**Linux History and Operation**

* **The Evolution of Linux**

In 1991, Linus Torvalds a student at the university of Helsinki, Finland, thought to have a freely available academic version of Unix started writing its own code. Later this project became the Linux kernel. He wrote this program specially for his own PC . He did it on MINIX using GNU C compiler. GNU C compiler is still the main choice to compile Linux code but other compilers are also used like Intel C compiler.

He started it just for fun but ended up with such a large project. Firstly he wanted to name it as 'Freax' but later it became 'Linux'.

He published the Linux kernel under his own license and was restricted to us as commercially. Linux uses most of its tools from GNU software and are under GNU copyright. In 1992, he released the kernel under GNU General Public License.

* **The GNU Movement and the GPL**

Linus Torvalds has given linux kernel license to GNU General Public License (GPL) version 2. GNU make sure that any software source code licensed under it have to make originating source code open and freely availble to all its users. Here, freely doesn't mean by cost but it means that it is freely available to users to distribute and modify the code.

There is the third version of GNU, GNU Lesser General Public License (LGPL) version 3. But it imposes some more permissions on the license. Torvalds doesn't like some provisions in version 3 and have announced that linux kernel will not come under version 3.

* **Linux Operations as a Server**
* **The Architecture and Structure of Linux**

1. **Kernel:** 1) it is core part of linux os.

2) responsible for all major activites.

3) it prevents conflicts between different processes.

1. **System Library:** by using system libraries application programs access kernel features.
2. **Shell:**It is an interface to the kernel which hides the complexity of the kernel’s functions from the users. It takes commands from the user and executes the kernel’s functions.
3. **Hardware Layer:**This layer consists all peripheral devices like RAM/ HDD/ CPU etc.
4. **System Utility:**these are responsible for performing specialised indivisual level task.

**Basic Commands**

**ls =** to listthe content of directory.

|  |  |
| --- | --- |
| **ls option** | **Description** |
| [ls -a](https://www.javatpoint.com/linux-ls#linux-ls-a) | In Linux, hidden files start with . (dot) symbol and they are not visible in the regular directory. The (ls -a) command will enlist the whole list of the current directory including the hidden files. |
| [ls -l](https://www.javatpoint.com/linux-ls#linux-ls-l) | It will show the list in a long list format. |
| ls -lh | This command will show you the file sizes in human readable format. Size of the file is very difficult to read when displayed in terms of byte. The (ls -lh) command will give you the data in terms of Mb, Gb, Tb, etc. |
| ls -lhS | If you want to display your files in descending order (highest at the top) according to their size, \\then you can use (ls -lhS) command. |
| [ls -l - -block-size=[SIZE]](https://www.javatpoint.com/linux-ls#linux-ls-l-block-size) | It is used to display the files in a specific size format. Here, in [SIZE] you can assign size according to your requirement. |
| [ls -d \*/](https://www.javatpoint.com/linux-ls#linux-ls-d-asterisk-slash) | It is used to display only subdirectories. |
| [ls -g or ls -lG](https://www.javatpoint.com/linux-ls#linux-ls-g) | With this you can exclude column of group information and owner. |
| ls -n | It is used to print group ID and owner ID instead of their names. |
| [ls --color=[VALUE]](https://www.javatpoint.com/linux-ls#linux-ls-color) | This command is used to print list as colored or discolored. |
| ls -li | This command prints the index number if file is in the first column. |
| `ls -p | It is used to identify the directory easily by marking the directories with a slash (/) line sign. |
| ls -r | It is used to print the list in reverse order. |
| ls -R | It will display the content of the sub-directories also. |
| ls -lX | It will group the files with same extensions together in the list. |
| ls -lt | It will sort the list by displaying recently modified filed at top. |
| [ls ~](https://www.javatpoint.com/linux-ls#linux-ls-tilde) | It gives the contents of home directory. |
| [ls ../](https://www.javatpoint.com/linux-ls#linux-ls-dot-dot-slash) | It give the contents of parent directory. |
| ls --version | It checks the version of ls command. |

**Cp** = 'cp' means copy. 'cp' command is used to copy a file or a directory.

|  |  |
| --- | --- |
| **Option** | **Function** |
| [cp -r](https://www.javatpoint.com/linux-cp-r) | To copy a directory along with its sub dirctories. |
| [cp file1 file 2 directory name](https://www.javatpoint.com/linux-copy-multiple-files) | To copy multiple file or directories in a directory. |
| [cp -backup](https://www.javatpoint.com/linux-cp-backup) | To backup the existing file before over writing it. |
| [cp -i](https://www.javatpoint.com/linux-cp-i) | Asks for confirmtion. |
| [cp -l](https://www.javatpoint.com/linux-cp-l) | To create hard link file. |
| [cp -p](https://www.javatpoint.com/linux-cp-p) | Preserves attribute of a file. |
| [cp -u -v](https://www.javatpoint.com/linux-cp-u-v) | To make sure source file is newer then destination file. |

**mv =** Linux mv command is used to move existing file or directory from one location to another. It is also used to rename a file or directory. If you want to rename a single directory or file then **'mv'** option will be better to use.

|  |  |
| --- | --- |
| **Option** | **Function** |
| [mv -i](https://www.javatpoint.com/linux-mv-i) | Asks for permission to over write. |
| [mv \*](https://www.javatpoint.com/linux-mv-asterisk) | Move multiple files to a specific directory. |
| [mv --suffix](https://www.javatpoint.com/linux-mv-suffix) | Used to take backup before over writing. |
| [mv -u](https://www.javatpoint.com/linux-mv-u) | Only move those files that doesn't exist. |

**sort=** The 'sort' command sorts the file content in an alphabetical order.

**Syntax:**

1. sort **<fileName>**

**grep =** The 'grep' command stands for **"global regular expression print"**. grep command filters the content of a file which makes our search easy.

**Syntax:**

1. command | grep **<searchWord>**
2. **grep -vM**: The 'grep -v' command displays lines not matching to the specified word.

**Syntax:**

* 1. grep -v **<searchWord>** **<fileName>**

1. **grep -i**: The 'grep -i' command filters output in a case-insensitive way.

**Syntax:**

* 1. grep -i **<searchWord>** **<fileName>**

1. **grep -A/ grep -B/ grep -C**
2. grep -A command is used to display the **line after the result**.
3. grep -B command is used to display the **line before the result**.
4. grep -C command is used to display the **line after and line before** the result.
5. grep -A**<lineNumber>** **<searchWord>** **<fileName>**
6. grep -B**<lineNumber>** **<searchWord>** **<fileName>**
7. grep -C**<lineNumber>** **<searchWord>** **<fileName>**

**cat=**

|  |  |
| --- | --- |
| **Option** | **Function** |
| [cat > fileName](https://www.javatpoint.com/linux-cat#linux-cat-create) | To create a file. |
| [cat oldfile > newfile](https://www.javatpoint.com/linux-cat#linux-cat-copy) | To copy content from older to new file. |
| [cat file1 file2 and so on > new file name](https://www.javatpoint.com/linux-cat#linux-cat-concatenate) | To concatenate contents of multiple files into one. |
| [cat -n/cat -b fileName](https://www.javatpoint.com/linux-cat#linux-cat-display-line-numbers) | To display line numbers. |
| [cat -e fileName](https://www.javatpoint.com/linux-cat#linux-cat-e) | To display $ character at the end of each line. |
| [cat [fileName] <<EOF](https://www.javatpoint.com/linux-cat#linux-cat-end-marker) | Used as page end marker. |

**Head=** The 'head' command displays the starting content of a file. By default, it displays starting 10 lines of any file

**Tail=** The 'head' command displays the starting content of a file. By default, it displays last 10 lines of any file

**Man=** The "man" is a short term for manual page.

**Locate=** The locate command and [find](https://www.javatpoint.com/linux-find)command is used to search a file by name.

Syntax:

1. locate [OPTION]... PATTERN...
2. We can limit the search results to avoid redundancy by using the "-n" option.

For example. To display the just 5 results from our queries, execute the command as follows:

locate -n 5 "\*.txt"

What is the difference between locate and find in Linux?

**locate simply looks its database and reports the file location.** **find does not use a database, it traverses all the directories and their sub directories and looks for files matching the given criterion**.

### Display The Number of Matching Entries

To display the sum of matching files, execute the command with the '-c' option. Consider the below command:

locate -c Demo\*

### Ignoring the case sensitivity

As we know, the Linux terminal is case sensitive. So, if we search a file in uppercase, it will only list the files in the upper case. To ignore the case sensitivity, execute the command with the '-i' option as follows:

1. locate -i "demo.txt"

### Updating the mlocate Database

The locate command depends upon the 'mlocate' database. So, if the locate command is not working properly, we need to update the database. To update the database, execute the 'updatedb' command as follows:

1. sudo updatedb

### Display Only Available Files in Our System

Sometimes, the locate command results deleted files. To avoid seeing the results for the deleted files, execute the command with the '-e' option. It will only display the files that are physically available in our system. Consider the below command:

1. locate -i -e \*demo.txt\*

### Track status of the mlocate Database

To locate the database statistics, execute the command with the '-S' option as follows:

1. locate -S

### Separate Output Entries with a Different Separator

The default output separator of the locate command is a newline. But we can change the default separator, For example, To use the ASCII NUL, execute the command using the '-0' option. Consider the below command:

1. locate -i -0 "demo.txt"

**find=** The find command helps us to find a particular file within a directory.

1. find **<location>** **<comparison-criteria>** **<search-term>**
2. find . -name "\*.txt"

**diff=**

**file=** file command is used to determine the file type. It does not care about the extension used for file. It simply uses file command and tell us the file type. It has several options.

1. file **<filename>**

|  |  |
| --- | --- |
| **Option** | **Function** |
| [file -s](https://www.javatpoint.com/linux-file-s) | Used for special files. |
| [file \*](https://www.javatpoint.com/linux-file-asterisk) | Used to list types of all the files. |
| [file /directory name/\*](https://www.javatpoint.com/linux-file-directory-name) | Used to list types of all the files from mentioned directory. |
| [file [range]\*](https://www.javatpoint.com/linux-file-range) | It will list out all the files starting from the alphabet present within the given range. |

**Rm=**

|  |  |
| --- | --- |
| **Option** | **Description** |
|  |  |
| [rm \*extension](https://www.javatpoint.com/linux-rm-extension) | Used to delete files having same extension. |
| [rm -r or R](https://www.javatpoint.com/linux-rm-r) | To delete a directory recursively. |
| [rm -i](https://www.javatpoint.com/linux-rm-i) | Remove a file interactively. |
| [rm -rf](https://www.javatpoint.com/linux-rm-rf) | Remove a directory forcefully. |

**Mkdir=**

|  |  |
| --- | --- |
| **Options** | **Description** |
| [mkdir -p, -parents](https://www.javatpoint.com/linux-mkdir-p) | Add directory including its sub directory. |
| [mkdir -v, -verbose](https://www.javatpoint.com/linux-mkdir-v) | Print a message for each created directory. |
| [mkdir -m -mode=MODE](https://www.javatpoint.com/linux-mkdir-m-mode) | Set access privilege. |

**Rmdir=** This command is used to delete a directory. But will not be able to delete a directory including a sub-directory. It means, a directory has to be empty to be deleted.

**ln and ln –s=**

|  |  |
| --- | --- |
| **ls option** | **Description** |
| [ls -a](https://www.javatpoint.com/linux-ls#linux-ls-a) | In Linux, hidden files start with . (dot) symbol and they are not visible in the regular directory. The (ls -a) command will enlist the whole list of the current directory including the hidden files. |
| [ls -l](https://www.javatpoint.com/linux-ls#linux-ls-l) | It will show the list in a long list format. |
| ls -lh | This command will show you the file sizes in human readable format. Size of the file is very difficult to read when displayed in terms of byte. The (ls -lh)command will give you the data in terms of Mb, Gb, Tb, etc. |
| ls -lhS | If you want to display your files in descending order (highest at the top) according to their size, then you can use (ls -lhS) command. |
| [ls -l - -block-size=[SIZE]](https://www.javatpoint.com/linux-ls#linux-ls-l-block-size) | It is used to display the files in a specific size format. Here, in [SIZE] you can assign size according to your requirement. |
| [ls -d \*/](https://www.javatpoint.com/linux-ls#linux-ls-d-asterisk-slash) | It is used to display only subdirectories. |
| [ls -g or ls -lG](https://www.javatpoint.com/linux-ls#linux-ls-g) | With this you can exclude column of group information and owner. |
| ls -n | It is used to print group ID and owner ID instead of their names. |
| [ls --color=[VALUE]](https://www.javatpoint.com/linux-ls#linux-ls-color) | This command is used to print list as colored or discolored. |
| ls -li | This command prints the index number if file is in the first column. |
| ls -p | It is used to identify the directory easily by marking the directories with a slash (/) line sign. |
| ls -r | It is used to print the list in reverse order. |
| ls -R | It will display the content of the sub-directories also. |
| ls -lX | It will group the files with same extensions together in the list. |
| ls -lt | It will sort the list by displaying recently modified filed at top. |
| [ls ~](https://www.javatpoint.com/linux-ls#linux-ls-tilde) | It gives the contents of home directory. |
| [ls ../](https://www.javatpoint.com/linux-ls#linux-ls-dot-dot-slash) | It give the contents of parent directory. |
| ls --version | It checks the version of ls command. |

**gzip and gunzip=**

1. gzip **<file1>** **<file2>** **<file3>**. . .
2. gunzip **<file1>** **<file2>** **<file3>**. . .
3. **Compressing Multi Files Together**

If you want to compress more than one file together, you can use 'cat' and gzip command with pipe command.

**Syntax:**

1. cat **<file1>** **<file2>**. . | gzip **>** **<newFile.gz>**

## **gzip -l**

The 'gzip -l' command tells about the compression ratio or how much the original file has compressed.

**Syntax:**

1. gzip -l **<file1>** **<file2>**. .

## **How To Compress A Directory**

The gzip command will not be able to compress a directory because it can only compress a single file. To compress a directory you have to use **'tar'** command.

Hyphen **(-)** is not mandatory in 'tar' command.

**'c'** is to **create**,

**'v'** is for verbose, to**display** output,

**'f'** to mention**destination** of your output file,

**'z'** for specifying **compress**with gzip.

**Syntax:**

1. tar cf - **<directory>** | gzip **>** **<directoryName>**

**makefile=**

**chmod=** In Unix-like operating systems, the **chmod** command is used to change the access mode of a file.  
The name is an abbreviation of **change mode**.

**Syntax :**

chmod [reference][operator][mode] file...

r Permission to read the file.

w Permission to write (or delete) the file.

x Permission to execute the file, or, in

the case of a directory, search it.

**Chown=** Linux chown command is used to change a file's ownership, directory, or symbolic link for a [user](https://www.javatpoint.com/linux-users) or [group](https://www.javatpoint.com/linux-groups). The chown stands for change owner. In [Linux](https://www.javatpoint.com/linux-tutorial), each file is associated with a corresponding owner or group.

The Linux system may have multiple users. Every user has a unique name and user ID. If only a user is available in the system, the user will be the owner of each file.

**Syntax:**

Below is the general syntax of the chown command:

1. chown [**OPTION**]... [OWNER][:[**GROUP**]] FILE...

### Display the User and Group Ownership of a File

To display the user and group owner of a file, execute the "ls -l" command with a specific file name. Consider the below command:

1. ls -l Demo.txt

### Change the owner of a File (Using user name)

To change the owner of a file, pass the user name (new owner) with the chown command as follows:

1. sudo chown <username> <File **name**>

### Change the Owner of a File (Using UID)

We can change the owner of a file or directory using the UID of a user. To change the ownership by using the UID, pass the UID with chown command as follows:

1. sudo chown 1000 Demo.txt

### Change the Group of a File

To change the group of a file, specify group name or GID with the chown command. It represents similar behavior as chgrp command. Execute the chown command specifying the group name separated by a colon as follows:

1. sudo chown :groupname Demo.txt

### Change the User and Group Name

We can change the user and group names together. To change the user and group name at once, execute the chown command as follows:

1. sudo chown <User **name**/UID>:<**Group** **name**/GID> <File **name**>

## SSH

SSH which stands for Secure Shell, It is used to connect to a remote computer securely. Compare to Telnet, SSH is secure wherein the client /server connection is authenticated using a digital certificate and passwords are encrypted. Hence it’s widely used by system administrators to control remote Linux servers.

The syntax to log into a remote Linux machine using SSH is

SSH username@ip-address or hostname

## FTP

FTP **is file transfer protocol**. It’s the **most preferred protocol for** **data transfer** amongst computers.

You can use FTP to –

* Logging in and establishing a connection with a remote host
* Upload and download files
* Navigating through directories
* Browsing contents of the directories
* The syntax to establish an FTP connection to a remote host is –
* ftp hostname="" or=""

## Telnet

Telnet helps to –

* connect to a remote Linux computer
* run programs remotely and conduct administration
* The syntax for this utility is:

telnet hostname="" or=""

Example:

telnet localhost

## Establishing an SFTP connection

SFTP works on a client-server model. It is a subsystem of SSH and supports all SSH authentication mechanisms.

To open an SFTP connection to a remote system, use the sftp command followed by the remote server username and the IP address or domain name:

**Vim editor**

install vim on Debian based Linux like ubuntu run the command:

sudo apt-get install vim

To install vim on an arch-based distro run the following command:

sudo pacman -S vim

Now vim will be installed on your system.

Now to exit type command

:!q

### ****Now, Let’s Start to use Vim****

To open a file in vim editor just write the file name after the vim command in the terminal as follows:

vim filename.txt

**Moving the Cursor:**

Till now, we are using the arrow keys to move the cursor into a file, but it is not recommended to use arrow keys in vim. Vim provides the special key to move the cursor in the vim editor following are the keys used to move into a file.

\* k -> move up ^

\* j -> move down k

\* h -> move right < h l >

\* l -> move left

**Text Editing: Deletion**

We provide x key in command mode to delete the character under the cursor. Move the cursor to the character which has to delete and press Esc key  and then press the x key

x

The character under the cursor will be deleted

**Count:**Count is the number for which replete the motion for count number. Here is a demonstration of the use of count and motion

To move courser 2 words forward use the following command

2w

**Deletion Commands:**Always use the Esc key to go into normal mode and use the insertion, deletion keys, and other keys.

To **delete the word** move the cursor to the beginning of the word and use dw command in normal mode. The word under the cursor will be deleted.

dw

To delete more than one word in a single line use the following command.

To delete 2 words use the command

d2w

To **delete the line** move cursor to the beginning of the line and use d$ command in normal mode. The line under the cursor will be deleted.

d$

**Undo and Redo:**

As we are programmers most time we are using undo and redo .vim to provide these to both features in it. To undo press u key in normal mode

u

To redo use the ctrl+r key in normal mode in vim

ctrl+r

**Replace :**

To replace the character under the cursor use rx command where ‘x’ is a character is to replace.

r

**Change Operator:**

In vim c is used as a change operator. To replace the word use ce command

ce

To replace the line use c$ command

c$

This command will delete the contains from the cursor to the end of the line. Then this will automatically go into insert mode then you can put anything in that line,

**Cursor location:**

We can use the h,j,k,l key to move the cursor in the file but it will be difficult to move in big file vim provide more commands to move into the file.

To move the cursor at the start of the file use the gg command

:gg

To move the cursor at the bottom of the file use the G command

:G

To view the current cursor location in the file using the following command:

:ctrl+g

In place of 32 use can put any line number.

**Search:**

To search the word After the cursor uses the backslash key and then write the word and press enter.

:/word

Use n to move on next matching word

:n

Use N to move on previous matching word

:N

**Search and Replace:**

To replace the word in file use s/ command in vim like

:s/searchword/replaceword/

To do replace all occurrence of word use g

:s/searchword/replaceword/g

This command will replace the word globally.

To confirm before replacements of words use gc

:s/searchword/replaceword/gc

To use this command in the whole file use % before the command

:%s/searchword/replaceword/gc

**Vim configuration:**

To configure vim, vim comes with the .vimrc file in the home directory if this file is not there then create one. Then open the file in vim by command

vim ~/.vimrc

You can put your all configurations in this file.

You can set line number by using a simple command

set number

Put this command in your .vimrc file

To enable syntax highlighter in vim use command

syntax on

Put this command in .vimrc file then save file and exit after opening file again you will see the line numbers in vim.

Vim comes with many color schemes using the command:

colorscheme COLOR\_SCHEME\_NAME

Replace COLOR\_SCHEME\_NAME with any color scheme between Default, blue, darkblue, delek, desert, elford, evening, industry, koehler, morning, murphy, pablo, peachpuff, ron, shine, slate, torte, zellner then save file and exit then again open file to see changes.

**Linux shell programming**

**What is shell?**

A **Shell** provides you with an interface to the Unix system. It gathers input from you and executes programs based on that input. When a program finishes executing, it displays that program's output.

Shell is an environment in which we can run our commands, programs, and shell scripts. There are different flavors of a shell, just as there are different flavors of operating systems. Each flavor of shell has its own set of recognized commands and functions.

**Different types of Linux shells**

**SHELL** is a program which provides the interface between the user and an operating system. When the user logs in OS starts a shell for user. **Kernel** controls all essential computer operations, and provides the restriction to hardware access, coordinates all executing utilities, and manages Resources between process. Using kernel only user can access utilities provided by operating system.

**Types of Shell:**

* **The C Shell –**

Denoted as **csh**

Bill Joy created it at the University of California at Berkeley. It incorporated features such as aliases and command history. It includes helpful programming features like built-in arithmetic and C-like expression syntax.

* **The Bourne Shell –**

Denoted as **sh**

It was written by Steve Bourne at AT&T Bell Labs. It is the original UNIX shell. It is faster and more preferred. It lacks features for interactive use like the ability to recall previous commands. It also lacks built-in arithmetic and logical expression handling. It is default shell for Solaris OS.

* **The Korn Shell**

It is denoted as **ksh**

It Was written by David Korn at AT&T Bell LabsIt is a superset of the Bourne shell.So it supports everything in the Bourne shell.It has interactive features. It includes features like built-in arithmetic and C-like arrays, functions, and string-manipulation facilities.It is faster than C shell. It is compatible with script written for C shell.

* **GNU Bourne-Again Shell –**

Denoted as **bash**

It is compatible to the Bourne shell. It includes features from Korn and Bourne shell.

**Bourne Again Shell (BASH)**

## **What Does Bourne Again Shell (Bash) Mean?**

Bourne again shell (Bash) is a free Unix shell that can be used in place of the Bourne shell. It is a complete implementation of the IEEE Portable Operating System Interface for Unix (POSIX) and Open Group shell specification.  
  
Bash is basically a command processor that typically runs in a text window, allowing the user to type commands that cause actions. It can read commands from a file, called a script. Like all Unix shells it supports the following:

* File name wildcarding
* Piping
* Hear documents
* Command execution
* Variables and control structures for condition testing and iteration

## **Techopedia Explains Bourne Again Shell (Bash)**

Bash was written for the GNU Project by Brian Fox. It is called Bourne again shell for many reasons, the first being that it is the open-source version of the Bourne shell and the second as a pun on the concept of being born again. Its acronym is also a description of what the project did, which was to bash together sh, csh, and ksh features.  
  
A Unix shell is a command-line interpreter that provides users with a basic user interface. It allows users to communicate with the system through a series of commands that are typed in the command-line window. There are no buttons or pop-up windows in a shell, simply lots and lots of text.  
  
Essentially, Bash allows users of Unix-like systems to control the innermost components of the operating system using text-based commands.  
  
Bash has a number of extensions and runs on Unix-like operating systems like Linux and Mac OS X. It was ported to Windows through the Subsystem for UNIX-based Applications (SUA) and by POSIX emulation using Cygwin or MSYS. It can even be used in MS-DOS.

**Shell variables (environment and user defined)**

Variables are the containers which store data or a useful piece of information as the value inside them. Below is the syntax for a variable:

## **Types of Bash Variables**

There are two types of variables in a shell or any UNIX system.

1. System-Defined Variables
2. User-Defined Variables

**1. System-Defined Variables:** These are the pre-defined variables as they are created and maintained by the LINUX operating system itself. Their standard convention is that generally they are defined in capital letters, i.e., UPPER\_CASE. So whenever you see a variable defined in upper cases, most likely, they are the system-defined variables.

**2. User-defined Variables:** These variables are created and maintained by the user. Generally, these types of variables are defined in LOWER\_CASES. There is not any strict rule to write these variables in lower-cases. We can write them in upper-cases also.

**Shell files (.bashrc, .profile, .bash\_profile, .bash\_logout**

**.bashrc**

The **.bashrc**file is a script file that’s executed when a user logs in. The file itself contains a series of configurations for the terminal session. This includes setting up or enabling: coloring, completion, shell history, command aliases, and more.

It is a [hidden file](https://www.linuxfordevices.com/tutorials/linux/hidden-files-in-linux) and simple [ls command](https://www.journaldev.com/24210/ls-command-in-linux-unix) won’t show the file.

**.profile**

. profile file in Linux **comes under the System startup files**(defines user environment after reading the initialization files that you have set up when you log in to shell). File like /etc/profile controls variables for profile of all users of the system whereas, . profile allows you to customize your own environment.

**.bash\_profile**

bash\_profile file is **a personal initialization file for configuring the user environment**. The file is defined in your home directory and can be used for the following: Modifying your working environment by setting custom environment variables and terminal settings.

**.bash\_logout**

The . bash\_logout file is the **individual login shell cleanup file**. It is executed when a login shell exits. This file exists in the user's home directory.

**.bashrc**

bashrc file is **a script file that's executed when a user logs in**. The file itself contains a series of configurations for the terminal session. This includes setting up or enabling: coloring, completion, shell history, command aliases, and more. It is a hidden file and simple ls command won't show the file.

**Positional parameters**

**Positional Parameters** are used to pass the information to the shell directly with command line argument or indirectly using set and command substitution. It can be assigned from shell argument and it is denoted by $N, where N is the single digit and referred as parameter N. if N is double digit use braces ${N}.

#### Command Line Argument for positional Parameters

You can pass many arguments in command line.  
For Example we run a script script.sh with 10 argument a to j

* $ ./script.sh a b c d e f g h i j

Now the detail of output we get from positional parameters

* **$0** is the name of script

        ./script.sh

* **$1** is the first parameter passed.

        a

* **$2** is second parameter passed.

        b

* **${10}** is the 10 parameter passed, use bracket for double digit

        j

* if no bracket is used i.e. **$10** then the output is

        a0

* **$@** lists all positional parameters, it sees parameters as different words.

         a b c d e f g h i j

* **$\*** lists all parameter, it sees parameters as a single word if quoted (“$\*”). Use IFS to separate the different word. See the last example for this IFS.

        a b c d e f g h i j

* **$#** is the number of parameter passed.

         10  
  
In command-line, however, **$0** is the name of the shell.  
  
$ $0  
  
-bash

**Wild cards**

Wildcards are a set of building blocks that allow you to create a pattern defining a set of files or directories. As you would remember, whenever we refer to a file or directory on the command line we are actually referring to a path. Whenever we refer to a path we may also use wildcards in that path to turn it into a set of files or directories.

Here is the basic set of wildcards:

* **\*** - represents zero or more characters
* **?** - represents a single character
* **[]** - represents a range of characters

The mechanism here is actually kinda interesting. On first glance you may assume that the command above ( ls ) receives the argument **b\*** then proceeds to translate that into the required matches. It is actually bash (The program that provides the command line interface) that does the translation for us. When we offer it this command it sees that we have used wildcards and so, before running the command ( in this case ls ) it replaces the pattern with every file or directory (ie path) that matches that pattern. We issue the command:

* ls b\*

Then the system translates this into:

* ls barry.txt blah.txt bob

and then executes the program. The program never sees the wildcards and has no idea that we used them. This is funky as it means we can use them on the command line whenever we want. We are not limited to only certain programs or situations.

**Tput command**

The tput command **allows shell scripts to do things like clear the screen, underline text, and center text no matter how wide the screen is**. To do these things, it translates the terminal-independent name of a terminal capability into its actual value for the terminal type being used.

Syntax

**For Outputting Terminal Information**

**tput** [  [**-T**](https://www.ibm.com/docs/en/aix/7.2?topic=t-tput-command#tput__row-d3e110719) *Type*] [ *CapabilityName {clear, init, longname, reset}* [ *Parameters*... ]]

**For Using stdin to Process Multiple Capabilities**

**tput**  [ [**-S**](https://www.ibm.com/docs/en/aix/7.2?topic=t-tput-command#tput__row-d3e110713) ]

1. To clear the screen for the current terminal, enter:

tput clear

1. To display the number of columns for the current terminals, enter:

tput cols

1. To display the number of columns for the aixterm terminal, enter:

tput  [-T](https://www.ibm.com/docs/en/aix/7.2?topic=t-tput-command#tput__row-d3e110719)aixterm cols

1. To set the shell variable **bold** to the begin standout mode sequence and the shell variable **offbold** to the end standout mode sequence, enter:
2. bold=`tput smso`

offbold='tput rmso'

Entering these commands might be followed by the following prompt:

echo "${bold}Name: ${offbold} \c"

1. To set the exit value to indicate if the current terminal is a hardcopy terminal, enter:

tput hc

1. To initialize the current terminal, enter:

tput init

**Shell metacharacters**

The command options, option arguments and command arguments are separated by the space character. However, we can also use special characters called metacharacters in a Linux command that the shell interprets rather than passing to the command.

The shell metacharacters are listed here for reference:

|  |  |
| --- | --- |
| Symbol | Meaning |
| > | Output redirection |
| >> | Output redirection (append) |
| < | Input redirection |
| \* | File substitution wildcard; zero or more characters |
| ? | File substitution wildcard; one character |
| [ ] | File substitution wildcard; any character between brackets |
| `cmd` | Command Substitution |
| $(cmd) | Command Substitution |
| | | The Pipe (|) |
| ; | Command sequence, Sequences of Commands |
| [ ] | File substitution wildcard; any character between brackets |
| || | OR conditional execution |
| && | AND conditional execution |
| ( ) | Group commands, Sequences of Commands |
| & | Run command in the background, Background Processes |
| # | Comment |
| $ | Expand the value of a variable |
| \ | Prevent or escape interpretation of the next character |
| << | Input redirection |

**Command line expansion**

When a command is entered in the command line, it expands into its output which is displayed.

This is called expansion.

The command you're typing will be printed with the help of echo command on the terminal. This command will be useful when you want to check what your command is doing in the shell.

|  |  |
| --- | --- |
| **Command** | **Function** |
| set -x | Used to enable shell expansion. |
| set +x | Used to disable shell expansion. |

## **set -x**

The 'set -x' command enables shell command display.

**Syntax:**

1. echo **<text>**

## **set +x**

The 'set +x' command disables shell command display.

**Syntax:**

1. echo **<text>**

### The Directory Stack

**dirs**

dirs [-clpv] [+*N* | -*N*]

Display the list of currently remembered directories. Directories are added to the list with the pushd command; the popd command removes directories from the list. The current directory is always the first directory in the stack.

**-c**

Clears the directory stack by deleting all of the elements.

**-l**

Produces a listing using full pathnames; the default listing format uses a tilde to denote the home directory.

**-p**

Causes dirs to print the directory stack with one entry per line.

**-v**

Causes dirs to print the directory stack with one entry per line, prefixing each entry with its index in the stack.

**+*N***

Displays the *N*th directory (counting from the left of the list printed by dirs when invoked without options), starting with zero.

**-*N***

Displays the *N*th directory (counting from the right of the list printed by dirs when invoked without options), starting with zero.

**popd**

popd [-n] [+*N* | -*N*]

When no arguments are given, popd removes the top directory from the stack and performs a cd to the new top directory. The elements are numbered from 0 starting at the first directory listed with dirs; that is, popd is equivalent to popd +0.

**-n**

Suppresses the normal change of directory when removing directories from the stack, so that only the stack is manipulated.

**+*N***

Removes the *N*th directory (counting from the left of the list printed by dirs), starting with zero.

**-*N***

Removes the *N*th directory (counting from the right of the list printed by dirs), starting with zero.

**pushd**

pushd [-n] [*+N* | *-N* | *dir*]

Save the current directory on the top of the directory stack and then cd to *dir*. With no arguments, pushd exchanges the top two directories and makes the new top the current directory.

**-n**

Suppresses the normal change of directory when rotating or adding directories to the stack, so that only the stack is manipulated.

**+*N***

Brings the *N*th directory (counting from the left of the list printed by dirs, starting with zero) to the top of the list by rotating the stack.

**-*N***

Brings the *N*th directory (counting from the right of the list printed by dirs, starting with zero) to the top of the list by rotating the stack.

***dir***

Makes *dir* be the top of the stack, making it the new current directory as if it had been supplied as an argument to the cd builtin.

## **Whats a job in Linux**

A job is a process that the shell manages. Each job is assigned a sequential job ID. Because a job is a process, each job has an associated PID. There are three types of job statuses:  
1. **Foreground**: When you enter a command in a terminal window, the command occupies that terminal window until it completes. This is a foreground job.  
2. **Background**: When you enter an ampersand (&) symbol at the end of a command line, the command runs without occupying the terminal window. The shell prompt is displayed immediately after you press Return. This is an example of a background job.  
3. **Stopped**: If you press Control + Z for a foreground job, or enter the stop command for a background job, the job stops. This job is called a stopped job.

## **Job Control Commands**

Job control commands enable you to place jobs in the foreground or background, and to start or stop jobs. The table describes the job control commands.

|  |  |
| --- | --- |
| **Option** | **Description** |
| jobs | Lists all jobs |
| bg %n | Places the current or specified job in the background, where n is the job ID |
| fg %n | Brings the current or specified job into the foreground, where n is the job ID |
| Control-Z | Stops the foreground job and places it in the background as a stopped job |

**Built-ins and functions**

**builtin** command is used to run a shell builtin, passing it arguments(args), and also to get the exit status. The main use of this command is to define a shell function having the same name as the shell builtin by keeping the functionality of the builtin within the function.

**Syntax:**

builtin [shell-builtin [arg ..]]

**Function** is a command in linux which is used to create functions or methods.

**1.using function keyword** : A function in linux can be declared by using keyword function before the name of the function. Different statements can be separated by a semicolon or a new line.   
**SYNTAX** 

function name { COMMANDS ; }

**2.using parenthesis** : A function can also be declared by using parenthesis after the name of the function. Different statements can be separated by a semicolon or a new line.   
**SYNTAX** 

name () { COMMANDS ; }