



# ABC IT Education

WE'LL TAKE YOU FROM ZERO TO HERO IN A SNAP

## Linux Systems Administration

### Linux Fundamentals II

## Basic Linux Commands

- We are now going to look at some basic Linux commands.
- These commands are basic but they are very important Linux commands that we will see and use often.
- Before we start looking at the commands, it is necessary to mention here that **commands in Linux are case sensitive**, so a command 'pwd' not the same as 'PWD' nor 'Pwd' nor any other combination containing one or more uppercase letter(s).
- To make things simple, most commands, a majority of them, in Linux are all lowercase letters.
- There are many commands in Linux and we will see them as we go along, but we will start here with the following commands: 'pwd', 'ls', 'cd', 'mkdir', 'rm', 'rmdir', 'mv', 'cat', 'cp', 'echo', 'clear' and 'exit'

- 
- As a systems administrator, you will see many more commands than this, but this is a good place to start.
  - `'pwd'` – display present working directory
  - `'ls'` – list directory contents
  - `'cd'` – change directory
  - `'mkdir'` – make directory
  - `'rm'` – remove ( delete ) files and directory
  - `'rmdir'` – remove ( delete ) directory
  - `'mv'` – move ( rename ) file or directory
  - `'cat'` – concatenate ( display contents of file
  - `'cp'` – copy files or group of files or directory

- 
- 'echo' - display line of text/string that are passed as an argument
  - 'clear' - clear the terminal screen
  - 'exit' - used to **exit** the shell where it is currently running

## Command Details

- 'pwd' - The pwd command is a command line utility for printing the current working directory. It will print the full system path of the current working directory to standard output. By default the pwd command ignores symlinks, although the full physical path of a current directory can be shown with an option.
- 'pwd' command options
- pwd -L : print the symbolic path
- pwd -P : print the actual path

- 
- 'ls' - The ls command is one of the basic commands that any Linux user should know. It is used to list information about files and directories within the file system.

- **'ls' command options**

- ls -a list all files including hidden file starting with '.'

|            |                                   |
|------------|-----------------------------------|
| ls --color | colored list [=always/never/auto] |
|------------|-----------------------------------|

|       |                               |
|-------|-------------------------------|
| ls -d | list directories - with ' */' |
|-------|-------------------------------|

|       |                                   |
|-------|-----------------------------------|
| ls -F | add one char of */=>@  to entries |
|-------|-----------------------------------|

|       |                                |
|-------|--------------------------------|
| ls -i | list file's inode index number |
|-------|--------------------------------|

|              |                                          |
|--------------|------------------------------------------|
| <u>ls -l</u> | list with long format - show permissions |
|--------------|------------------------------------------|

|               |                                         |
|---------------|-----------------------------------------|
| <u>ls -la</u> | list long format including hidden files |
|---------------|-----------------------------------------|

|        |                                          |
|--------|------------------------------------------|
| ls -lh | list long format with readable file size |
|--------|------------------------------------------|

|               |                                      |
|---------------|--------------------------------------|
| <u>ls -ls</u> | list with long format with file size |
|---------------|--------------------------------------|

|              |                     |
|--------------|---------------------|
| <u>ls -t</u> | sort by time & date |
|--------------|---------------------|

|       |                        |
|-------|------------------------|
| ls -X | sort by extension name |
|-------|------------------------|

- 
- ls -r : list in reverse order
  - ls -R : list recursively directory tree
  - ls -s : list file size
  - ls -S : sort by file size
- 
- **'cd'** – The cd command is one of the most basic and frequently used commands when working on the Linux command line. The cd command, means “change directory” is used to change the current working directory in Linux .
  - **'cd' command options**
  - cd -L : Follow symbolic links. By default, cd behaves as if -L option is specified.

- 
- `d -P` : Don't follow symbolic links. In other words, when this option is specified and you try to navigate to a symlink that points to a directory, `cd` will change into the directory.
  - In its simplest form when used without any argument, `cd` will take you to your home directory.
  - **Absolute and Relative Path Names**
  - When specifying a directory you can use absolute or relative path names. Absolute or full path starts from the system root `/`, and relative path starts from your current directory.
  - When you log into your Linux system your current working directory defaults to your home directory. Assuming that *Downloads* directory exists in your home directory you can navigate to it by using the relative path to the directory:
  - `cd Downloads`

- 
- You can also navigate to the same directory by using its absolute path:
  - `cd /home/username/Downloads`
  - In short, if the path starts with a slash (/) it is the absolute path to the directory.
  - **The Parent Directory**
  - On Unix-like operating systems the current working directory is represented by a single dot (.). Two dots (..), one after the other, are representing the parent directory or in other words the directory immediately above the current one.
  - If you type `cd .` you will change into the current directory or to put it simple the command will do nothing.

- 
- Suppose you are currently in the `/usr/local/share` directory, to switch to the `/usr/local` directory (one level up from the current directory), you would type: `cd ../`
  - To move two levels up to the `/usr` directory (the parent's parent) you could run the following: `cd ../../`
  - Here is another example. Let's say you are in the `/usr/local/share` directory and you want to switch to the `/usr/local/src`. You can do that by typing: `cd ../src`
  - **Navigate to the Previous Directory**
  - To change back to the previous working directory pass the dash (-) character as an argument to the `cd` command: `cd -`



## ➤ **Navigate to the Home Directory**

- To navigate to your home directory simply type `cd`. Another way to return directly to your home directory is to use the tilde (`~`) character, as shown below: `cd ~`
- For example, if you want to navigate to the Downloads directory, which is inside your home directory you would type: `cd ~/Downloads`
- You can also navigate to another user's home directory using the following syntax: `cd ~username`
- **Directories with Space in Their Names**
- If the directory you want to change to has spaces in its name, you should either surround the path with quotes or use the backslash (`\`) character to escape the space:  
`cd 'Dir name with space'` or `cd Dir\ name\ with\ space`

- 
- In Linux systems, you can create new directories either from the command line or with the help of your desktop's file manager. The command that allows you to create directories (also known as folders) is **mkdir**.
  - **Linux mkdir Command Syntax**
  - The syntax for the Linux **mkdir** command is as follows:
  - `mkdir [OPTION] [DIRECTORY]`
  - The command can take one or more directory names as its arguments.
  - To create a directory with the **mkdir** command you need to have write permissions on the parent directory. Otherwise, you will receive a permission denied error.



## ➤ How to Create a New Directory

- To create a directory in Linux pass the name of the directory as the argument to the **mkdir** command. For example, to create a new directory newdir you would run the following command: `mkdir newdir`
- You can verify that the directory was created by listing the contents using the `ls` command: `ls -l`
- Output: `drwxrwxr-x 2 username username 4096 Jan 20 03:39 newdir`
- When providing only the directory name, without the full path, it will be created in the current working directory.
- The current working directory is the directory from which you are running the commands. To change the current working directory, use the `cd` command.

- 
- To create a directory in another location you'll need to provide the absolute or relative file path to the parent directory. For example, to create a new directory in the /tmp directory you would type: `mkdir /tmp/newdir`
  - If you try to create a directory in a parent directory where the user does not have sufficient permissions you will receive Permission denied error: `mkdir /root/newdir`
  - `mkdir: cannot create directory '/root/newdir': Permission denied`
  - The `-v` (`--verbose`) option tells `mkdir` to print a message for each created directory.



## ➤ How to Create Parent Directories

- A parent directory is a directory that is above another directory in the directory tree. To create parent directories use the -p option.
- Let's say you want to create a directory /home/linuxize/Music/Rock/Gothic:
- `mkdir /home/linuxize/Music/Rock/Gothic`
- If any of the parent directories don't exist you will get an error as shown below:
- `mkdir: cannot create directory '/home/linuxize/Music/Rock/Gothic': No such file or directory`
- Instead of creating missing parent directories one by one, use the -p option: `mkdir -p /home/linuxize/Music/Rock/Gothic`

- 
- When mkdir is invoked with the -p option, it will create the directory only if it doesn't exist.
  - If you try to create a directory that already exists and the -p option is not provided, mkdir will print File exists error:  
mkdir newdir
  - mkdir: cannot create directory 'newdir': File exists
  - **How to Set Permissions when Creating a Directory**
  - To create a directory with specified permissions, use the -m (-mode) option. The syntax for assigning permissions is the same as with the chmod command.
  - In the following example, the new directory will have 700 permissions which means that only the user who created the directory will be able to access it:
  - mkdir -m 700 newdir

- 
- If the -m option is not passed to mkdir the newly created directories will have 775 or 755 permissions, depending on the umask value.
  - **How to Create Multiple Directories**
  - To create multiple directories, specify the directories' names as the command arguments, separated by space:
  - `mkdir dir1 dir2 dir3`
  - The mkdir command also allows you to create a complex directory tree with one command:
  - `mkdir -p Music/{Jazz/Blues,Folk,Disco,Rock/{Gothic,Punk,Progressive},Classical/Baroque/Early}`

- 
- The command above will create the following directory tree:

```
Music/  
|-- Classical  
|   |-- Baroque  
|       |-- Early  
|-- Disco  
|-- Folk  
|-- Jazz  
|   |-- Blues  
|-- Rock  
|-- Gothic  
|-- Progressive  
|-- Punk
```

- 
- rm is a command-line utility for removing files and directories. It is one of the essential commands that every Linux user/admin should be familiar with.
  - In here we will explain how to use the rm command through examples and explanations of the most common rm options.
  - **How to Use the rm Command**
  - The general syntax for the rm (remove) command is as follows:
    - rm [OPTIONS]... FILE...
    - By default, when executed without any option, rm doesn't remove directories and doesn't prompt the user for whether to proceed with the removal of the given files.

- 
- To delete a single file, use the `rm` command followed by the file name as an argument: `rm filename`
  - If you don't have write permissions on the parent directory, you will get "Operation not permitted" error.
  - If the file is not write protected, it will be removed without notice. On success, the command doesn't produce any output and returns zero.
  - When removing write protected files, the command will prompt you for confirmation, as shown below:
  - `rm: remove write-protected regular empty file 'filename'?`
  - Type `y` and hit Enter to remove the file.
  - The `-f` option tells `rm` never to prompt the user and to ignore nonexistent files and arguments.
  - `rm -f filename`

- 
- If you want to get information about what is being removed, use the -v (verbose) option: `rm -v filename`
  - Output: removed 'filename'
  - **Removing Multiple Files**
  - `rm` allows you to delete multiple files at once. To do that, pass the filenames as arguments separated by space:
  - `rm filename1 filename2 filename3`
  - You can use regular expressions to match multiple files. For example, to remove all .png files in the current directory, you would type: `rm *.png`
  - When using regular expressions, before running the `rm` command, it is always a good idea to list the files with the `ls` command so that you can see which files will be deleted.



## ➤ Removing Directories (Folders)

➤ To remove one or more **empty directories** use the -d option: `rm -d dirname` or simply use the command **rmdir**

➤ `rm -d` is functionally identical to the **rmdir** command.

➤ To remove non-empty directories and all the files within them recursively, use the -r (recursive) option:

➤ `rm -r dirname`

## ➤ Prompt Before Removal

➤ The -i option tells `rm` to prompt the user for each given file before removing it: `rm -i filename1 filename2`

➤ To confirm type `y` and press Enter:

➤ `rm: remove regular empty file 'filename1'?`

➤ `rm: remove regular empty file 'filename2'?`

- 
- When removing more than three files or recursively removing a directory, to get a single prompt for the entire operation, use the -I option:
  - `rm -i filename1 filename2 filename3 filename4`
  - You will be asked to confirm the removal of all given files and directories: `=> rm: remove 4 arguments?`
  - `rm -rf`
  - If the given directory or a file within the directory is write-protected, the `rm` command will prompt you to confirm the operation. To remove a directory without being prompted, use the -f option: `rm -rf dirname`
  - The `rm -rf` command is very dangerous and should be used with extreme caution!

- 
- The mv command (short from move) is used to rename or move files and directories from one location to another.
  - **How to Use the mv Command**
  - mv [OPTIONS] source destination
  - The source can be one or more files or directories and destination can be a single file or directory.
  - If you specify multiple files or directories as a source, the destination must be a directory. In this case, the source files are moved to the target directory.
  - If you specify a single file as source, and the destination target is an existing directory then the file is moved to the specified directory.

- 
- If you specify a single file as source, and a single file as destination target then you're renaming the file.
  - If the source is a directory and destination doesn't exist source will be renamed to destination, otherwise it be moved inside the destination directory.
  - To move a file or directory you need to have write permissions on both source and destination. Otherwise, you will receive a permission denied error.
  - For example, to move the file file1 from the current working directory to the /tmp directory you would run:  
`$ mv file1 /tmp`
  - To rename a file specify the destination file name:  
`$ mv file1 file2`

- 
- The command for moving directories is the same as when moving files. In the following example if the dir2 directory exists the command will move dir1 inside dir2. If dir2 doesn't exist, dir1 will be renamed to dir2: `$ mv dir1 dir2`
  - **Moving Multiple Files and Directories**
  - To move multiple files and directories, specify the files you want to move as the source. For example to move the files file1 and file2 to the dir1 directory you would type:  
`$ mv file1 file2 dir1`
  - The mv command also allows you to use pattern matching. For example, to move all pdf files from the current directory to the ~/Documents directory you would use: `$ mv *.pdf ~/Documents`



## ➤ **mv Command Options**

- The mv command accepts options that affect default command behavior. In some Linux distributions mv may be an alias to the mv command with a custom set of options. For example in CentOS mv is an alias to mv -i. You can find whether mv is an alias using the type command: `$ type mv`
- If mv is alias the output will look something like this:
- Output: mv is aliased to `mv -i`
- Note: If you specify conflicting options, the option specified last takes precedence.
- **Prompt before overwriting**
- By default if the destination file exists it will be overwritten. To prompt for confirmation use the -i option:

- 
- `mv -i file1 /tmp`
  - Output: `mv: overwrite '/tmp/file1'?`
  - To overwrite the file type the character `y` or `Y`.
  - **Force overwriting**
  - If you try to overwrite a read only file the `mv` command will prompt you whether you want to overwrite the file:
  - `mv -i file1 /tmp`
  - `mv: replace '/tmp/file1', overriding mode 0400 (r-----)?`
  - To avoid being prompted use the `-f` options:
  - `mv -f file1 /tmp`
  - This option is especially useful when you need to overwrite multiple read only file.



- **Do not overwrite existing files**

- The -n option tells mv never to overwrite any existing file:

- `$ mv -n file1 /tmp`

- If a file1 exists the command above will do nothing, otherwise it will move the file to the /tmp directory.

- **Backing up files**

- If the destination file exists you can create a backup of it using the -b option: `$ mv -b file1 /tmp`

- The backup file will have the same name as the original file with a tilde (~) appended to it.

- Use the ls command to verify that the backup was created:

- `$ ls /tmp/file1*`

- Output: `/tmp/file1 /tmp/file1~`



- **Verbose output**

- Another option that can be useful is -v. When using this option the command will print the name of each moved file:

- `$ mv -i file1 /tmp`

- Output: renamed 'file1' -> '/tmp/file1'

- **'cat' Command**

- The cat command is one of the most widely used commands in Linux and its name comes from its functionality to concatenate files. It can read and concatenate files, writing their contents to the standard output. If no file is specified or if the input file name is specified as a single hyphen (-) it reads from the standard input.

- 
- Cat is most commonly used to display the contents of one or multiple text files, combine files by appending the contents of one file to the end of another file, and create new files.
  - **Cat Command Syntax**
  - The cat utility expressions take the following form:
  - `cat [OPTIONS] [FILE_NAMES]`
  - **OPTIONS** - cat options. Use `cat --help` to view all available options.
  - **FILE\_NAMES** - Zero or more file names.



## ➤ **Displaying File Contents**

- The most basic and common usage of the cat command is to read the contents of files.

- For example, the following command will display the contents of the /etc/issue file in the terminal: `$ cat /etc/issue`

## ➤ **Redirect Contents of File**

- Instead of displaying the output to stdout (on the screen) you can redirect it to a file.
- The following command will copy the contents of file1.txt to file2.txt using the (>) operator : `$ cat file1.txt > file2.txt`

- 
- If the file2.txt file doesn't exist, the command will create it. Otherwise, it will overwrite the file.
  - Use the (>>) operator to append the contents of file1.txt to file2.txt : `$ cat file1.txt >> file2.txt`
  - Same as before, if the file is not present, it will be created.
  - **Print Line Numbers**
  - To display contents of a file with line numbers, use the -n option: `$ cat -n /etc/lsb-release`
  - 1     `DISTRIB_ID=Ubuntu`
  - 2     `DISTRIB_RELEASE=18.04`
  - 3     `DISTRIB_CODENAME=bionic`
  - 4     `DISTRIB_DESCRIPTION="Ubuntu 18.04.1 LTS"`



- **Suppress Repeated Empty Lines**

- Use the -s option to omit the repeated empty output lines:

- `$ cat -s file.txt`

- **Display TAB characters**

- The -T option allows you to visually distinguish between tabs and spaces. `$ cat -T /etc/hosts`

- Output

- `127.0.0.1^Ilocalhost`

- `127.0.1.1^Iubuntu1804.localdomain`

- The TAB characters will be displayed as ^I.



## ➤ Display End of Lines

➤ To display the invisible line ending character use the -e argument: `$ cat -e /etc/lsb-release`

➤ Output:

➤ `DISTRIB_ID=Ubuntu$`

➤ `DISTRIB_RELEASE=18.04$`

➤ `DISTRIB_CODENAME=bionic$`

➤ `DISTRIB_DESCRIPTION="Ubuntu 18.04.1 LTS"$`

➤ The Line endings will be displayed as \$.



## ➤ Concatenating Files

- When passing two or more file names as arguments to the cat command the contents of the files will be concatenated. cat reads the files in the sequence given in its arguments and displays the file's contents in the same sequence.
- For example, the following command will read the contents of file1.txt and file2.txt and display the result in the terminal: `$ cat file1.txt file2.txt`
- You can concatenate two or more text files and write them to a file.
- The following command will concatenate the contents of file1.txt and file2.txt and write them to a new file combinedfile.txt using the (>) operator :
- `$ cat file1.txt file2.txt > combinedfile.txt`

- 
- If the combinedfile.txt file doesn't exist, the command will create it. Otherwise, it will overwrite the file.
  - To concatenate the contents of file1.txt and file2.txt and append the result to file3.txt to use the (>>) operator: `$ cat file1.txt file2.txt >> file3.txt`
  - If the file is not present, it will be created.
  - When concatenating files with cat, you can use the same arguments as shown in the previous section.
  - **Creating Files**
  - Creating small files with cat it often easier than opening a text editor such as nano, Vim, Sublime Text or Visual Studio Code.

- 
- To create a new file, use the cat command followed by the redirection operator (>) and the name of the file you want to create. Press Enter, type the text and once you are done, press the CTRL+D to save the file.
  - In the following example, we are creating a new file named file1.txt: `$ cat > file1.txt`
  - If a file named file1.txt is present, it will be overwritten. Use the '>>' operator to append the output to an existing file.
  - `cat >> file1.txt`



## ➤ **'cp' Command**

- Copying files and directories is one of the most common tasks you'll perform on a daily basis.
- cp is a command-line utility for copying files and directories on Unix and Linux systems. In this article, we will explain how to use the cp command.
- **How to Use cp command**
- The general syntax for the cp command is as follows:
- `cp [OPTIONS] SOURCE... DESTINATION`
- The SOURCE can have one or more files or directories as arguments and the DESTINATION can have a single file or directory as an argument.

- 
- When the SOURCE and DESTINATION arguments are both files the cp command copies the first file to the second one. If the file doesn't exist the command will create it.
  - When the SOURCE has multiple files or directories as arguments, the DESTINATION argument must be a directory. In this case, the SOURCE files and directories are moved to the DESTINATION directory.
  - When the SOURCE and DESTINATION arguments are both directories the cp command will copy the first directory into the second one.
  - To copy files and directories you must have at least read permissions on the source file and write permission on the destination directory. Otherwise, you will get a permission denied error.

- 
- Copying Files with cp Command
  - The most basic scenario of using cp is copying a file in the current working directory. For example, to copy a file named file.txt to file\_backup.txt you would run the following command:
    - `$ cp file file_backup` or `$ cp file{,}_backup}`
    - To copy a file to another directory specify the absolute or the relative path to the destination directory. When only the directory name is specified as a destination the copied file will have the same name as the original file.
    - In the following example we are copying the file file.txt to the /backup directory:
      - `$ cp file.txt /backup`

- 
- If you want to copy the file under a different name you need to specify the desired file name. The command below will copy the file to the specified directory as new\_file.txt.
  - `cp file.txt /backup/new_file.txt`
  - By default, if the destination file exists it will be overwritten. The `-n` option tells `cp` to not overwrite an existing file. To prompt for confirmation use the `-i` option.
  - `cp -i file.txt file_backup.txt`
  - If you want to copy the file only if it is newer than the destination use the `-u` option:
  - `cp -u file.txt file_backup.txt`

- 
- When copying a file, the new file will be owned by the user running the command. Use the -p option to preserve the file mode, ownership and timestamps: `$ cp -p file.txt file_backup.txt`
  - Another option that can be useful is -v, which tells cp to print verbose output: `$ cp -v file.txt file_backup.txt`
  - Output: 'file.txt' -> 'file\_backup.txt'
  - **Copying Directories with cp Command**
  - To copy a directory, including all its files and subdirectories, use the -R or -r option. In the following example we are copying the directory Pictures to Pictures\_backup:
  - `cp -R Pictures Pictures_backup`

- 
- The command above will create a destination directory and recursively copy all files and subdirectories from the source to the destination directory.
  - If the destination directory already exists the source directory itself and all its content will be copied to the destination directory. To copy only the files and subdirectories but not the target directories use the -T option: `$ cp -RT Pictures Pictures_backup`
  - Another way to copy only the directory's content but not the directory itself is to use the wildcard character (\*). The disadvantage of the following command is that it will not copy the hidden files and directories (the ones starting with a dot .): `$ cp -RT Pictures/* Pictures_backup/`

- 
- All the options we used in the previous section when copying files can be also used when copying directories. The main difference is that when copying directories you always need to use the -R option.
  - **Copy Multiple Files and Directories**
  - To copy multiple files and directories at once specifying their names and use the destination directory as the last argument: `$ cp file.txt dir file1.txt dir1`
  - Note: When copying multiple files the destination must be a directory.



## ➤ **'echo' Command**

- The echo command is one of the most basic and frequently used commands in Linux. The arguments passed to echo are printed to the standard output.
- echo is commonly used in shell scripts to display a message or output the results of other commands.
- echo is a shell builtin in Bash and most of the other popular shells like Zsh and Ksh. Its behavior is slightly different from shell to shell.
- There is also a standalone `/usr/bin/echo` utility, but usually, the shell built-in version will take precedence. We will cover the Bash builtin version of echo.

- 
- The syntax for the echo command is as follows:
  - `echo [-neE] [ARGUMENTS]`
  - When the -n option is used, trailing newline is suppressed.
  - If the -e option is given, the following backslash-escaped characters will be interpreted:
    - `\\` - Displays a backslash character.
    - `\a` - Alert (BEL)
    - `\b` - Displays a backspace character.
    - `\c` - Suppress any further output
    - `\e` - Displays an escape character.
    - `\f` - Displays a form feed character.
    - `\n` - Displays a new line.
    - `\r` - Displays a carriage return.
    - `\t` - Displays a horizontal tab.
    - `\v` - Displays a vertical tab.

- 
- The -E option disables the interpretation of the escape characters. This is the default.
  - There are a few points to consider when using the echo command.
  - The shell will substitute all variables, wildcard matching, and special characters before passing the arguments to the echo command.
  - Although not necessary, it is a good programming practice to enclose the arguments passed to echo in double or single quotes.
  - When using single quotes " the literal value of each character enclosed within the quotes will be preserved. Variables and commands will not be expanded.



## ➤ **echo Examples**

- The examples below show how to use the echo command:
- Display a line of text on standard output.
- `echo Hello, World!`, Output: Hello, World!
- **Display a line of text containing a double quote.**
- To print a double quote, enclose it within single quotes or escape it with the backslash character.
- `$ echo 'Hello "Linuxize"'` or `$ echo "Hello \"Linuxize\""`
- Output: Hello "Linuxize"
- **Display a line of text containing a single quote.**
- To print a single quote, enclose it within double quotes or use the ANSI-C Quoting.

- 
- `$ echo "I'm a Linux user." or $ echo $'I\'m a Linux user.'`
  - Output: I'm a Linux user.
  - **Display a message containing special characters.**
  - Use the `-e` option to enable the interpretation of the escape characters.
  - `$ echo -e "You know nothing, Jon Snow.\n\t- Ygritte"`
  - Output: You know nothing, Jon Snow.  
- Ygritte
  - **Pattern matching characters.**
  - The `echo` command can be used with pattern matching characters, such as the wildcard characters. For example, the command below will return the names of all the `.php` files in the current directory.

- 
- `$ echo The PHP files are: *.php`
  - Output:
  - The PHP files are: index.php contact.php functions.php
  - **Redirect to a file**
  - Instead of displaying the output to the screen, you can redirect it to a file using the `>`, `>>` operators.
  - `echo -e 'The only true wisdom is in knowing you know nothing.\nSocrates' >> /tmp/file.txt`
  - If the file.txt doesn't exist, the command will create it. When using `>` the file will be overwritten, while the `>>` will append the output to the file.

- 
- Use the cat command to view the content of the file:
  - `$ cat /tmp/file.txt`
  - Output:
  - The only true wisdom is in knowing you know nothing.  
Socrates
  - **Displaying variables**
  - echo can also display variables. In the following example, we'll print the name of the currently logged in user:
  - `$ echo $USER` Output: `< currently logged in user>`
  - `$USER` is a shell variable that holds your username.



- **Displaying output of a command**

- Use the `$(command)` expression to include the command output in the echo's argument. The following command will display the current date:

- `$ echo "The date is: $(date +%D)"`

- Output: The date is: 04/17/19

- **'clear' Command**

- clear is a standard Linux command to clear the terminal screen. This command will ignore any command-line parameters that may be present. Also, the clear command doesn't take any argument and it is almost similar to cls command on a number of other operating systems.

- Syntax: `$ clear`



## ➤ **'exit' Command**

- exit command in linux is used to exit the shell where it is currently running. It takes one more parameter as [N] and exits the shell with a return of status N. If n is not provided, then it simply returns the status of last command that is executed.
- Syntax: \$ exit [n] **Note** n is optional



## ➤ **'wc' Command**

- The 'wc' command or 'word count' command allows you to count the number of lines, words, characters, and bytes of each given file or standard input and print the result.
- How to Use the wc Command
- The syntax for the wc command is as follows:
- `wc OPTION... [FILE]...`
- The wc command can accept zero or more input FILE names. If no FILE is specified, or when FILE is -, wc will read the standard input. A word is a string of characters delimited by a space, tab, or newline.

- 
- When the 'wc' command is used without any options, the wc command will print four columns, the number of lines, words, byte counts and the name of the file for each file passed as an argument.
  - When using the standard input the fourth column (filename) is not displayed.
  - Example, the following command will display information about the virtual file /proc/cpuinfo: `$ wc /proc/cpuinfo`
  - The output will look something like the following:
  - 448 3632 22226 /proc/cpuinfo, meaning
  - 448 is the number of lines.
  - 3632 is the number of words.
  - 22226 is the number of characters.

- 
- When using the standard input, the file name is not shown:
  - `$ wc < /proc/cpuinfo`
  - **Note:** `<` means redirect file to standard input, similar to typing the file in standard input.
  - Output is:
  - `448 3632 22226`
  - To display information about more than one file, pass the filenames, as arguments, separated by space:
  - `$ wc /proc/cpuinfo /proc/meminfo`
  - The command will give you information about each file and a line including total statistics:

- 
- Output:
  - 448 3632 22226 /proc/cpuinfo
  - 49 143 1363 /proc/meminfo
  - 497 3775 23589 total
  - The options below allow you to select which counts are printed.
  - -l, --lines - Print the number of lines.
  - -w, --words - Print the number of words.
  - -m, --chars - Print the number of characters.
  - -c, --bytes - Print the number of bytes.
  - -L, --max-line-length - Print the length of the longest line.

- 
- When using multiple options counts are printed in the following order: newline, words, characters, bytes, maximum line length.
  - For example, to display only the number of words you would use: `$ wc -w /proc/cpuinfo`
  - Output:
  - `3632 /proc/cpuinfo`
  - Here is another example that will print the number of lines and the length of the longest line. `$ wc -lL /proc/cpuinfo`
  - Output:
  - `448 792 /proc/cpuinfo`



## ➤ **Count the Number of Lines**

- The wc command is mostly used with the -l option to count only the number of lines in a text file. For example, to count the number of lines in the /etc/passwd file you would type:

- `$ wc -l /etc/passwd`

- The first column is the number of lines and the second one is the name of the file: 44 /etc/passwd

## ➤ **Count the Number of Words**

- To count only the number of words in a text file use wc -w followed by the file name. The following example counts the number of words in the ~/Documents/file.txt file:

- The number of words is shown in the first column:

- `513 /home/linuxize/Documents/file.txt`



- **Counting Files in the Current Directory**

- The find command passes a list of all files in the current directory with each file name on a single line to the 'wc' command, which counts the number of lines and prints the result: `$ find . -type f | wc -l`

- **Count the number of users**

- In the example below wc is used to count the number of lines from the output of the getent command .
- `$ getent passwd | wc -l`

- 
- 'getent' Linux command helps the user to get the entries in a number of important text files called databases. This includes the passwd and the group of databases which stores the user information.
  - 'getent' is a common way to look up user details on Linux. Since getent uses the same name of service as the system, getent will show all information, including that gained from the network information sources such as LDAP.
  - The databases usually searched: ahosts, ahostsv4, ahostsv6, aliases, ethers (Ethernet addresses), group, gshadow, hosts, netgroup, networks, passwd, protocols, rpc, services, and shadow.

- 
- Example 1: Fetch the list of user accounts on a Linux system stored in the 'passwd' database. This will show all the user accounts, regardless of the type of name service being used.
  - Syntax: `$ getent database [key ...]`
  - **'grep' Command**
  - The 'grep' command which stands for "global regular expression print" is one of the most powerful and commonly used commands in Linux.
  - 'grep' searches one or more input files for lines that match a given pattern and writes each matching line to standard output. If no files are specified, 'grep' reads from the standard input which is usually the output of another command.



## ➤ **Grep Command Syntax**

- The 'grep' command expressions take the following form:
- `$ grep [OPTIONS] PATTERN [FILE...]`
- The items in square brackets are optional.
- **OPTIONS** - Zero or more options. 'grep' provides a number of options that control its behavior.
- **PATTERN** - Search pattern.
- **FILE** - Zero or more input file names.



## ➤ **How to use Grep to Search for a String in Files**

- The most basic usage of the 'grep' command is to search for a string (text) in a file.
- Example: display the lines from the /etc/passwd file containing the string bash you use the command:
  - `$ grep bash /etc/passwd`
  - Output:
    - `root:x:0:0:root:/root:/bin/bash`
    - `cwmonie:x:1000:1000:cwmonie:/home/cwmonie:/bin/bash`
  - Note: If the string includes spaces, enclose it in single or double quotation marks as follows:
    - `$ grep "Gnome Display Manager" /etc/passwd`



## ➤ **Grep Invert Match (Exclude)**

- To display the lines that do not match a pattern, use the -v ( or --invert-match) option.
- Example: display the lines from the /etc/passwd file that do not contain the string nologin:
- `$ grep -v nologin /etc/passwd`
- Output:
- `root:x:0:0:root:/root:/bin/bash`
- `colord:x:124:124::/var/lib/colord:/bin/false`
- `git:x:994:994:git daemon user:/usr/bin/git-shell`
- `cwmonie:x:1000:1000:linuxize:/home/cwmonie:/bin/bash`



## ➤ Use grep to Search for a String in Command Output

- Instead of specifying input files you can **pipe** the output of another command to 'grep', and then display only lines matching a given pattern.
- For example, to find out the processes that contain bash use the command below:
- `$ ps -ef | grep color:` will display all the process lines that include color, including the grep process.
- You can also chain multiple pipes in one command. As you can see in the output above there is also a line containing the grep process. If you don't want that line to be shown pass the output to another grep instance as shown below.
- `$ ps -ef | grep bash | grep -v grep`



## ➤ **Grep Recursive Search**

- To recursively search for a pattern, use the -r option (or --recursive). This will search through all files in the specified directory, skipping the symlinks ( symbolic links ) that are encountered recursively. To follow all symbolic links, use the -R option (or --dereference-recursive).
- In the following example we are searching for the string localhost in all files inside the our home directory:
- `grep -r localhost ~`
- The command will print the matching lines prefixed by the full path to the file.

- 
- If instead of -r you use -R option grep will follow all symbolic links:
  - `$ grep -R linuxize.com /etc`
  - **Grep Show Only Filename**
  - To print only the names of files containing the matched pattern you can use the -l ( or --files-with-matches) option.
  - For example to search through all files ending with .conf in the [current working directory]: `$ grep -l localhost *.conf`
  - **Note:** The -l option is usually used in combination with the recursive option -R:
  - `grep -Rl localhost /tmp`



## ➤ **Grep Case Insensitive**

- By default the grep command is case sensitive which means that uppercase and lowercase characters are treated as distinct.
- To ignore case when searching, use the -i option (or --ignore-case).
- For example, when searching for Zebra without any option the following command will not show any output i.e there are no matching lines.
- `$ grep Zebra /usr/share/words`
- But if you perform a case insensitive search using the -i option, it will match both upper and lower case letters:
- `grep -i Zebra /usr/share/words`

- 
- Specifying "Zebra" will match "zebra", "ZEbrA" or any other combination of upper and lower case letters for that string, like zebra, zebra's, zebras etc
  - **Grep Show Line Numbers**
  - To show the number of the lines which contains a string that matches a pattern, use the -n ( or --line-number) option. When using this option grep will print the matches to standard output prefixed with the line number it was found on.
  - For example to display the lines from the /etc/services file containing the string bash prefixed with the matching line number you can use the following command:
  - `grep -n 10000 /etc/services`

- 
- The output below shows us that the matches are found on lines 10423 and 10424.

- 10423:ndmp                      10000/tcp

- 10424:ndmp                      10000/udp

- **Grep Count Matches**

- To print a count of matching lines to standard output, use the -c ( or --count) option.

- In the example below, we are counting the number of accounts that have /usr/bin/zsh as a shell.

- \$ grep -c '/usr/bin/zsh' /etc/passwd

- Output: 4



## ➤ **Grep Multiple Strings (Patterns)**

- Two or more search patterns can be joined by the OR operator |.
- Grep interprets the pattern as a basic regular expression where the meta-characters like | lose their special meaning and the back slashed versions must be used.
- In the example below we are searching all occurrences of the words fatal, error and critical in the Nginx log error file:
- `grep 'fatal\|error\|critical' /var/log/nginx/error.log`
- If you use the extended regular expression option -E (or --extended-regexp) then the operator | should not be escaped, as shown below:
- `$ grep -E 'fatal|error|critical' /var/log/nginx/error.log`



## ➤ **'head' Command**

- The head command prints the first lines (10 by default) of one or more files or piped data to standard output.
- The syntax for the head command is as follows:
- head [OPTION]... [FILE]...
- OPTION - head options, we'll go over the most common options.
- FILE - Zero or more input file names. If no FILE is given, or when FILE is -, head will read the standard input.
- How to Use the Head Command
- The simplest form is when used without any option, the head command will display the first 10 lines.
- `$ head filename.txt`



## ➤ **How to Display a Specific Number of Lines**

- Use the -n (--lines) option followed by an integer specifying the number of lines to be shown:
- `$ head -n <NUMBER> filename.txt`
- You can omit the letter n and use just the hyphen (-) and the number (with no space between them).
- To display the first 30 lines of a file named filename.txt you would type:
- `$ head -n 30 filename.txt`
- The following will produce the same result as the above commands:
- `$ head -30 filename.txt`



## ➤ How to Display a Specific Number of Bytes

- The `-c` (`--bytes`) option allows to print a specific number of bytes: `$ head -c <NUMBER> filename.txt`
- For example to display the first 100 bytes of data from the file named `filename.txt` you would type:
  - `$ head -c 100 filename.txt`
- You can use a multiplier suffix after the number to specify the number of bytes to show. `b` multiplies by 512, `KB` multiplies by 1000, `K` multiplies by 1024, `MB` multiplies by 1000000 ( $1000 * 1000$ ), `M` multiplies by 1048576 ( $1024 * 1024$ ), and so on for `GB`, `G` and also for `T`, `P`, `E`, `Z`, `Y`.
- The following command will display the first five kilobytes (5120) of the file `filename.txt`:
  - `$ head -c 5K filename.txt`



## ➤ **How to Display Multiple Files**

- If multiple files are provided as input to the head command, it will display the first ten lines from each provided file.
- `$ head filename1.txt filename2.txt`
- You can use the same options as when displaying one file.
- This example shows the first 20 lines of the files filename1.txt and filename2.txt:
- `head -n 20 filename1.txt filename2.txt`
- When more than one file is used, the output precede each with a header showing the file name.



## ➤ **How to Use Head with Other Commands**

- The head command can be used in combination with other commands by redirecting the standard output from/to other commands ( utilities ) using pipes ( | ).
- The following command will hash the \$RANDOM environment variable, display the first 32 bytes and display 24 characters random string:
- `echo $RANDOM | sha512sum | head -c 24 ; echo`



## ➤ **'tail' Command**

- The tail command displays the bottom part (10 lines by default) of one or more files or piped data. It can be also used to monitor the file changes in real time.
- One of the most common uses of the tail command is to watch and analyze logs and other files that change over time, usually combined with other tools like grep.
- The syntax for the tail command is as follows:
- `tail [OPTION]... [FILE]...`
- **OPTION** - tail options, we'll go over the most common options.
- **FILE** - Zero or more input file names. If no FILE is given, or when FILE is -, tail will read the standard input.



- How to Use the Tail Command

- The simplest form is when used without any option, the tail command will display the last 10 lines.

- `$ tail filename.txt`

- **How to Display a Specific Number of Lines**

- Use the `-n` (`--lines`) option followed by an integer specifying the number of lines to be shown:

- `$ tail -n <NUMBER> filename.txt`

- You can omit the letter `n` and use just the hyphen (`-`) and the number (with no space between them).

- To display the first 30 lines of a file named `filename.txt` you would type:

- `$ tail -n 30 filename.txt`

- 
- The following will produce the same result as the above commands:

- `$ tail -30 filename.txt`

- **How to Display a Specific Number of Bytes**

- The `-c` (`--bytes`) option allows to print a specific number of bytes: `$ tail -c <NUMBER> filename.txt`

- For example to display the last 100 bytes of data from the file named `filename.txt` you would type:

- `$ tail -c 100 filename.txt`

- You can use a multiplier suffix after the number to specify the number of bytes to show. `b` multiplies by 512, `KB` multiplies by 1000, `K` multiplies by 1024, `MB` multiplies by 1000000 ( $1000 * 1000$ ), `M` multiplies by 1048576 ( $1024 * 1024$ ), and so on for `GB`, `G` and also for `T`, `P`, `E`, `Z`, `Y`.

- 
- The following command will display the last five kilobytes (5120) of the file filename.txt:

- `$ tail -c 5K filename.txt`

- **How to Watch a File for Changes**

- To monitor a file for changes use the -f (--follow) option:

- `tail -f filename.txt`

- This option is particularly useful for monitoring log files. For example, to display the last 10 lines of the `/var/log/nginx/error.log` file, and monitor the file for updates you would use:

- `tail -f /var/log/nginx/error.log`

- 
- To interrupt the tail command while it is watching a file, press Ctrl+C.
  - To keep monitoring the file when it is recreated, use the -F option.
  - `tail -F filename.txt`
  - This option is useful in situations when the tail command is following a log file that rotates. When used with -F option the tail command will reopen the file the as soon as it became available again.



## ➤ **How to Display Multiple Files**

- If multiple files are provided as input to the head command, it will display the last ten lines from each provided file.
- `$ tail filename1.txt filename2.txt`
- You can use the same options as when displaying one file.
- This example shows the last 20 lines of the files filename1.txt and filename2.txt:
- `tail -n 20 filename1.txt filename2.txt`
- When more than one file is used, the output precede each with a header showing the file name.



## ➤ **How to Use Tail with Other Commands**

- The tail command can be used in combination with other commands by redirecting the standard output from/to other utilities using pipes.
- For example to monitor the apache access log file and only display those lines that contain the IP address 192.168.42.12 you would use:
  - `tail -f /var/log/apache2/access.log | grep 192.168.42.12`
- The following ps command will display the top ten running processes sorted by CPU usage:
  - `ps aux | sort -nk +3 | tail -5`



## ■ **Commands variants**

- There are 3 commands, egrep, fgrep and tree, that are similar to some commands we have already seen. Research the 3 commands so you are aware of them because there may be situations where they may be better than their variants



# ABC IT Education

WE'LL TAKE YOU FROM ZERO TO HERO IN A SNAP

## Linux Systems Administration

### Linux Fundamentals II

## Basic Linux Commands

- We are now going to look at some basic Linux commands.
- These commands are basic but they are very important Linux commands that we will see and use often.
- Before we start looking at the commands, it is necessary to mention here that **commands in Linux are case sensitive**, so a command 'pwd' not the same as 'PWD' nor 'Pwd' nor any other combination containing one or more uppercase letter(s).
- To make things simple, most commands, a majority of them, in Linux are all lowercase letters.
- There are many commands in Linux and we will see them as we go along, but we will start here with the following commands: 'pwd', 'ls', 'cd', 'mkdir', 'rm', 'rmdir', 'mv', 'cat', 'cp', 'echo', 'clear' and 'exit'

- 
- As a systems administrator, you will see many more commands than this, but this is a good place to start.
  - `'pwd'` – display present working directory
  - `'ls'` – list directory contents
  - `'cd'` – change directory
  - `'mkdir'` – make directory
  - `'rm'` – remove ( delete ) files and directory
  - `'rmdir'` – remove ( delete ) directory
  - `'mv'` – move ( rename ) file or directory
  - `'cat'` – concatenate ( display contents of file
  - `'cp'` – copy files or group of files or directory

- 
- 'echo' - display line of text/string that are passed as an argument
  - 'clear' - clear the terminal screen
  - 'exit' - used to **exit** the shell where it is currently running

## Command Details

- 'pwd' - The pwd command is a command line utility for printing the current working directory. It will print the full system path of the current working directory to standard output. By default the pwd command ignores symlinks, although the full physical path of a current directory can be shown with an option.
- 'pwd' command options
- pwd -L : print the symbolic path
- pwd -P : print the actual path

- 
- 'ls' - The ls command is one of the basic commands that any Linux user should know. It is used to list information about files and directories within the file system.

- **'ls' command options**

- ls -a list all files including hidden file starting with '.'

|            |                                   |
|------------|-----------------------------------|
| ls --color | colored list [=always/never/auto] |
|------------|-----------------------------------|

|       |                               |
|-------|-------------------------------|
| ls -d | list directories - with ' */' |
|-------|-------------------------------|

|       |                                   |
|-------|-----------------------------------|
| ls -F | add one char of */=>@  to entries |
|-------|-----------------------------------|

|       |                                |
|-------|--------------------------------|
| ls -i | list file's inode index number |
|-------|--------------------------------|

|              |                                          |
|--------------|------------------------------------------|
| <u>ls -l</u> | list with long format - show permissions |
|--------------|------------------------------------------|

|               |                                         |
|---------------|-----------------------------------------|
| <u>ls -la</u> | list long format including hidden files |
|---------------|-----------------------------------------|

|        |                                          |
|--------|------------------------------------------|
| ls -lh | list long format with readable file size |
|--------|------------------------------------------|

|               |                                      |
|---------------|--------------------------------------|
| <u>ls -ls</u> | list with long format with file size |
|---------------|--------------------------------------|

|              |                     |
|--------------|---------------------|
| <u>ls -t</u> | sort by time & date |
|--------------|---------------------|

|       |                        |
|-------|------------------------|
| ls -X | sort by extension name |
|-------|------------------------|

- 
- ls -r : list in reverse order
  - ls -R : list recursively directory tree
  - ls -s : list file size
  - ls -S : sort by file size
- 
- **'cd'** – The cd command is one of the most basic and frequently used commands when working on the Linux command line. The cd command, means “change directory” is used to change the current working directory in Linux .
  - **'cd' command options**
  - cd -L : Follow symbolic links. By default, cd behaves as if -L option is specified.

- 
- `d -P` : Don't follow symbolic links. In other words, when this option is specified and you try to navigate to a symlink that points to a directory, `cd` will change into the directory.
  - In its simplest form when used without any argument, `cd` will take you to your home directory.
  - **Absolute and Relative Path Names**
  - When specifying a directory you can use absolute or relative path names. Absolute or full path starts from the system root `/`, and relative path starts from your current directory.
  - When you log into your Linux system your current working directory defaults to your home directory. Assuming that *Downloads* directory exists in your home directory you can navigate to it by using the relative path to the directory:
    - `cd Downloads`

- 
- You can also navigate to the same directory by using its absolute path:
  - `cd /home/username/Downloads`
  - In short, if the path starts with a slash (/) it is the absolute path to the directory.
  - **The Parent Directory**
  - On Unix-like operating systems the current working directory is represented by a single dot (.). Two dots (..), one after the other, are representing the parent directory or in other words the directory immediately above the current one.
  - If you type `cd .` you will change into the current directory or to put it simple the command will do nothing.

- 
- Suppose you are currently in the `/usr/local/share` directory, to switch to the `/usr/local` directory (one level up from the current directory), you would type: `cd ../`
  - To move two levels up to the `/usr` directory (the parent's parent) you could run the following: `cd ../../`
  - Here is another example. Let's say you are in the `/usr/local/share` directory and you want to switch to the `/usr/local/src`. You can do that by typing: `cd ../src`
  - **Navigate to the Previous Directory**
  - To change back to the previous working directory pass the dash (-) character as an argument to the `cd` command: `cd -`



## ➤ **Navigate to the Home Directory**

- To navigate to your home directory simply type `cd`. Another way to return directly to your home directory is to use the tilde (`~`) character, as shown below: `cd ~`
- For example, if you want to navigate to the Downloads directory, which is inside your home directory you would type: `cd ~/Downloads`
- You can also navigate to another user's home directory using the following syntax: `cd ~username`
- **Directories with Space in Their Names**
- If the directory you want to change to has spaces in its name, you should either surround the path with quotes or use the backslash (`\`) character to escape the space:  
`cd 'Dir name with space'` or `cd Dir\ name\ with\ space`

- 
- In Linux systems, you can create new directories either from the command line or with the help of your desktop's file manager. The command that allows you to create directories (also known as folders) is **mkdir**.
  - **Linux mkdir Command Syntax**
  - The syntax for the Linux **mkdir** command is as follows:
  - `mkdir [OPTION] [DIRECTORY]`
  - The command can take one or more directory names as its arguments.
  - To create a directory with the **mkdir** command you need to have write permissions on the parent directory. Otherwise, you will receive a permission denied error.



## ➤ How to Create a New Directory

- To create a directory in Linux pass the name of the directory as the argument to the **mkdir** command. For example, to create a new directory newdir you would run the following command: `mkdir newdir`
- You can verify that the directory was created by listing the contents using the `ls` command: `ls -l`
- Output: `drwxrwxr-x 2 username username 4096 Jan 20 03:39 newdir`
- When providing only the directory name, without the full path, it will be created in the current working directory.
- The current working directory is the directory from which you are running the commands. To change the current working directory, use the `cd` command.

- 
- To create a directory in another location you'll need to provide the absolute or relative file path to the parent directory. For example, to create a new directory in the /tmp directory you would type: `mkdir /tmp/newdir`
  - If you try to create a directory in a parent directory where the user does not have sufficient permissions you will receive Permission denied error: `mkdir /root/newdir`
  - `mkdir: cannot create directory '/root/newdir': Permission denied`
  - The `-v` (`--verbose`) option tells `mkdir` to print a message for each created directory.



## ➤ How to Create Parent Directories

- A parent directory is a directory that is above another directory in the directory tree. To create parent directories use the -p option.
- Let's say you want to create a directory /home/linuxize/Music/Rock/Gothic:
- `mkdir /home/linuxize/Music/Rock/Gothic`
- If any of the parent directories don't exist you will get an error as shown below:
- `mkdir: cannot create directory '/home/linuxize/Music/Rock/Gothic': No such file or directory`
- Instead of creating missing parent directories one by one, use the -p option: `mkdir -p /home/linuxize/Music/Rock/Gothic`

- 
- When mkdir is invoked with the -p option, it will create the directory only if it doesn't exist.
  - If you try to create a directory that already exists and the -p option is not provided, mkdir will print File exists error:  
mkdir newdir
  - mkdir: cannot create directory 'newdir': File exists
  - **How to Set Permissions when Creating a Directory**
  - To create a directory with specified permissions, use the -m (-mode) option. The syntax for assigning permissions is the same as with the chmod command.
  - In the following example, the new directory will have 700 permissions which means that only the user who created the directory will be able to access it:
  - mkdir -m 700 newdir

- 
- If the -m option is not passed to mkdir the newly created directories will have 775 or 755 permissions, depending on the umask value.
  - **How to Create Multiple Directories**
  - To create multiple directories, specify the directories' names as the command arguments, separated by space:
  - `mkdir dir1 dir2 dir3`
  - The mkdir command also allows you to create a complex directory tree with one command:
  - `mkdir -p Music/{Jazz/Blues,Folk,Disco,Rock/{Gothic,Punk,Progressive},Classical/Baroque/Early}`

- 
- The command above will create the following directory tree:

```
Music/  
|-- Classical  
|   |-- Baroque  
|       |-- Early  
|-- Disco  
|-- Folk  
|-- Jazz  
|   |-- Blues  
|-- Rock  
|-- Gothic  
|-- Progressive  
|   |-- Punk
```

- 
- rm is a command-line utility for removing files and directories. It is one of the essential commands that every Linux user/admin should be familiar with.
  - In here we will explain how to use the rm command through examples and explanations of the most common rm options.
  - **How to Use the rm Command**
  - The general syntax for the rm (remove) command is as follows:
    - rm [OPTIONS]... FILE...
    - By default, when executed without any option, rm doesn't remove directories and doesn't prompt the user for whether to proceed with the removal of the given files.

- 
- To delete a single file, use the `rm` command followed by the file name as an argument: `rm filename`
  - If you don't have write permissions on the parent directory, you will get "Operation not permitted" error.
  - If the file is not write protected, it will be removed without notice. On success, the command doesn't produce any output and returns zero.
  - When removing write protected files, the command will prompt you for confirmation, as shown below:
  - `rm: remove write-protected regular empty file 'filename'?`
  - Type `y` and hit Enter to remove the file.
  - The `-f` option tells `rm` never to prompt the user and to ignore nonexistent files and arguments.
  - `rm -f filename`

- 
- If you want to get information about what is being removed, use the -v (verbose) option: `rm -v filename`
  - Output: removed 'filename'
  - **Removing Multiple Files**
  - `rm` allows you to delete multiple files at once. To do that, pass the filenames as arguments separated by space:
  - `rm filename1 filename2 filename3`
  - You can use regular expressions to match multiple files. For example, to remove all .png files in the current directory, you would type: `rm *.png`
  - When using regular expressions, before running the `rm` command, it is always a good idea to list the files with the `ls` command so that you can see which files will be deleted.



## ➤ Removing Directories (Folders)

- To remove one or more **empty directories** use the -d option: `rm -d dirname` or simply use the command **rmdir**
- `rm -d` is functionally identical to the **rmdir** command.
- To remove non-empty directories and all the files within them recursively, use the -r (recursive) option:
- `rm -r dirname`

## ➤ Prompt Before Removal

- The -i option tells `rm` to prompt the user for each given file before removing it: `rm -i filename1 filename2`
- To confirm type `y` and press Enter:
- `rm: remove regular empty file 'filename1'?`
- `rm: remove regular empty file 'filename2'?`

- 
- When removing more than three files or recursively removing a directory, to get a single prompt for the entire operation, use the `-I` option:
  - `rm -i filename1 filename2 filename3 filename4`
  - You will be asked to confirm the removal of all given files and directories: `=> rm: remove 4 arguments?`
  - `rm -rf`
  - If the given directory or a file within the directory is write-protected, the `rm` command will prompt you to confirm the operation. To remove a directory without being prompted, use the `-f` option: `rm -rf dirname`
  - The `rm -rf` command is very dangerous and should be used with extreme caution!

- 
- The mv command (short from move) is used to rename or move files and directories from one location to another.
  - **How to Use the mv Command**
  - mv [OPTIONS] source destination
  - The source can be one or more files or directories and destination can be a single file or directory.
  - If you specify multiple files or directories as a source, the destination must be a directory. In this case, the source files are moved to the target directory.
  - If you specify a single file as source, and the destination target is an existing directory then the file is moved to the specified directory.

- 
- If you specify a single file as source, and a single file as destination target then you're renaming the file.
  - If the source is a directory and destination doesn't exist source will be renamed to destination, otherwise it be moved inside the destination directory.
  - To move a file or directory you need to have write permissions on both source and destination. Otherwise, you will receive a permission denied error.
  - For example, to move the file file1 from the current working directory to the /tmp directory you would run:  
`$ mv file1 /tmp`
  - To rename a file specify the destination file name:  
`$ mv file1 file2`

- 
- The command for moving directories is the same as when moving files. In the following example if the dir2 directory exists the command will move dir1 inside dir2. If dir2 doesn't exist, dir1 will be renamed to dir2: `$ mv dir1 dir2`
  - **Moving Multiple Files and Directories**
  - To move multiple files and directories, specify the files you want to move as the source. For example to move the files file1 and file2 to the dir1 directory you would type:  
`$ mv file1 file2 dir1`
  - The mv command also allows you to use pattern matching. For example, to move all pdf files from the current directory to the ~/Documents directory you would use: `$ mv *.pdf ~/Documents`



## ➤ **mv Command Options**

- The mv command accepts options that affect default command behavior. In some Linux distributions mv may be an alias to the mv command with a custom set of options. For example in CentOS mv is an alias to mv -i. You can find whether mv is an alias using the type command: `$ type mv`
- If mv is alias the output will look something like this:
- Output: mv is aliased to `mv -i`
- Note: If you specify conflicting options, the option specified last takes precedence.
- **Prompt before overwriting**
- By default if the destination file exists it will be overwritten. To prompt for confirmation use the -i option:

- 
- `mv -i file1 /tmp`
  - Output: `mv: overwrite '/tmp/file1'?`
  - To overwrite the file type the character `y` or `Y`.
  - **Force overwriting**
  - If you try to overwrite a read only file the `mv` command will prompt you whether you want to overwrite the file:
  - `mv -i file1 /tmp`
  - `mv: replace '/tmp/file1', overriding mode 0400 (r-----)?`
  - To avoid being prompted use the `-f` options:
  - `mv -f file1 /tmp`
  - This option is especially useful when you need to overwrite multiple read only file.



- **Do not overwrite existing files**

- The -n option tells mv never to overwrite any existing file:

- `$ mv -n file1 /tmp`

- If a file1 exists the command above will do nothing, otherwise it will move the file to the /tmp directory.

- **Backing up files**

- If the destination file exists you can create a backup of it using the -b option: `$ mv -b file1 /tmp`

- The backup file will have the same name as the original file with a tilde (~) appended to it.

- Use the ls command to verify that the backup was created:

- `$ ls /tmp/file1*`

- Output: `/tmp/file1 /tmp/file1~`



- **Verbose output**

- Another option that can be useful is -v. When using this option the command will print the name of each moved file:

- `$ mv -i file1 /tmp`

- Output: renamed 'file1' -> '/tmp/file1'

- **'cat' Command**

- The cat command is one of the most widely used commands in Linux and its name comes from its functionality to concatenate files. It can read and concatenate files, writing their contents to the standard output. If no file is specified or if the input file name is specified as a single hyphen (-) it reads from the standard input.

- 
- Cat is most commonly used to display the contents of one or multiple text files, combine files by appending the contents of one file to the end of another file, and create new files.
  - **Cat Command Syntax**
  - The cat utility expressions take the following form:
  - `cat [OPTIONS] [FILE_NAMES]`
  - **OPTIONS** - cat options. Use `cat --help` to view all available options.
  - **FILE\_NAMES** - Zero or more file names.



## ➤ **Displaying File Contents**

- The most basic and common usage of the cat command is to read the contents of files.

- For example, the following command will display the contents of the /etc/issue file in the terminal: `$ cat /etc/issue`

## ➤ **Redirect Contents of File**

- Instead of displaying the output to stdout (on the screen) you can redirect it to a file.
- The following command will copy the contents of file1.txt to file2.txt using the (>) operator : `$ cat file1.txt > file2.txt`

- 
- If the file2.txt file doesn't exist, the command will create it. Otherwise, it will overwrite the file.
  - Use the (>>) operator to append the contents of file1.txt to file2.txt : `$ cat file1.txt >> file2.txt`
  - Same as before, if the file is not present, it will be created.
  - **Print Line Numbers**
  - To display contents of a file with line numbers, use the -n option: `$ cat -n /etc/lsb-release`
  - 1     `DISTRIB_ID=Ubuntu`
  - 2     `DISTRIB_RELEASE=18.04`
  - 3     `DISTRIB_CODENAME=bionic`
  - 4     `DISTRIB_DESCRIPTION="Ubuntu 18.04.1 LTS"`



- **Suppress Repeated Empty Lines**

- Use the -s option to omit the repeated empty output lines:

- `$ cat -s file.txt`

- **Display TAB characters**

- The -T option allows you to visually distinguish between tabs and spaces. `$ cat -T /etc/hosts`

- Output

- `127.0.0.1^Ilocalhost`

- `127.0.1.1^Iubuntu1804.localdomain`

- The TAB characters will be displayed as ^I.



## ➤ **Display End of Lines**

➤ To display the invisible line ending character use the -e argument: `$ cat -e /etc/lsb-release`

➤ Output:

➤ `DISTRIB_ID=Ubuntu$`

➤ `DISTRIB_RELEASE=18.04$`

➤ `DISTRIB_CODENAME=bionic$`

➤ `DISTRIB_DESCRIPTION="Ubuntu 18.04.1 LTS"$`

➤ The Line endings will be displayed as \$.



## ➤ Concatenating Files

- When passing two or more file names as arguments to the cat command the contents of the files will be concatenated. cat reads the files in the sequence given in its arguments and displays the file's contents in the same sequence.
- For example, the following command will read the contents of file1.txt and file2.txt and display the result in the terminal: `$ cat file1.txt file2.txt`
- You can concatenate two or more text files and write them to a file.
- The following command will concatenate the contents of file1.txt and file2.txt and write them to a new file combinedfile.txt using the (>) operator :
- `$ cat file1.txt file2.txt > combinedfile.txt`

- 
- If the combinedfile.txt file doesn't exist, the command will create it. Otherwise, it will overwrite the file.
  - To concatenate the contents of file1.txt and file2.txt and append the result to file3.txt to use the (>>) operator: `$ cat file1.txt file2.txt >> file3.txt`
  - If the file is not present, it will be created.
  - When concatenating files with cat, you can use the same arguments as shown in the previous section.
  - **Creating Files**
  - Creating small files with cat it often easier than opening a text editor such as nano, Vim, Sublime Text or Visual Studio Code.

- 
- To create a new file, use the cat command followed by the redirection operator (>) and the name of the file you want to create. Press Enter, type the text and once you are done, press the CTRL+D to save the file.
  - In the following example, we are creating a new file named file1.txt: `$ cat > file1.txt`
  - If a file named file1.txt is present, it will be overwritten. Use the '>>' operator to append the output to an existing file.
  - `cat >> file1.txt`



## ➤ **'cp' Command**

- Copying files and directories is one of the most common tasks you'll perform on a daily basis.
- cp is a command-line utility for copying files and directories on Unix and Linux systems. In this article, we will explain how to use the cp command.
- **How to Use cp command**
- The general syntax for the cp command is as follows:
- `cp [OPTIONS] SOURCE... DESTINATION`
- The SOURCE can have one or more files or directories as arguments and the DESTINATION can have a single file or directory as an argument.

- 
- When the SOURCE and DESTINATION arguments are both files the cp command copies the first file to the second one. If the file doesn't exist the command will create it.
  - When the SOURCE has multiple files or directories as arguments, the DESTINATION argument must be a directory. In this case, the SOURCE files and directories are moved to the DESTINATION directory.
  - When the SOURCE and DESTINATION arguments are both directories the cp command will copy the first directory into the second one.
  - To copy files and directories you must have at least read permissions on the source file and write permission on the destination directory. Otherwise, you will get a permission denied error.

- 
- Copying Files with cp Command
  - The most basic scenario of using cp is copying a file in the current working directory. For example, to copy a file named file.txt to file\_backup.txt you would run the following command:
    - `$ cp file file_backup` or `$ cp file{,}_backup}`
    - To copy a file to another directory specify the absolute or the relative path to the destination directory. When only the directory name is specified as a destination the copied file will have the same name as the original file.
    - In the following example we are copying the file file.txt to the /backup directory:
      - `$ cp file.txt /backup`

- 
- If you want to copy the file under a different name you need to specify the desired file name. The command below will copy the file to the specified directory as new\_file.txt.
  - `cp file.txt /backup/new_file.txt`
  - By default, if the destination file exists it will be overwritten. The `-n` option tells `cp` to not overwrite an existing file. To prompt for confirmation use the `-i` option.
  - `cp -i file.txt file_backup.txt`
  - If you want to copy the file only if it is newer than the destination use the `-u` option:
  - `cp -u file.txt file_backup.txt`

- 
- When copying a file, the new file will be owned by the user running the command. Use the -p option to preserve the file mode, ownership and timestamps: `$ cp -p file.txt file_backup.txt`
  - Another option that can be useful is -v, which tells cp to print verbose output: `$ cp -v file.txt file_backup.txt`
  - Output: 'file.txt' -> 'file\_backup.txt'
  - **Copying Directories with cp Command**
  - To copy a directory, including all its files and subdirectories, use the -R or -r option. In the following example we are copying the directory Pictures to Pictures\_backup:
  - `cp -R Pictures Pictures_backup`

- 
- The command above will create a destination directory and recursively copy all files and subdirectories from the source to the destination directory.
  - If the destination directory already exists the source directory itself and all its content will be copied to the destination directory. To copy only the files and subdirectories but not the target directories use the -T option: `$ cp -RT Pictures Pictures_backup`
  - Another way to copy only the directory's content but not the directory itself is to use the wildcard character (\*). The disadvantage of the following command is that it will not copy the hidden files and directories (the ones starting with a dot .): `$ cp -RT Pictures/* Pictures_backup/`

- 
- All the options we used in the previous section when copying files can be also used when copying directories. The main difference is that when copying directories you always need to use the -R option.
  - **Copy Multiple Files and Directories**
  - To copy multiple files and directories at once specifying their names and use the destination directory as the last argument: `$ cp file.txt dir file1.txt dir1`
  - Note: When copying multiple files the destination must be a directory.



## ➤ **'echo' Command**

- The echo command is one of the most basic and frequently used commands in Linux. The arguments passed to echo are printed to the standard output.
- echo is commonly used in shell scripts to display a message or output the results of other commands.
- echo is a shell builtin in Bash and most of the other popular shells like Zsh and Ksh. Its behavior is slightly different from shell to shell.
- There is also a standalone `/usr/bin/echo` utility, but usually, the shell built-in version will take precedence. We will cover the Bash builtin version of echo.

- 
- The syntax for the echo command is as follows:
  - `echo [-neE] [ARGUMENTS]`
  - When the -n option is used, trailing newline is suppressed.
  - If the -e option is given, the following backslash-escaped characters will be interpreted:
    - `\\` - Displays a backslash character.
    - `\a` - Alert (BEL)
    - `\b` - Displays a backspace character.
    - `\c` - Suppress any further output
    - `\e` - Displays an escape character.
    - `\f` - Displays a form feed character.
    - `\n` - Displays a new line.
    - `\r` - Displays a carriage return.
    - `\t` - Displays a horizontal tab.
    - `\v` - Displays a vertical tab.

- 
- The -E option disables the interpretation of the escape characters. This is the default.
  - There are a few points to consider when using the echo command.
  - The shell will substitute all variables, wildcard matching, and special characters before passing the arguments to the echo command.
  - Although not necessary, it is a good programming practice to enclose the arguments passed to echo in double or single quotes.
  - When using single quotes " the literal value of each character enclosed within the quotes will be preserved. Variables and commands will not be expanded.



## ➤ **echo Examples**

- The examples below show how to use the echo command:
- Display a line of text on standard output.
- `echo Hello, World!`, Output: Hello, World!
- **Display a line of text containing a double quote.**
- To print a double quote, enclose it within single quotes or escape it with the backslash character.
- `$ echo 'Hello "Linuxize"'` or `$ echo "Hello \"Linuxize\""`
- Output: Hello "Linuxize"
- **Display a line of text containing a single quote.**
- To print a single quote, enclose it within double quotes or use the ANSI-C Quoting.

- 
- `$ echo "I'm a Linux user." or $ echo $'I\'m a Linux user.'`
  - Output: I'm a Linux user.
  - **Display a message containing special characters.**
  - Use the `-e` option to enable the interpretation of the escape characters.
  - `$ echo -e "You know nothing, Jon Snow.\n\t- Ygritte"`
  - Output: You know nothing, Jon Snow.  
- Ygritte
  - **Pattern matching characters.**
  - The `echo` command can be used with pattern matching characters, such as the wildcard characters. For example, the command below will return the names of all the `.php` files in the current directory.

- 
- `$ echo The PHP files are: *.php`
  - Output:
  - The PHP files are: index.php contact.php functions.php
  - **Redirect to a file**
  - Instead of displaying the output to the screen, you can redirect it to a file using the `>`, `>>` operators.
  - `echo -e 'The only true wisdom is in knowing you know nothing.\nSocrates' >> /tmp/file.txt`
  - If the file.txt doesn't exist, the command will create it. When using `>` the file will be overwritten, while the `>>` will append the output to the file.

- 
- Use the cat command to view the content of the file:
  - `$ cat /tmp/file.txt`
  - Output:
  - The only true wisdom is in knowing you know nothing.  
Socrates
  - **Displaying variables**
  - echo can also display variables. In the following example, we'll print the name of the currently logged in user:
  - `$ echo $USER` Output: `< currently logged in user>`
  - `$USER` is a shell variable that holds your username.



- **Displaying output of a command**

- Use the `$(command)` expression to include the command output in the echo's argument. The following command will display the current date:

- `$ echo "The date is: $(date +%D)"`

- Output: The date is: 04/17/19

- **'clear' Command**

- clear is a standard Linux command to clear the terminal screen. This command will ignore any command-line parameters that may be present. Also, the clear command doesn't take any argument and it is almost similar to cls command on a number of other operating systems.

- Syntax: `$ clear`



## ➤ **'exit' Command**

- exit command in linux is used to exit the shell where it is currently running. It takes one more parameter as [N] and exits the shell with a return of status N. If n is not provided, then it simply returns the status of last command that is executed.
- Syntax: \$ exit [n] **Note** n is optional



## ➤ **'wc' Command**

- The 'wc' command or 'word count' command allows you to count the number of lines, words, characters, and bytes of each given file or standard input and print the result.
- How to Use the wc Command
- The syntax for the wc command is as follows:
- `wc OPTION... [FILE]...`
- The wc command can accept zero or more input FILE names. If no FILE is specified, or when FILE is -, wc will read the standard input. A word is a string of characters delimited by a space, tab, or newline.

- 
- When the 'wc' command is used without any options, the wc command will print four columns, the number of lines, words, byte counts and the name of the file for each file passed as an argument.
  - When using the standard input the fourth column (filename) is not displayed.
  - Example, the following command will display information about the virtual file /proc/cpuinfo: `$ wc /proc/cpuinfo`
  - The output will look something like the following:
  - 448 3632 22226 /proc/cpuinfo, meaning
  - 448 is the number of lines.
  - 3632 is the number of words.
  - 22226 is the number of characters.

- 
- When using the standard input, the file name is not shown:
  - `$ wc < /proc/cpuinfo`
  - **Note:** `<` means redirect file to standard input, similar to typing the file in standard input.
  - Output is:
  - `448 3632 22226`
  - To display information about more than one file, pass the filenames, as arguments, separated by space:
  - `$ wc /proc/cpuinfo /proc/meminfo`
  - The command will give you information about each file and a line including total statistics:

- 
- Output:
  - 448 3632 22226 /proc/cpuinfo
  - 49 143 1363 /proc/meminfo
  - 497 3775 23589 total
  - The options below allow you to select which counts are printed.
  - -l, --lines - Print the number of lines.
  - -w, --words - Print the number of words.
  - -m, --chars - Print the number of characters.
  - -c, --bytes - Print the number of bytes.
  - -L, --max-line-length - Print the length of the longest line.

- 
- When using multiple options counts are printed in the following order: newline, words, characters, bytes, maximum line length.
  - For example, to display only the number of words you would use: `$ wc -w /proc/cpuinfo`
  - Output:
  - `3632 /proc/cpuinfo`
  - Here is another example that will print the number of lines and the length of the longest line. `$ wc -lL /proc/cpuinfo`
  - Output:
  - `448 792 /proc/cpuinfo`



## ➤ **Count the Number of Lines**

- The `wc` command is mostly used with the `-l` option to count only the number of lines in a text file. For example, to count the number of lines in the `/etc/passwd` file you would type:

- `$ wc -l /etc/passwd`

- The first column is the number of lines and the second one is the name of the file: `44 /etc/passwd`

## ➤ **Count the Number of Words**

- To count only the number of words in a text file use `wc -w` followed by the file name. The following example counts the number of words in the `~/Documents/file.txt` file:

- The number of words is shown in the first column:

- `513 /home/linuxize/Documents/file.txt`



- **Counting Files in the Current Directory**

- The find command passes a list of all files in the current directory with each file name on a single line to the 'wc' command, which counts the number of lines and prints the result: `$ find . -type f | wc -l`

- **Count the number of users**

- In the example below wc is used to count the number of lines from the output of the getent command .
- `$ getent passwd | wc -l`

- 
- 'getent' Linux command helps the user to get the entries in a number of important text files called databases. This includes the passwd and the group of databases which stores the user information.
  - 'getent' is a common way to look up user details on Linux. Since getent uses the same name of service as the system, getent will show all information, including that gained from the network information sources such as LDAP.
  - The databases usually searched: ahosts, ahostsv4, ahostsv6, aliases, ethers (Ethernet addresses), group, gshadow, hosts, netgroup, networks, passwd, protocols, rpc, services, and shadow.

- 
- Example 1: Fetch the list of user accounts on a Linux system stored in the 'passwd' database. This will show all the user accounts, regardless of the type of name service being used.
  - Syntax: `$ getent database [key ...]`
  - **'grep' Command**
  - The 'grep' command which stands for "global regular expression print" is one of the most powerful and commonly used commands in Linux.
  - 'grep' searches one or more input files for lines that match a given pattern and writes each matching line to standard output. If no files are specified, 'grep' reads from the standard input which is usually the output of another command.



## ➤ **Grep Command Syntax**

- The 'grep' command expressions take the following form:
- `$ grep [OPTIONS] PATTERN [FILE...]`
- The items in square brackets are optional.
- **OPTIONS** - Zero or more options. 'grep' provides a number of options that control its behavior.
- **PATTERN** - Search pattern.
- **FILE** - Zero or more input file names.



## ➤ **How to use Grep to Search for a String in Files**

- The most basic usage of the 'grep' command is to search for a string (text) in a file.
- Example: display the lines from the /etc/passwd file containing the string bash you use the command:
  - `$ grep bash /etc/passwd`
  - Output:
    - `root:x:0:0:root:/root:/bin/bash`
    - `cwmonie:x:1000:1000:cwmonie:/home/cwmonie:/bin/bash`
  - Note: If the string includes spaces, enclose it in single or double quotation marks as follows:
    - `$ grep "Gnome Display Manager" /etc/passwd`



## ➤ **Grep Invert Match (Exclude)**

- To display the lines that do not match a pattern, use the -v ( or --invert-match) option.
- Example: display the lines from the /etc/passwd file that do not contain the string nologin:
- `$ grep -v nologin /etc/passwd`
- Output:
- `root:x:0:0:root:/root:/bin/bash`
- `colord:x:124:124::/var/lib/colord:/bin/false`
- `git:x:994:994:git daemon user:/usr/bin/git-shell`
- `cwmonie:x:1000:1000:linuxize:/home/cwmonie:/bin/bash`



## ➤ Use grep to Search for a String in Command Output

- Instead of specifying input files you can **pipe** the output of another command to 'grep', and then display only lines matching a given pattern.
- For example, to find out the processes that contain bash use the command below:
- `$ ps -ef | grep color:` will display all the process lines that include color, including the grep process.
- You can also chain multiple pipes in one command. As you can see in the output above there is also a line containing the grep process. If you don't want that line to be shown pass the output to another grep instance as shown below.
- `$ ps -ef | grep bash | grep -v grep`



## ➤ **Grep Recursive Search**

- To recursively search for a pattern, use the -r option (or --recursive). This will search through all files in the specified directory, skipping the symlinks ( symbolic links ) that are encountered recursively. To follow all symbolic links, use the -R option (or --dereference-recursive).
- In the following example we are searching for the string localhost in all files inside the our home directory:
- `grep -r localhost ~`
- The command will print the matching lines prefixed by the full path to the file.

- 
- If instead of -r you use -R option grep will follow all symbolic links:
  - `$ grep -R linuxize.com /etc`
  - **Grep Show Only Filename**
  - To print only the names of files containing the matched pattern you can use the -l ( or --files-with-matches) option.
  - For example to search through all files ending with .conf in the [current working directory]: `$ grep -l localhost *.conf`
  - **Note:** The -l option is usually used in combination with the recursive option -R:
  - `grep -Rl localhost /tmp`



## ➤ **Grep Case Insensitive**

- By default the grep command is case sensitive which means that uppercase and lowercase characters are treated as distinct.
- To ignore case when searching, use the -i option (or --ignore-case).
- For example, when searching for Zebra without any option the following command will not show any output i.e there are no matching lines.
- `$ grep Zebra /usr/share/words`
- But if you perform a case insensitive search using the -i option, it will match both upper and lower case letters:
- `grep -i Zebra /usr/share/words`

- 
- Specifying "Zebra" will match "zebra", "ZEbrA" or any other combination of upper and lower case letters for that string, like zebra, zebra's, zebras etc
  - **Grep Show Line Numbers**
  - To show the number of the lines which contains a string that matches a pattern, use the -n ( or --line-number) option. When using this option grep will print the matches to standard output prefixed with the line number it was found on.
  - For example to display the lines from the /etc/services file containing the string bash prefixed with the matching line number you can use the following command:
  - `grep -n 10000 /etc/services`

- 
- The output below shows us that the matches are found on lines 10423 and 10424.

- 10423:ndmp                    10000/tcp

- 10424:ndmp                    10000/udp

- **Grep Count Matches**

- To print a count of matching lines to standard output, use the -c ( or --count) option.

- In the example below, we are counting the number of accounts that have /usr/bin/zsh as a shell.

- `$ grep -c '/usr/bin/zsh' /etc/passwd`

- Output: 4



## ➤ **Grep Multiple Strings (Patterns)**

- Two or more search patterns can be joined by the OR operator |.
- Grep interprets the pattern as a basic regular expression where the meta-characters like | lose their special meaning and the back slashed versions must be used.
- In the example below we are searching all occurrences of the words fatal, error and critical in the Nginx log error file:
- `grep 'fatal\|error\|critical' /var/log/nginx/error.log`
- If you use the extended regular expression option -E (or --extended-regexp) then the operator | should not be escaped, as shown below:
- `$ grep -E 'fatal|error|critical' /var/log/nginx/error.log`



## ➤ **'head' Command**

- The head command prints the first lines (10 by default) of one or more files or piped data to standard output.
- The syntax for the head command is as follows:
- head [OPTION]... [FILE]...
- OPTION - head options, we'll go over the most common options.
- FILE - Zero or more input file names. If no FILE is given, or when FILE is -, head will read the standard input.
- How to Use the Head Command
- The simplest form is when used without any option, the head command will display the first 10 lines.
- `$ head filename.txt`



## ➤ **How to Display a Specific Number of Lines**

- Use the -n (--lines) option followed by an integer specifying the number of lines to be shown:
- `$ head -n <NUMBER> filename.txt`
- You can omit the letter n and use just the hyphen (-) and the number (with no space between them).
- To display the first 30 lines of a file named filename.txt you would type:
- `$ head -n 30 filename.txt`
- The following will produce the same result as the above commands:
- `$ head -30 filename.txt`



## ➤ How to Display a Specific Number of Bytes

- The `-c` (`--bytes`) option allows to print a specific number of bytes: `$ head -c <NUMBER> filename.txt`
- For example to display the first 100 bytes of data from the file named `filename.txt` you would type:
  - `$ head -c 100 filename.txt`
- You can use a multiplier suffix after the number to specify the number of bytes to show. `b` multiplies by 512, `KB` multiplies by 1000, `K` multiplies by 1024, `MB` multiplies by 1000000 ( $1000 * 1000$ ), `M` multiplies by 1048576 ( $1024 * 1024$ ), and so on for `GB`, `G` and also for `T`, `P`, `E`, `Z`, `Y`.
- The following command will display the first five kilobytes (5120) of the file `filename.txt`:
  - `$ head -c 5K filename.txt`



## ➤ **How to Display Multiple Files**

- If multiple files are provided as input to the head command, it will display the first ten lines from each provided file.
- `$ head filename1.txt filename2.txt`
- You can use the same options as when displaying one file.
- This example shows the first 20 lines of the files filename1.txt and filename2.txt:
- `head -n 20 filename1.txt filename2.txt`
- When more than one file is used, the output precede each with a header showing the file name.



## ➤ **How to Use Head with Other Commands**

- The head command can be used in combination with other commands by redirecting the standard output from/to other commands ( utilities ) using pipes ( | ).
- The following command will hash the \$RANDOM environment variable, display the first 32 bytes and display 24 characters random string:
- `echo $RANDOM | sha512sum | head -c 24 ; echo`



## ➤ **'tail' Command**

- The tail command displays the bottom part (10 lines by default) of one or more files or piped data. It can be also used to monitor the file changes in real time.
- One of the most common uses of the tail command is to watch and analyze logs and other files that change over time, usually combined with other tools like grep.
- The syntax for the tail command is as follows:
- `tail [OPTION]... [FILE]...`
- **OPTION** - tail options, we'll go over the most common options.
- **FILE** - Zero or more input file names. If no FILE is given, or when FILE is -, tail will read the standard input.



- How to Use the Tail Command

- The simplest form is when used without any option, the tail command will display the last 10 lines.

- `$ tail filename.txt`

- **How to Display a Specific Number of Lines**

- Use the `-n` (`--lines`) option followed by an integer specifying the number of lines to be shown:

- `$ tail -n <NUMBER> filename.txt`

- You can omit the letter `n` and use just the hyphen (`-`) and the number (with no space between them).

- To display the first 30 lines of a file named `filename.txt` you would type:

- `$ tail -n 30 filename.txt`

- 
- The following will produce the same result as the above commands:

- `$ tail -30 filename.txt`

- **How to Display a Specific Number of Bytes**

- The `-c` (`--bytes`) option allows to print a specific number of bytes: `$ tail -c <NUMBER> filename.txt`

- For example to display the last 100 bytes of data from the file named `filename.txt` you would type:

- `$ tail -c 100 filename.txt`

- You can use a multiplier suffix after the number to specify the number of bytes to show. `b` multiplies by 512, `KB` multiplies by 1000, `K` multiplies by 1024, `MB` multiplies by 1000000 ( $1000 * 1000$ ), `M` multiplies by 1048576 ( $1024 * 1024$ ), and so on for `GB`, `G` and also for `T`, `P`, `E`, `Z`, `Y`.

- 
- The following command will display the last five kilobytes (5120) of the file filename.txt:
  - `$ tail -c 5K filename.txt`
  - **How to Watch a File for Changes**
  - To monitor a file for changes use the -f (--follow) option:
  - `tail -f filename.txt`
  - This option is particularly useful for monitoring log files. For example, to display the last 10 lines of the `/var/log/nginx/error.log` file, and monitor the file for updates you would use:
  - `tail -f /var/log/nginx/error.log`

- 
- To interrupt the tail command while it is watching a file, press Ctrl+C.
  - To keep monitoring the file when it is recreated, use the -F option.
  - `tail -F filename.txt`
  - This option is useful in situations when the tail command is following a log file that rotates. When used with -F option the tail command will reopen the file the as soon as it became available again.



## ➤ **How to Display Multiple Files**

- If multiple files are provided as input to the head command, it will display the last ten lines from each provided file.
- `$ tail filename1.txt filename2.txt`
- You can use the same options as when displaying one file.
- This example shows the last 20 lines of the files filename1.txt and filename2.txt:
- `tail -n 20 filename1.txt filename2.txt`
- When more than one file is used, the output precede each with a header showing the file name.



## ➤ **How to Use Tail with Other Commands**

- The tail command can be used in combination with other commands by redirecting the standard output from/to other utilities using pipes.
- For example to monitor the apache access log file and only display those lines that contain the IP address 192.168.42.12 you would use:
  - `tail -f /var/log/apache2/access.log | grep 192.168.42.12`
- The following ps command will display the top ten running processes sorted by CPU usage:
  - `ps aux | sort -nk +3 | tail -5`



## ■ **Commands variants**

- There are 3 commands, egrep, fgrep and tree, that are similar to some commands we have already seen. Research the 3 commands so you are aware of them because there may be situations where they may be better than their variants