** command.
rm -r	Remove a non-empty directory and its content.
rm -f	Ignore any prompt when deleting a write-protected file.
rm -rf	Ignore any prompt when deleting a write-protected non-empty folder.
rm -i	Prompt for confirmation before each removal.
rm -I	Output a prompt only once before deleting more than three files.
rm *	Wildcard that represents multiple characters.
rm ?	Wildcard that represents a single character.
rmdir -p	Remove an empty subdirectory and its parent directory.
rmdir -v	Print the information that the specified directory was deleted.

Using the **rmdir** command prevents such accidents since it only works for empty directories. It will return the following error message if the directory contains files:

```
rmdir: failed to remove 'Directory': Directory not empty
```

The **rmdir** command syntax is as follows:

```
rmdir option DirectoryName
```

You may use the **rmdir** command without an option by removing it from the syntax. Since the command line is case-sensitive, remember to type the folder name accordingly.

For example, here's a command for deleting the **Simple-Directory** folder:

```
rmdir Simple-Directory
```

The **rmdir** command also lets you delete multiple empty directories simultaneously. To do so, add the directories as additional arguments after**rmdir**. The syntax looks as follows:

```
rmdir Directory_1 Directory_2 Directory_3
```

There are various options you can combine with the **rmdir** command. For example, use **-p** to delete a subdirectory and its parent.

For example, the following command will delete the **/Directory/SubDirectory** directory path:

```
rmdir -p /Directory/SubDirectory
```

The above command will delete the **SubDirectory** folder in the **Directory**path. Then, it will also delete the **Directory** folder if it is empty.

The next option is **-v** or **verbose**. This option tells the command line interface to print a confirmation message to verify that **rmdir** has successfully deleted the specified directory. Here's an example of this command:

```
rmdir -v Simple-Directory
```

The output message will appear similar to the following:

```
rmdir: removing directory, 'Simple-Directory'
```

Use the **rm** command to remove non-empty directories. This command is meant for removing files, but we can combine it with options like **-r**, **-rf**, and**-d** to delete a directory. Here's the syntax for the command:

```
rm option FileOrFolderName
```

Use the **-r** or **recursive** option to remove the directory and its content. For example, the following command will delete **Simple-Directory**, including the subdirectories and files within it:

```
rm -r Simple-Directory
```

> **Caution**
>
> Deleting a directory using the **rm -r** command will also wipe its content. Execute this command with caution since you can only recover them using a backup.

If the directory is **write-protected**, the command line interface will request confirmation before deleting. To delete a directory in Linux without confirmation, use the **-rf** option.

```
rm -rf Simple-Directory
```

You can also use the **rm** command to remove empty directories in Linux. However, you need the **-d** option.

```
rm -d Simple-Directory
```

Similar to the **rmdir** command, you can use **rm** to remove multiple directories. Add the directory names as new arguments in the command line:

```
rm -r Directory_1 Directory_2 Directory_3
```

You may want to use the **rm** command to manually remove files instead of deleting the folder. This method is safer as it prevents accidental file removals.

To delete a single file in the current working directory, use the **rm** command with the file name:

```
rm file.txt
```

You can also delete multiple files in the directory by listing their names:

```
rm file1.txt file2.txt file3.txt
```

All these commands only work if you are in the specified files' directory. However, you can put the file path as an argument to delete a file in another folder:

```
rm dir/subdir/file.txt
```

Since this command will permanently delete your files, add the **-i** option to enable removal confirmation. It adds an extra step to prevent accidental deletion. Here's the command:

```
rm -i file1.txt file2.txt file3.txt
```

Terminal will prompt you with an option to proceed or cancel the deletion. Type **Y** and press **Enter** if you want to delete the file. Otherwise, hit **N** and **Enter**.

Alternatively, use **-I** if you want the confirmation for deleting more than three files to show only once. While this option is less safe than **-i**, it is still relatively effective to prevent accidents. Here's what the command looks like:

```
rm -I file1.txt file2.txt file3.txt
```

The system will still prompt the confirmation if you are removing write-protected files. To skip the confirmation prompts, use the **-f** option:

```
rm -f file.txt
```

Instead of deleting specific files individually, you can use wildcards to remove multiple items with a single command. There are two types of wildcards – the asterisk (*****) and the question mark (**?**).

The asterisk represents multiple unknown characters, commonly used to delete files with a certain extension. For instance, the command below will delete all **TXT** files in the current working directory:

```
rm *.txt
```

You can also use the asterisk to delete all files beginning with a specific letter.

```
rm a*
```

In the example above, the asterisk represents all unknown characters following the letter **a**. The command will delete all files beginning with an **a**, regardless of their extensions, such as **amazon.txt**, **alligator.png**, and **aaron.zip**.

Meanwhile, the question mark wildcard represents a single character. Combining it with the asterisk wildcard lets you delete files with a single character extension, such as **S**, **O**, and **C**. Here's an example command:

```
rm *.?
```

> **Caution**
>
> Be careful when using wildcards, as you may end up accidentally deleting critical files. Before proceeding, check all the files in the directory using the **ls** command to ensure there are no important ones.

10. mkdir command

Use the **mkdir** command to create one or multiple directories and set their permissions. Ensure you are authorized to make a new folder in the parent directory. Here's the basic syntax:

```
mkdir [option] [directory_name]
```

To create a folder within a directory, use the path as the command parameter. For example, **mkdir music/songs** will create a **songs** folder inside **music**. Here are several common **mkdir** command options:

- **-p** – creates a directory between two existing folders. For example, **mkdir -p Music/2024/Songs** creates a new **202**4 directory.
- **-m** – sets the folder permissions. For instance, enter **mkdir -m777 directory** to create a directory with read, write, and execute permissions for all users.
- **-v** – prints a message for each created directory.

File Viewing and Manipulations

Viewing File Contents (`less`, `more`, `head`, `tail`)

11. `less` Command

`less` is a command-line pager utility used to view the contents of a file one screen at a time. It is particularly useful for large files because it doesn't load the entire file into memory.

```
less filename
```

- **Move Forward:** `Space` or `f`
- **Move Backward:** `b`

- **Go to Next Line:** `Enter` or `Down Arrow`
- **Go to Previous Line:** `Up Arrow`
- **Search Forward:** `/pattern` (then press `n` to go to the next match)
- **Search Backward:** `?pattern` (then press `N` to go to the previous match)
- **Quit:** `q`

```
less /var/log/syslog
```

12. `more` Command

`more` is another command-line pager similar to `less`, but it provides fewer features and is less flexible. It displays the contents of a file one screen at a time.

```
more filename
```

- **Move Forward:** `Space` or `f`
- **Go to Next Line:** `Enter`
- **Search Forward:** `/pattern`
- **Quit:** `q`

```
more /var/log/syslog
```

13. `head` Command

`head` is a command-line utility that displays the beginning of a file. By default, it shows the first 10 lines.

- **Display the First 10 Lines:**

  ```
  head filename
  ```

- **Display the First N Lines:**

  ```
  head -n N filename
  ```

- **First 10 Lines:**

  ```
  head /var/log/syslog
  ```

- **First 20 Lines:**

  ```
  head -n 20 /var/log/syslog
  ```

14. `tail` Command

tail` is a command-line utility that displays the end of a file. By default, it shows the last 10 lines. It is particularly useful for monitoring log files.

- **Display the Last 10 Lines:**

```
tail filename
```

- **Display the Last N Lines:**

```
tail -n N filename
```

- **Follow a File (Live Update):**

```
tail -f filename
```

- **Last 10 Lines:**

```
tail /var/log/syslog
```

- **Last 20 Lines:**

```
tail -n 20 /var/log/syslog
```

- **Follow a File:**

```
tail -f /var/log/syslog
```