# Module 7

## Introduction to shell script

Introduction to Linux Shell and Shell Scripting

If you are using any major operating system you are indirectly interacting to **shell**. If you are running Ubuntu, Linux Mint or any other Linux distribution, you are interacting to shell every time you use terminal. Today we will discuss about linux shells and shell scripting so before understanding shell scripting we have to get familiar with following terminologies:

* Kernel
* Shell
* Terminal

## What is Kernel

The kernel is a computer program that is the core of a computer’s operating system, with complete control over everything in the system. It manages following resources of the Linux system –

* File management
* Process management
* I/O management
* Memory management
* Device management etc.

## What is Shell

A shell is special user program which provide an interface to user to use operating system services. Shell accept human readable commands from user and convert them into something which kernel can understand. It is a command language interpreter that execute commands read from input devices such as keyboards or from files. The shell gets started when the user logs in or start the terminal.

## Command Line Shell

Shell can be accessed by user using a command line interface. A special program called Terminal in linux/macOS or Command Prompt in Windows OS is provided to type in the human readable commands such as “cat”, “ls” etc. and then it is being execute.

## Shell Scripting

Usually shells are interactive that mean, they accept command as input from users and execute them. However some time we want to execute a bunch of commands routinely, so we have type in all commands each time in terminal.

As shell can also take commands as input from file we can write these commands in a file and can execute them in shell to avoid this repetitive work. These files are called **Shell Scripts** or **Shell Programs**. Shell scripts are similar to the batch file in MS-DOS. Each shell script is saved with **.sh** file extension eg. **myscript.sh**

A shell script have syntax just like any other programming language. If you have any prior experience with any programming language like Python, C/C++ etc. it would be very easy to get started with it.

A shell script comprises following elements –

* Shell Keywords – if, else, break etc.
* Shell commands – cd, ls, echo, pwd, touch etc.

Functions Control flow – if..then..else, case and shell loops etc.

## Why do we need shell scripts

There are many reasons to write shell scripts:

* To avoid repetitive work and automation .
* System admins use shell scripting for routine backups.
* System monitoring .
* Adding new functionality to the shell etc.

## Advantages of shell scripts

* The command and syntax are exactly the same as those directly entered in command line, so programmer do not need to switch to entirely different syntax.
* Writing shell scripts are much quicker .
* Quick start .

## • Interactive debugging etc.

## Disadvantages of shell scripts

* Given to costly errors, a single mistake can change the command which might be harmful.
* Not well suited for large and complex task .
* Provide minimal data structure unlike other scripting languages. etc

**Simple shell scripts, Interactive shell script :**

* A simple shell script typically refers to a script that performs a straightforward task or a series of simple tasks. It may not involve complex logic or extensive error handling. Simple shell scripts are often used for tasks like file manipulation, text processing, or running a sequence of commands.

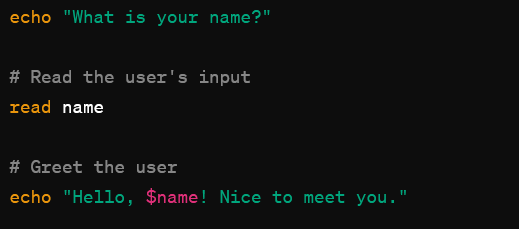
echo “This is CCLMS”

Cal

Date

**Interactive shell script :**

* an interactive shell script is one that interacts with the user, typically by prompting for input, displaying information, and responding to user actions. Interactive shell scripts often incorporate features such as user input validation, menu systems, and interactive command-line interfaces (CLIs). These scripts allow users to interact with the script in real-time, providing input and receiving immediate feedback.



There are **5** basic operators in bash/shell scripting:

* Arithmetic Operators
* Relational Operators
* Boolean Operators
* Bitwise Operators
* File Test Operators

1. **Arithmetic Operators**: These operators are used to perform normal arithmetics/mathematical operations. There are 7 arithmetic operators:
   * **Addition (+)**: Binary operation used to add two operands.
   * **Subtraction (-)**: Binary operation used to subtract two operands.
   * **Multiplication (\*)**: Binary operation used to multiply two operands.
   * **Division (/)**: Binary operation used to divide two operands.
   * **Modulus (%)**: Binary operation used to find remainder of two operands.
   * **Increment Operator (++)**: Unary operator used to increase the value of operand by one.
   * **Decrement Operator (- -)**: Unary operator used to decrease the value of a operand by one

1. **Relational Operators**: Relational operators are those operators which define the relation between two operands. They give either true or false depending upon the relation. They are of 6 types:
   * **‘==’ Operator**: Double equal to operator compares the two operands. Its returns true is they are equal otherwise returns false.
   * **‘!=’ Operator**: Not Equal to operator return true if the two operands are not equal otherwise it returns false.
   * **‘<‘ Operator**: Less than operator returns true if first operand is less than second operand otherwise returns false.
   * **‘<=’ Operator**: Less than or equal to operator returns true if first operand is less than or equal to second operand otherwise returns false
   * **‘>’ Operator**: Greater than operator return true if the first operand is greater than the second operand otherwise return false.
   * **‘>=’ Operator**: Greater than or equal to operator returns true if first operand is greater than or equal to second operand otherwise returns false

1. **Logical Operators** : They are also known as boolean operators. These are used to perform logical operations. They are of 3 types:
   * **Logical AND (&&)**: This is a binary operator, which returns true if both the operands are true otherwise returns false.
   * **Logical OR (||)**: This is a binary operator, which returns true is either of the operand is true or both the operands are true and return false if none of then is false.
   * **Not Equal to (!)**: This is a unary operator which returns true if the operand is false and returns false if the operand is true.

1. **Bitwise Operators**: A bitwise operator is an operator used to perform bitwise operations on bit patterns. They are of 6 types:
   * **Bitwise And (&)**: Bitwise & operator performs binary AND operation bit by bit on the operands.
   * **Bitwise OR (|)**: Bitwise | operator performs binary OR operation bit by bit on the operands.
   * **Bitwise XOR (^)**: Bitwise ^ operator performs binary XOR operation bit by bit on the operands.
   * **Bitwise complement (~)**: Bitwise ~ operator performs binary NOT operation bit by bit on the operand.
   * **Left Shift (<<)**: This operator shifts the bits of the left operand to left by number of times specified by right operand.
   * **Right Shift (>>)**: This operator shifts the bits of the left operand to right by number of times specified by right operand.

***They manipulate individual bits within binary representations of data. These operators are typically used in low-level programming, such as systems programming, device drivers, cryptography, and optimization algorithms. Understanding bitwise operators can be beneficial for understanding low-level programming concepts and optimizing performance-critical code.***

1. **File Test Operator**: These operators are used to test a particular property of a file.
   * **-b operator**: This operator check whether a file is a block **special file or not**. It returns true if the file is a block special file otherwise false.
   * **-c operator**: This operator checks whether a file is a **character special file or not**. It returns true if it is a character special file otherwise false.
   * **-d operator**: This operator checks if the given **directory exists or not**. If it exists then operators returns true otherwise false.
   * **-e operator**: This operator checks whether the given **file exists or not**. If it exits this operator returns true otherwise false.
   * **-r operator**: This operator checks whether the given file has **read access or not**. If it has read access then it returns true otherwise false.
   * **-w operator**: This operator check whether the given file has **write access or not**. If it has write then it returns true otherwise false.
   * **-x operator**: This operator check whether the given file has **execute access or not**. If it has execute access then it returns true otherwise false.
   * **-s operator**: This operator checks the size of the given file. If the size of given file is greater than 0 then it returns true otherwise it is false.

**Conditional Statements:** There are total 5 conditional statements which can be used in bash programming

* 1. if statement
  2. if-else statement
  3. if..elif..else..fi statement (Else If ladder)
  4. if..then..else..if..then..fi..fi..(Nested if)
  5. switch statement

Their description with syntax is as follows:

## if statement

This block will process if specified condition is true.

***Syntax:***

if [ expression ] then

statement

fi

## if-else statement

If specified condition is not true in if part then else part will be execute.

***Syntax***

if [ expression ] then statement1

else statement2

fi

## if..elif..else..fi statement (Else If ladder)

To use multiple conditions in one if-else block, then elif keyword is used in shell. If expression1 is true then it executes statement 1 and 2, and this process continues. If none of the condition is true then it processes else part.

***Syntax*** if [ expression1 ] then statement1 statement2

.

.

elif [ expression2 ] then

statement3 statement4

.

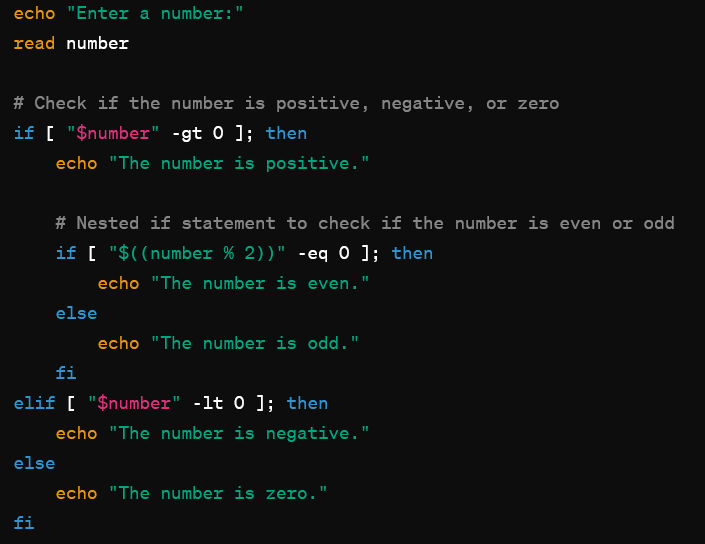
.

else statement5

fi

## if..then..else..if..then..fi..fi..(Nested if)

Nested if-else block can be used when, one condition is satisfies then it again checks another condition. In the syntax, if expression1 is false then it processes else part, and again expression2 will be check.



## switch statement

case statement works as a switch statement if specified value match with the pattern then it will execute a block of that particular pattern

When a match is found all of the associated statements until the double semicolon (;;) is executed.

A case will be terminated when the last command is executed.

If there is no match, the exit status of the case is zero. ***Syntax:***

case expression in

Pattern 1) Statement 1;;

Pattern 2) Statement 2;;

Pattern n) Statement n;;

\*) default statement ;;

esac

Note: ;; (Double Semicolon):

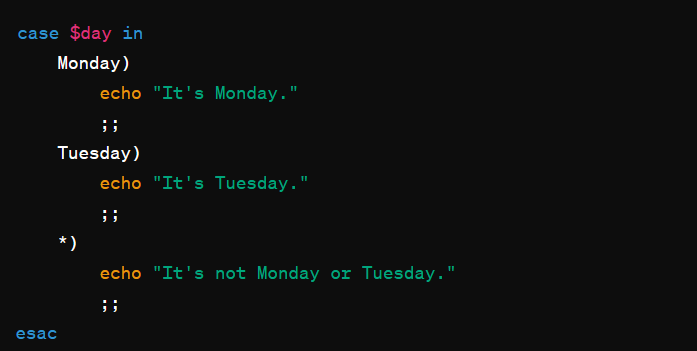
* ;; is used to terminate each pattern and its associated code block within the case statement.
* It signifies the end of the code block for the matched pattern and indicates to Bash that it should move to the next pattern.

\*) (Wildcard Pattern):

* \*) is a wildcard pattern that matches any value.
* It serves as the default case to execute if the expression matches none of the specified patterns.

 esac (Reverse of case):

* esac is the reverse of the case keyword.
* esac is used to terminate the case block and maintain the structure of the case statement.



**Expression evaluation**

**Expression Evaluation with test and [ ]:**

Expression evaluation involves various methods to evaluate conditions, perform computations, and manipulate strings.

In Bash, **test** and **[ ]** are built-in commands used for evaluating expressions and performing conditional tests.

**Syntax :**

# Using 'test' command

test expression

# Using '[' command (same as 'test', but requires closing bracket ']')

[ expression ]

**Example :**

# Check if a file exists

if test -f "myfile.txt"; then

echo "File exists."

fi

# Check if a number is greater than 10

if [ "$num" -gt 10 ]; then

echo "Number is greater than 10."

fi

**Computation with ‘expr’**

**‘expr’** is a command-line utility used for evaluating expressions and performing arithmetic computations.

**Syntax:**

expr expression

**Example:**

**sum=$(expr $num1 + $num2)**

**echo "Sum: $sum"**

**Using ‘expr’ for String Manipulation:**

**‘expr’** can also be used for string manipulation, such as substring extraction, pattern matching, and string length computation.

**Syntax :**

# Extract substring

substring=$(expr substr "$string" start length)

# Search for pattern and return position

position=$(expr index "$string" pattern)

# Compute length of string

length=$(expr length "$string")

Example:

# Extract substring

substring=$(expr substr "Hello, World!" 1 5)

echo "Substring: $substring"

# Output: "Hello"

# Find position of 'World'

position=$(expr index "Hello, World!" "World")

echo "Position: $position"

# Output: 7

# Compute length of string

length=$(expr length "Hello, World!")

echo "Length: $length"

# Output: 12

**Loop (while, for)**

#### **while Loop:**

The while loop is used to execute a block of code repeatedly as long as a specified condition is true.

Syntax :

while condition

do

# Commands to execute

done

Example :

count=1

while [ $count -le 5 ]

do

echo "Count: $count"

count=$((count + 1))

done

#### **for Loop:**

The for loop is used to iterate over a sequence of values (such as numbers, filenames, or strings) and execute a block of code for each value.

**Syntax :** for variable in list

do

# Commands to execute

Done

**Example :**

for i in {1..5}

do

echo "Number: $i"

done

**Use of Positional Parameters:**

Positional parameters refer to the arguments passed to a script or function when it is invoked. The positional parameters are stored in special variables ($1, $2, $3, ..., $9, $@, $\*, $#, $0) and can be accessed within the script to process input data.

* $1, $2, $3, ...: Represent the first, second, third, etc., positional parameters passed to the script.
* $@, $\*: Represent all positional parameters passed to the script as separate arguments or as a single string, respectively.
* $#: Represents the total number of positional parameters passed to the script.
* $0: Represents the name of the script itself.

Example :

echo "Script name: $0"

echo "First argument: $1"

echo "Second argument: $2"

echo "All arguments: $@"

echo "Number of arguments: $#"

**The essential duties of a UNIX system administrator typically include:**

1. **System Installation and Configuration:** Installing, configuring, and maintaining UNIX-based operating systems such as Linux, FreeBSD, or Solaris on servers or workstations.
2. **User and Group Management:** Creating, managing, and maintaining user accounts, groups, permissions, and access controls to ensure proper security measures are in place.
3. **System Monitoring and Performance Tuning:** Monitoring system performance, resource usage, and troubleshooting performance issues. Implementing optimizations to ensure optimal system performance.
4. **Backup and Recovery:** Implementing and maintaining backup strategies to ensure data integrity and availability.
5. **Security Management:** Implementing security measures such as firewalls, and security patches to protect the system from unauthorized access, malware, and other security threats.
6. **Network Configuration and Administration:** Configuring and managing network interfaces, IP addresses, routing tables, and network services such as DNS, DHCP, and NTP.
7. **Filesystem Management:** Creating, mounting, and managing filesystems. Monitoring disk usage and implementing strategies for storage optimization.
8. **Package Management:** Installing, upgrading, and managing software packages using package management tools such as yum, apt, or pkg.
9. **Scripting and Automation:** Writing scripts (e.g., shell scripts, Python scripts) to automate routine tasks, streamline system administration processes, and improve efficiency.
10. **Troubleshooting and Problem Resolution:** Identifying and resolving system issues, errors, and malfunctions. Performing root cause analysis and implementing solutions to prevent recurrence.
11. **Documentation:** Maintaining comprehensive documentation of system configurations, procedures, troubleshooting steps, and changes made to the system to facilitate knowledge sharing and ensure continuity of operations.
12. **User Support:** Providing technical support and assistance to users, including troubleshooting software and hardware issues, answering questions, and providing guidance on system usage.
13. **Compliance and Auditing:** Conducting monthly security audits and assessments to identify and address security vulnerabilities.

As a UNIX system administrator, you will be **responsible for the installation, configuration, and maintenance of our UNIX systems**. In this role, you will troubleshoot server errors, install new system hardware, respond to user issues, and monitor the performance of the network.

**Starting and shutdown.**

**Starting (Boot Process):**

**Power On**: Initially, the system is powered on either by physical means (e.g., pressing the power button) or remotely (e.g., through a management interface).

**BIOS/UEFI Initialization:** The Basic Input/Output System (BIOS) or Unified Extensible Firmware Interface (UEFI) initializes hardware components such as the CPU, memory, storage devices, and peripherals.

**Boot Loader:** The boot loader (e.g., GRUB, LILO) is loaded into memory, which is responsible for loading the operating system kernel into memory.

**Kernel Initialization:** The operating system kernel is loaded into memory and begins its initialization process, including hardware detection, device initialization, and setting up essential system data structures.

**User Login:** Once the system is fully initialized, it prompts for user authentication (if applicable) and presents the login screen or console for user interaction.

**shutdown** - bring the system down.

**/sbin/shutdown** [**-t** *sec*] [**-arkhncfFHP**] *time* [*warning-message*]

**shutdown** brings the system down in a secure way. All logged-in users are notified that the system is going down, and **login**(1) is blocked. It is possible to shut the system down immediately or after a specified delay. All processes are first notified that the system is going down by the signal SIGTERM. This gives programs like **vi**(1) the time to save the file being edited, mail and news processing programs a chance to exit cleanly, etc. **shutdown** does its job by signalling the **init** process, asking it to change the runlevel. Runlevel **0** is used to halt the system, runlevel **6** is used to reboot the system, and runlevel **1** is used to put to system into a state where administrative tasks can be performed; this is the default if neither the *h* or *-r* flag is given to **shutdown**. To see which actions are taken on halt or reboot see the appropriate entries for these runlevels in the file */etc/inittab*.

## OPTIONS

|  |  |
| --- | --- |
| **Tag** | **Description** |
| **-a** | Use **/etc/shutdown.allow**. |
| **-t** *sec* | Tell **init**(8) to wait *sec* seconds between sending processes the warning and the kill signal, before changing to another runlevel. |
| **-k** | Don’t really shutdown; only send the warning messages to everybody. |
| **-r** | Reboot after shutdown. |
| **-h** | Halt or poweroff after shutdown. |
| **-H** | Halt action is to halt or drop into boot monitor on systems that support it. |
| **-P** | Halt action is to turn off the power. |
| **-n** | [DEPRECATED] Don’t call **init**(8) to do the shutdown but do it ourself. The use of this option is discouraged, and its results are not always what you’d expect. |
| **-f** | Skip fsck on reboot. |
| **-F** | Force fsck on reboot. |
| **-c** | Cancel an already running shutdown. With this option it is of course not possible to give the **time** argument, but you can enter a explanatory message on the command line that will be sent to all users. |
| *Time* | When to shutdown. |
| *warning-message* | Message to send to all users. |

The *time* argument can have different formats.

First, it can be an absolute time in the format *hh:mm*, in which *hh* is the hour (1 or 2 digits) and *mm* is the minute of the hour (in two digits). Second, it can be in the format **+***m*, in which *m* is the number of minutes to wait. The word **now** is an alias for **+0**.

If shutdown is called with a delay, it creates the advisory file */etc/nologin* which causes programs such as *login(1)* to not allow new user logins. Shutdown removes this file if it is stopped before it can signal init (i.e. it is cancelled or something goes wrong). It also removes it before calling init to change the runlevel.

The **-f** flag means ‘reboot fast’. This only creates an advisory file */fastboot* which can be tested by the system when it comes up again. The boot rc file can test if this file is present, and decide not to run **fsck**(1) since the system has been shut down in the proper way. After that, the boot process should remove */fastboot*.

The **-F** flag means ‘force fsck’. This only creates an advisory file */forcefsck* which can be tested by the system when it comes up again. The boot rc file can test if this file is present, and decide to run **fsck**(1) with a special ‘force’ flag so that even properly unmounted filesystems get checked. After that, the boot process should remove */forcefsck*.

The **-n** flag causes **shutdown** not to call **init**, but to kill all running processes itself. **shutdown** will then turn off quota, accounting, and swapping and unmount all filesystems.

**Brief idea about user account management (username, password)**

## Creating a user account

To create a normal user account, we use the **useradd** command, like shown in the following snippet of code:

sudo useradd [options] [username]

In the above snippet, the username is the name by which we will create the new account.

Note that the name can not be the same for two users.

In the place of options, we can pass different flags to enable or disable different settings. The following table contains those options and their brief descriptions:

|  |  |
| --- | --- |
| **Option** | **Description** |
| -b, --base-dir *BASE\_DIR* | This option is used to specify the default base directory for the new user being created. *BASE\_DIR* is concatenated with the account name to specify the base directory if the d flag is not used to define the home directory. |
| *-*c, --comment *COMMENT* | This option is used to write a short description of the new account. It is also used as the user's full name for the time being. |

-d, --home *HOME\_DIR* This option is used to specify the home directory for the newly created user.

-D, --defaults This option is used to display the current default values for the `useradd` command

-e, --expiredate *EXPIRE\_DATE* This option is used to specify the date in the YYYY-MMDD format on which the newly created user account will be disabled.

-f, --inactive This option is used to specify the number of days that the account will stay active after its password has expired.

-g, --gid *GROUP* This option is to specify the name or number of the group of newly created user.

-G, --groups This option is used to specify a list of groups which will *Group1[,Group2,...[GroupN]]]* be joined by the newly created user.

-h, --help This option is to display the help message

-k, --skel *SKEL\_DIR* This option is used to specify files and folders which will be copied into the home directory of the newly created user.

-K, --key *KEY=VALUE* This option overrides the default values present in /etc/login.defs. These inclue UID\_MIN, UID\_MAX, UMASK etc.

- I, -no-log-init This option prevents the user to be added from the lastlog and faillog databases.

-m, --create-home This option is used to create the user's home directory if it does not exist.

-M This option is used to prevent the creation of the home directory.

-N, --no-user-group This option is used to prevent the creation of a group with the user's name.

-o, --non-unique This options allows the creation of a user account with a duplicate ID.

-p, --password *PASSWORD* The encrypted password, as returned by the `crypt`.

-r, --system This option is used to create a system account.

-s, --shell *SHELL* This option is used to specify the name of the user's login shell.

-u, --uid *UID* This option is used to set the numerical value of the user'

ID. This value must be unique.

-U, --user-group This option enables the creation of a group with the same name as the user's name.

|  |  |
| --- | --- |
| -Z, --selinux-user *SEUSER* | This option is used to determine the SELinux user for the user's login. |
|  |  |

## Modifying a user account

To modify a user account, we use the **usermod** command as shown in the following snippet:

usermod [options] [username]

In the above snippet, the username is the name of the account that is to be modified.

In the place of options, different flags are passed for different settings.

## Deleting a user account

To delete a user account, we use the **userdel** command as shown in the following snippet:

userdel [options] [username]

In the above snippet, the username is the account’s name that is to be deleted.

In the place of options, different flags are passed for different settings.

## Groups

In Linux or Unix-based operating systems, we can form **groups** of users’ accounts. Groups are used to manage the user accounts collectively. We can manage the access permissions for the entire group.

A single user can be a part of multiple groups, and a group can have multiple users.

The terminal commands related to groups are discussed in detail below.

## Creating a new group

We use the **groupadd** command to create a new group, as shown in the snippet below.

groupdadd [options] [groupname]

In the above snippet, groupname is the name assigned to the newly created group.

In place of options, we can pass different flags for different settings.

## Modifying a group

We use the **groupmod** command to modify an existing group, as shown in the snippet below.

groupmod [options] [groupname]

In the above snippet, groupname is the group’s name that is to be modified.

In place of options, we can pass different flags for different settings.

## Deleting a group

To delete a group, we use the groupdel command as shown in the following snippet:

groupdel [groupname]

Root account

This is also called **superuser** and would have complete and unfettered control of the system. A superuser can run any commands without any restriction. This user should be assumed as a system administrator.

System accounts

System accounts are those needed for the operation of system-specific components for example mail accounts and the **sshd** accounts. These accounts are usually needed for some specific function on your system, and any modifications to them could adversely affect the system. User accounts

User accounts provide interactive access to the system for users and groups of users. General users are typically assigned to these accounts and usually have limited access to critical system files and directories.

Unix supports a concept of *Group Account* which logically groups a number of accounts. Every account would be a part of another group account. A Unix group plays important role in handling file permissions and process management.

Managing Users and Groups

There are four main user administration files −

* **/etc/passwd** − Keeps the user account and password information. This file holds the majority of information about accounts on the Unix system.

• **/etc/shadow** − Holds the encrypted password of the corresponding account.

Not all the systems support this file.

* **/etc/group** − This file contains the group information for each account.
* **/etc/gshadow** − This file contains secure group account information.

Check all the above files using the **cat** command.

The following table lists out commands that are available on majority of Unix systems to create and manage accounts and groups –

|  |  |
| --- | --- |
| **Sr.No.** | **Command & Description** |
| 1 | **useradd**  Adds accounts to the system |
| 2 | **usermod**  Modifies account attributes |
| 3 | **userdel**  Deletes accounts from the system |
| 4 | **groupadd**  Adds groups to the system |
| 5 | **Groupmod**  Modifies group attributes |
| 6 | **Groupdel**  Removes groups from the system |

User Management in Linux

A user is an entity, in a Linux operating system, that can manipulate files and perform several other operations. Each user is assigned an ID that is unique for each user in the operating system. In this post, we will learn about users and commands which are used to get information about the users. After installation of the operating system, the **ID 0 is assigned to the root user** and the IDs 1 to 999 (both inclusive) are assigned to the system users and hence the ids for local user begins from 1000 onwards.

In a single directory, we can create 60,000 users. Now we will discuss the important commands to manage users in Linux.

1. To ***list out all the users in Linux***,

use the [awk command](https://www.geeksforgeeks.org/awk-command-unixlinux-examples/) with -F option. Here, we are accessing a file and printing only first column with the help of *print $1* and *awk*. awk -F':' '{ print $1}' /etc/passwd

1. Using id command, you can get the ***ID of any username***.

Every user has an id assigned to it and the user is identified with the help of this id. By default, this id is also the group id of the user. id username

1. The command to add a user.

*useradd command* adds a new user to the directory. The user is given the ID automatically depending on which category it falls in. The username of the user will be as provided by us in the command. sudo useradd username

1. Using [passwd command](https://www.geeksforgeeks.org/passwd-command-in-linux-with-examples/) to assign a password to a user.

After using this command we have to enter the new password for the user and then the password gets updated to the new password. passwd username

1. ***Accessing a user configuration file***. cat /etc/passwd

This commands prints the data of the configuration file. This file contains information about the user in the format.

username : x : user id : user group id : : /home/username : /bin/bash

1. The command to **change the user ID for a user**.

usermod -u new\_id username

This command can change the user ID of a user. The user with the given username will be assigned with the new ID given in the command and the old ID will be removed.

*Example:* sudo usermod -u 1982 test

1. Command to **Modify the group ID of a user**.

usermod -g new\_group\_id username

This command can change the group ID of a user and hence it can even be used to move a user to an already existing group. It will change the group ID of the user whose username is given and sets the group ID as the given new\_group\_id.

*Example:* sudo usermod -g 1005 test

1. You can **change the user login name** using *usermod* command.

The below command is used to change the login name of the user. The old login name of the user is changed to the new login name provided.

sudo usermod -l new\_login\_name old\_login\_name

1. The **command to change the home directory**. The below command change the home directory of the user whose username is given and sets the new home directory as the directory whose path is provided.

usermod -d new\_home\_directory\_path username

1. You can also **delete a user name**.

The below command deletes the user whose username is provided. Make sure that the user is not part of a group. If the user is part of a group then it will not be deleted directly, hence we will have to first remove him from the group and then we can delete him. userdel -r username .

1. **Home Directory:**

The home directory is a directory associated with each user account, where the user's personal files and configuration settings are stored.

When a user logs in, they are typically placed in their home directory by default.

The path to a user's home directory is usually /home/<username> in many UNIX distributions.

1. **Group ID (GID):**

Every user account is associated with one or more groups, each identified by a Group ID (GID).

The primary group of a user is specified in the /etc/passwd file, while secondary groups are listed in the /etc/group file.

Group membership determines the user's access rights to files, directories, and other system resources.

1. **Disk Quota:**

Disk quota is a mechanism used to limit the amount of disk space a user or group can consume on a filesystem.

Disk quotas help administrators manage disk usage, prevent users from monopolizing disk space, and ensure fair resource allocation.

Quotas can be set for individual users or groups using tools like quota and edquota, and are typically managed at the filesystem level.

1. **Terminal:**

The terminal (or tty) refers to the device or interface through which a user interacts with the system's command-line interface.

When a user logs in, they are typically assigned a terminal session, represented by a virtual terminal (e.g., tty1, tty2)

Terminal settings, such as terminal type and characteristics, can be configured using utilities like stty and environment variables like TERM.