Scan me

Linux Operating System

By

Dr. Amar Panchal

Connect @amar.career

What is Linux?

- Linux is a Unix-like, open source and community-developed operating system (OS) for computers, servers, mainframes, mobile devices and embedded devices.
- The Linux OS was developed by Linus Torvalds in 1991, which sprouted as an idea to improve the UNIX OS.

Linux OS can be found in many different settings

- **Desktop OS** for personal productivity computing
- **Headless server OS** for systems that do not require a graphical user interface (GUI) or directly connected terminal and keyboard. Headless systems are often used for remotely managed networking server and other devices.
- **Embedded device or appliance OS** for systems that require limited computing function.
- **Network OS for routers**, switches, domain name system servers, home networking devices and more. For example, Cisco offers a version of the Cisco Internetwork Operating System (IOS) that uses the Linux kernel.
- **Software development OS** for enterprise software development. For example, git for distributed source control; vim and emacs for source code editing; and compilers and interpreters for almost every programming language.
- **Cloud OS** for cloud instances. Major cloud computing providers offer access to cloud computing instances running Linux for cloud servers, desktops and other services.

Popular Linux distributions

Ubuntu Linux
One of the most popular Linux distributions, Ubuntu is based on Debian Linux.

Linux Mint
A popular desktop Linux distribution, Linux Mint is based on Debian Linux.

Puppy Linux
Puppy Linux is a lightweight Linux distro, intended to be run from removable media.

Fedora
Fedora is a community-supported Linux distribution sponsored by Red Hat, an IBM subsidiary. Red Hat Enterprise Linux is based on Fedora.

Debian Linux
One of the first Linux distributions, first published in 1993. Many other Linux distros, including Ubuntu and Kali, are based on Debian.

SUSE Linux
Includes openSUSE, a community distribution; and SUSE Linux Enterprise, a commercial distribution designed for enterprise use.

Red Hat Enterprise Linux (RHEL)
RHEL is a commercial enterprise Linux distribution, available through subscription. Customers receive patches, bug fixes, updates, upgrades and technical support.

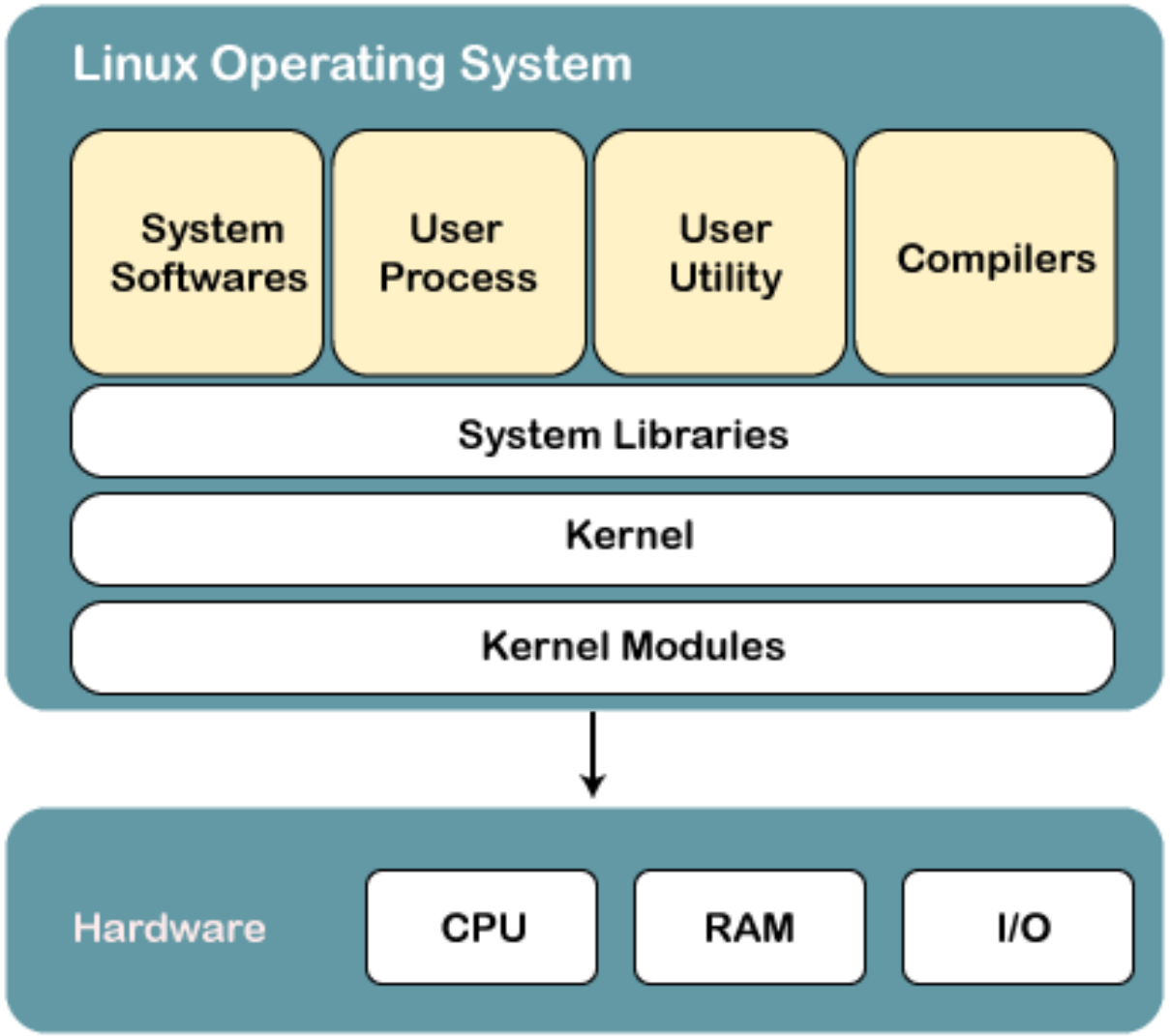
TAILS
TAILS, the amnesic incognito live system, is a lightweight distribution intended to preserve user privacy and anonymity. TAILS runs from removable media and is based on the Debian distribution.

Kali Linux
Kali provides users with what amounts to a hacker's toolkit and is widely used for penetration testing by information security professionals.

The Linux operating system comprises of:

1. **Bootloader –** The software that manages the boot process of your computer. For most users, this will simply be a splash screen that pops up and eventually goes away to boot into the operating system.
2. **Kernel –** This is the one piece of the whole that is actually called 'Linux'. The kernel is the core of the system and manages the CPU, memory, and peripheral devices. The kernel is the lowest level of the OS.
3. **Init system –** This is a sub-system that bootstraps the user space and is charged with controlling daemons. One of the most widely used init systems is systemd, which also happens to be one of the most controversial. It is the init system that manages the boot process, once the initial booting is handed over from the bootloader (i.e., GRUB or GRand Unified Bootloader).
4. **Daemons –** These are background services (printing, sound, scheduling, etc.) that either start up during boot or after you log into the desktop.
5. **Graphical server –** This is the sub-system that displays the graphics on your monitor. It is commonly referred to as the X server or just X.
6. **Desktop environment –** This is the piece that the users actually interact with. There are many desktop environments to choose from (GNOME, Cinnamon, Mate, Pantheon, Enlightenment, KDE, Xfce, etc.). Each desktop environment includes built-in applications (such as file managers, configuration tools, web browsers, and games).
7. **Applications –** Desktop environments do not offer the full array of apps. Just like Windows and macOS, Linux offers thousands upon thousands of high-quality software titles that can be easily found and installed. Most modern Linux distributions (more on this below) include App Store-like tools that centralize and simplify application installation. For example, Ubuntu Linux has the Ubuntu Software Center (a rebrand of GNOME Software) which allows you to quickly search among the thousands of apps and install them from one centralized location.

Structure Of Linux Operating System



Shell

- It is often mistaken that Linus Torvalds has developed Linux OS, but actually he is only responsible for development of Linux kernel.
- Complete Linux system = Kernel + GNU system utilities and libraries + other management scripts + installation. Scripts.

Shell

- A Shell provides you with an interface to the Unix system.
- 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.
- 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
- Let's discuss about linux shells and shell scripting so before understanding shell scripting we have to get familiar with following terminologies .

TYPES

- BASH (Bourne Again SHeII)
- CSH (C Shell)
- KSH (Korn SHeII)

Why do we need 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.

There are several shells are available for Linux systems

- BASH (Bourne Again SHeII)
  - It is most widely used shell in Linux systems.
  - It is used as default login shell in Linux systems and in macOS.
  - It can also be installed on Windows OS.

- CSH (C Shell)
  - The C shell s syntax and usage are very similar to the C programming language.

- KSH (Korn SHeII)
  - The Korn Shell also was the base for the POSIX Shell standard specifications etc.
  - Each shell does the same job but understand different
  - commands and provide different built in functions.

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 CDmmands 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 lo get started with it

Command in Shell Scripting :

- ls — the is command : the list command function in the linux terminal to show all of the major directories filed under a given file system
  - ls –help ls –l (r w x)
- cp – copy file
- rm - remove file
- mv — the mv command move allow a user to move a file to another folder.
- mkdir — the mkdir command : make directory command allows the user to make a new directory.
- cd — the cd command : change directory will allow the user to change between file directories

- Cat filename- see file
- man — the man command : the manual command is used to show the manual of the inputted command.
- touch — the touch command to make file in the directory.
- clear — the clear command : the clear command does exactly what it say clear. it clear all readout and information from the screen.

Steps in Writing a Shell Script

- Write a script file using nano _____.sh:
  - The first line identifies the file as a **bash** script.
    #!/bin/bash
  - Comments begin with a **#** and end at the end of the line.
- give the user (and others, if (s)he wishes) permission to execute it.
  - chmod XXX filename
  - X user X group X public
  - 0 no permission
  - 1 execute
  - 2 write
  - 3(2+1) w+e
  - 4 read
  - 5(4+1) r+e
  - 6 (4+2) r+w
  - 7(4+2+1) all
- Run from local dir
  - ./filename

Variables

- Create a variable
  - Variablename=value (no spaces, no $)
  - read variablename (no $)
- Access a variable's value
  - $variablename
- Set a variable
  - Variablename=value (no spaces, no $ before variablename)

Read Write

- echo “message”
- read variable
- echo “message $variable”

The read Command (continued)

Read from stdin (screen)
Read until new line

Format	Meaning
`read answer`	Reads a line from `stdin` into the variable `answer`
`read first last`	Reads a line from `stdin` up to the whitespace, putting the first word in `first` and the rest of the of line into `last`
`read`	Reads a line from `stdin` and assigns it to `REPLY`
`read –a arrayname`	Reads a list of word into an array called `arrayname`
`read –p prompt`	Prints a prompt, waits for input and stores input in `REPLY`
`read –r line`	Allows the input to contain a backslash.

Arithmetic operators $((a+b)) or $(($a+$b))

Operator	Description	Example
+ (Addition)	Adds values on either side of the operator	`expr $a + $b` will give 30
- (Subtraction)	Subtracts right hand operand from left hand operand	`expr $a - $b` will give -10
* (Multiplication)	Multiplies values on either side of the operator	`expr $a \* $b` will give 200
/ (Division)	Divides left hand operand by right hand operand	`expr $b / $a` will give 2
% (Modulus)	Divides left hand operand by right hand operand and returns remainder	`expr $b % $a` will give 0
= (Assignment)	Assigns right operand in left operand	a = $b would assign value of b into a
== (Equality)	Compares two numbers, if both are same then returns true.	[$a == $b] would return false.
!= (Not Equality)	Compares two numbers, if both are different then returns true.	[$a != $b] would return true.

Relational Operators:

- ‘==’ 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

`test` Command Operators – Integer Tests

Test operator	Tests True if
`[ int1 -eq int2 ]`	int1 = int2
`[ int1 -ne int2 ]`	int1 ≠ int2
`[ int1 -gt int2 ]`	int1 > int2
`[ int1 -ge int2 ]`	int1 ≥ int2
`[ int1 -lt int2 ]`	int1 < int2
`[ int1 -le int2 ]`	int1 ≤ int2

Logical Operators :

- 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.

`test` Command Operators – Logical Tests

Test Operator	Test True If
`[ string1 -a string2 ]`	Both string1 and string 2 are true.
`[ string1 -o string2 ]`	Both string1 or string 2 are true.
`[ ! string ]`	Not a string1 match

Test operator	Tests True if
`[[ pattern1 && Pattern2 ]]`	Both pattern1 and pattern2 are true
`[[ pattern1 || Pattern2 ]]`	Either pattern1 or pattern2 is true
`[[ !pattern ]]`	Not a pattern match

pattern1 and *pattern2* can contain metacharacters.

Bitwise Operators:

- 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.

Positional Parameters

Positional Parameter	What It References
`$0`	References the name of the script
`$#`	Holds the value of the number of positional parameters
`$*`	Lists all of the positional parameters
`$@`	Means the same as $@, except when enclosed in double quotes
`"$*"`	Expands to a single argument (e.g., "`$1 $2 $3`")
`"$@"`	Expands to separate arguments (e.g., "`$1`" "`$2`" "`$3`")
`$1 .. ${10}`	References individual positional parameters
`set`	Command to reset the script arguments

- #!/bin/sh
- echo "File Name: $0"
- echo "First Parameter : $1"
- echo "Second Parameter : $2"
- echo "Quoted Values: $@"
- echo "Quoted Values: $*"
- echo "Total Number of Parameters : $#"

```
ubuntu@ubuntu:~$ ./commandline.sh amar try this new thing
File Name: ./commandline.sh
First Parameter : amar
Second Parameter : try
Quoted Values: amar try this new thing
Quoted Values: amar try this new thing
Total Number of Parameters : 5
```

`$* and $@`

- **`$*`** and **`$@`** can be used as part of the list in a for loop or can be used as par of it.
- When expanded **`$@`** and **`$*`** are the same unless enclosed in double quotes.
  - **`$*`** is evaluated to a single string while **`$@`** is evaluated to a list of separate word.

Conditional Statements

```
if test expression
then
     command
fi
```

```
read -p "enter a number:"  x
read -p "Enter Other number:" y
if [ $x > $y ]
then
echo $x is greater than $y
fi
if [ $x < $y ]
then
echo $x is less than $y
fi
if [ $x == $y ]
then
echo $x is equal to $y
fi
```

Conditional Statements

```
if [[ condition ]]
then
     statement
elif [[ condition ]]; then
     statement
else
     do this by default
fi
```

```
read -p "enter a number:"  x
read -p "Enter Other number:" y

if [ $x -gt $y ]
then
echo $x is greater than $y
elif [ $x -lt $y ]
then
echo $x is less than $y
elif [ $x -eq $y ]
then
echo $x is equal to $y
fi
```

Loops

- For loop
  - for <variable name> in <a list of items>
    do <some command> $<variable name>
    done

```
for i in {1..5}
do
   echo $i
done
```

```
for X in amar teach this
do
        echo $X
done
```

Loops

while command
do
  Statement(s) to be executed if command is true
done

```
!/bin/sh

a=0

while [ $a -lt 10 ]
do
  echo $a
  a=`expr $a + 1`
done
```

Loops

until command
do
command(s)
done

```
!/bin/sh

a=0

while [ $a -lt 10 ]
do
   echo $a
   a=`expr $a + 1`
done
```

The **`until`** Command

until works like the while command, except it execute the loop if the exit status is nonzero (i.e., the command failed).

- Format:

```
until command
do
    command(s)
done
```

```
#!/bin/sh

a=0

until [ ! $a -lt 10 ]
do
  echo $a
  a=`expr $a + 1`
done
```

Case

Syntax:

```
case variable
    value1 )
        commands
        ;;
    value2 )
        commands
        ;;
    *)  #default
        Commands
        ;;
esac
```

```
#!/bin/sh

FRUIT="kiwi"

case "$FRUIT" in
  "apple") echo "Apple pie is quite tasty."
  ;;
  "banana") echo "I like banana nut bread."
  ;;
  "kiwi") echo "New Zealand is famous for
kiwi."
  ;;
esac
```

`test` Command Operators – String Test

Test Operator	Tests True if
`[ string1 = string2 ]`	String1 is equal to String2 (space surrounding = is necessary
`[ string1 != string2 ]`	String1 is not equal to String2 (space surrounding != is not necessary
`[ string ]`	String is not null.
`[ -z string ]`	Length of string is zero.
`[ -n string ]`	Length of string is nonzero.
`[ -l string ]`	Length of string (number of character)

[[ ]] gives some pattern matching

[[$name == [Tt]om]] matches if $name contains Tom or tom

[[$name == [^t]om]] matches if $name contains any character but t followed by om

[[$name == ?o]] matches if $name contains any character followed by o and then whatever number of characters after that.*

Just shell patterns, not regex

Exit Status

- Every process running in Linux has an exit status code, where **0** indicates successful conclusion of the process and nonzero values indicates failure to terminate normally.
- Linux and UNIX provide ways of determining an exit status and to use it in shell programming.
- The **`?`** in **`bash`** is a shell variable that contains a numeric value representing the exit status.

Exit Status Demo

- All commands return something
- Standard 0 = success and 1 = failure
  - Backwards 0/1 from a true/false boolean

grep 'not there' myscript

echo $?

1= failure

grep 'a' myscript

echo $?

0 = success