UNIT 3: SHELL PROGRAMMING GETTING STARTED WITH LINUX, LINUX BOOTING

CS – 22: Operating Systems Concepts With Unix / Linux

- **Shell Keywords, Shell Variables and System variables**
- **PS2, PATH, HOME, LOGNAME, MAIL, IFS, SHELL, TERM, MAILCHECK**
- **User variables: set, unset and echo command with shell**
- **Variables, Positional Parameters**
- **Interactive shell script using read and echo**
- **Decision Statements if then fi, if then else fi, if then elif else fi, case esac**
- **Logical Operators**
- **Looping statements**
- **Arithmetic in Shell script**
- **History of Linux: GNU, GPL Concept, Open Source & Freeware**
- **Structure and Features of Linux: Installation and Configuration of Linux**
- **Using with Ubuntu: Startup, Shutdown and boot loaders of**
- **Linux Booting Process ,LILO Configuration, GRUB Configuration, User Interfaces (GUI and CUI)**

Assignment Questions

1. **Give the full form of LILO.**
2. **Which is the extension for shell-script file?**
3. **What is shell?**
4. **Explain boot loader.**
5. **Explain GRUB**
6. **Explain positional parameters.**
7. **Explain decision statements in unix.**
8. **Explain system variables.**
9. **Write a shell script to check the number is prime number or not.**
10. **Write a shell script for the following pattern.**

```
1
12
123
1234
12345
```

PREPARED BY: PROF. N.K.PANDYA (PHD*, MCA, BCA, BPP)
KAMANI SCIENCE COLLEGE, AMRELI(BCA/BSC)

Shell, Shell Keywords, Variable

A shell is special user program which provide an interface to user to use operating system services. Shell accepts 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.
Shell is broadly classified into two categories:

- Command Line Shell
- Graphical shell

Shell System Variables/Environment variables are variables that are available system-wide and are inherited by all spawned child processes and shells.

USER: The current logged in user.
HOME: The home directory of the current user.
EDITOR: The default file editor to be used. This is the editor that will be used when you type edit in your terminal.
SHELL: The path of the current user's shell, such as bash or zsh.
LOGNAME: The name of the current user.
PATH: A list of directories to be searched when executing commands. When you run a command, the system will search those directories in this order and use the first found executable.
LANG: The current locales settings.
TERM: The current terminal emulation.
MAIL: Location of where the current user's mail is stored.

Shell User Variables/Shell variables are variables that apply only to the current shell instance. Each shell such as zsh and bash, has its own set of internal shell variables.

env: The command allows you to run another program in a custom environment without modifying the current one. When used without an argument it will print a list of the current environment variables.
printenv: The command prints all or the specified environment variables.
set: The command sets or unsets shell variables. When used without an argument it will print a list of all variables including environment and shell variables, and shell functions.
unset: The command deletes shell and environment variables.
export: The command sets environment variables.

Positional parameters

A positional parameter is a variable within a shell program; its value is set from an argument specified on the command line that invokes the program. Positional parameters are numbered and are referred to with a preceding ``$'': $1, $2, $3, and so on.

Shell Variables

They are named places to store data, usually in the form of character strings, and their values can be obtained by preceding their names with dollar signs ($). Certain variables, called environment variables, are conventionally named in all capital letters, and their values are made known (with the export statement) to subprocesses.

If you are a programmer, you already know that just about every major programming language uses variables in some way; in fact, an important way of characterizing differences between languages is comparing their facilities for variables.

Interactive shell script using read and echo

read is a bash built-in command that reads the contents of a line into a variable. It allows for word splitting that is tied to the special shell variable IFS. It is primarily used for catching user input but can be used to implement functions taking input from standard input.

echo command in Linux is used to display line of text/string that are passed as an argument. This is a built-in command that is mostly used in shell scripts and batch files to output status text to the screen or a file.

Shell Scripting

Shell Scripting is an open-source computer program designed to be run by the Unix/Linux shell. Shell Scripting is a program to write a series of commands for the shell to execute. It can combine lengthy and repetitive sequences of commands into a single and simple script that can be stored and executed anytime which, reduces programming efforts.

Shell Scripts are written using text editors. On your Linux system, open a text editor program, open a new file to begin typing a shell script or shell programming, then give the shell permission to execute your shell script and put your script at the location from where the shell can find it.

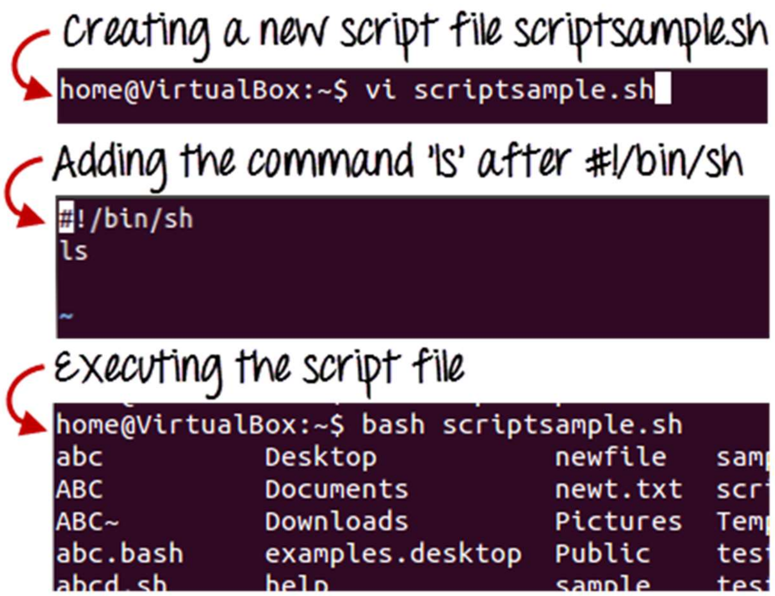
Steps to create shell script:

- **Step 1.** Create a file using a vi editor(or any other editor). Name script file with extension **.sh**
- **Step 2.** Start the script with **#! /bin/sh**
- **Step 3.** Write some code.(can be anything including linux/unix commands)
- **Step 4.** Save the script file as **<filename.sh**>
- **Step 5.** For executing the script type **bash <filename.sh>**

Note: "#!" is an operator called shebang which directs the script to the interpreter location. So, if we use" #! /bin/sh" the script gets directed to the bourne-shell.

Example:

```
#!/bin/sh
ls
```


Creating a new script file scriptsample.sh
Adding the command 'ls' after #!/bin/sh
Executing the script file

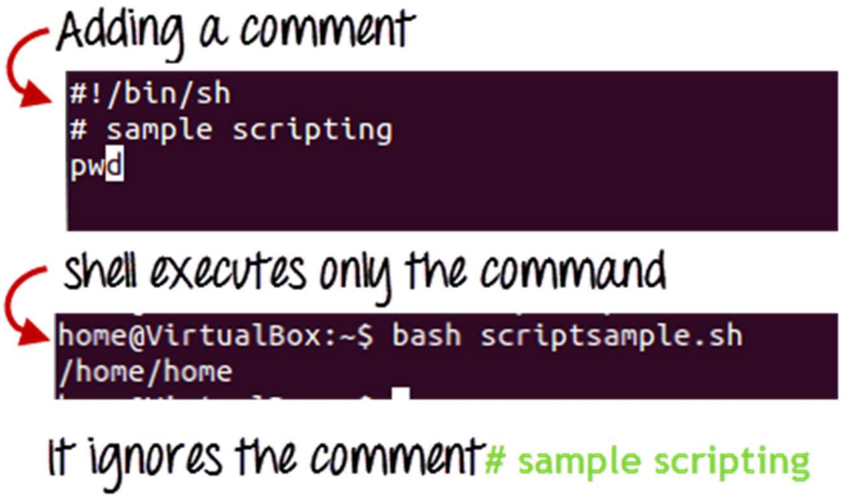
```
home@VirtualBox:~$ vi scriptsample.sh
```

```
#!/bin/sh
ls

~
```

```
home@VirtualBox:~$ bash scriptsample.sh
abc           Desktop             newfile    sam
ABC           Documents           newt.txt   scr
ABC~          Downloads           Pictures   Tem
abc.bash      examples.desktop    Public     tes
abcd.sh       help                sample     tes
```

Adding shell comments
Commenting is important in any program. In Shell programming, the syntax to add a comment is
#comment

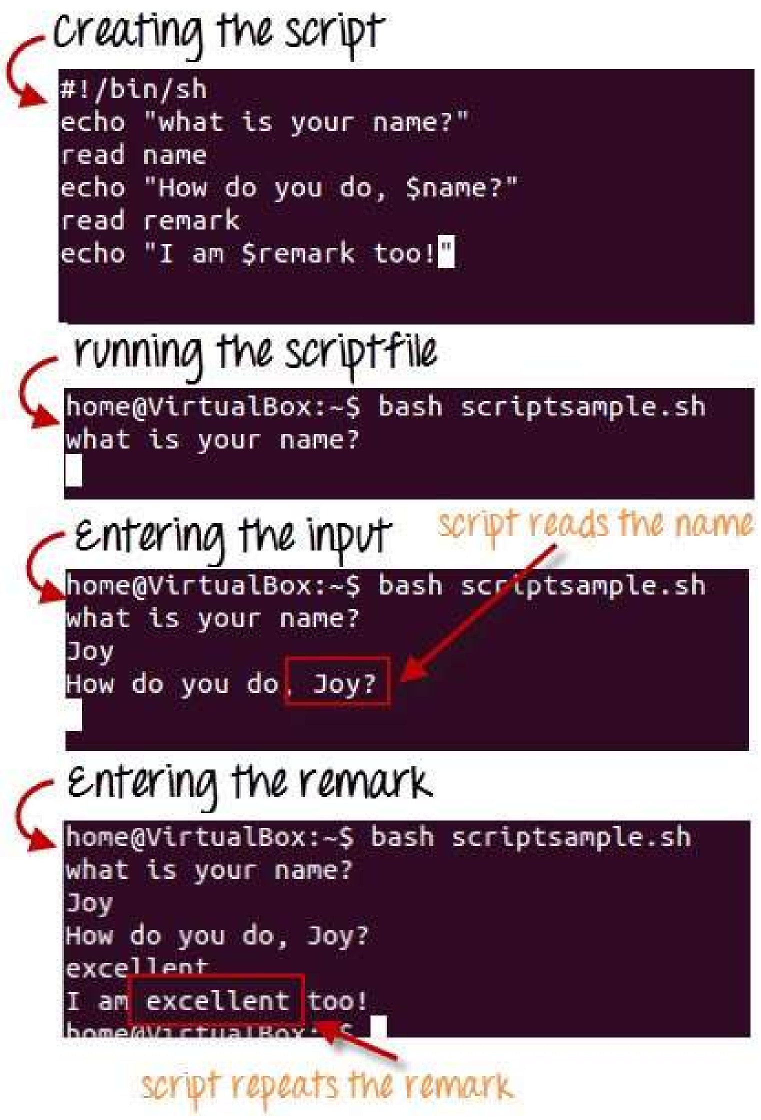


```
#!/bin/sh
# sample scripting
pwd
```

```
home@VirtualBox:~$ bash scriptsample.sh
/home/home
```

sample scripting

Shell Variables
As discussed earlier, Variables store data in the form of characters and numbers. Similarly, Shell variables are used to store information and they can by the shell only.
Example:
variable ="Hello"
echo $variable



```
#!/bin/sh
echo "what is your name?"
read name
echo "How do you do, $name?"
read remark
echo "I am $remark too!"
```

```
home@VirtualBox:~$ bash scriptsample.sh
what is your name?
```

```
home@VirtualBox:~$ bash scriptsample.sh
what is your name?
Joy
How do you do, Joy?
```

```
home@VirtualBox:~$ bash scriptsample.sh
what is your name?
Joy
How do you do, Joy?
excellent
I am excellent too!
home@VirtualBox:~$
```

- Kernel is the nucleus of the operating systems, and it communicates between hardware and software
- Shell is a program which interprets user commands through CLI like Terminal
- The Bourne shell and the C shell are the most used shells in Linux
- Linux Shell scripting is writing a series of command for the shell to execute
- Shell variables store the value of a string or a number for the shell to read
- Shell scripting in Linux can help you create complex programs containing conditional statements, loops, and functions
- Basic Shell Scripting Commands in Linux: cat, more, less, head, tail, mkdir, cp, mv, rm, touch, grep, sort, wc, cut and, more.

Decision Statements if then fi, if then else fi, if then elif else fi, case esac

Decision-making is the act of performing different actions based on different conditions.The following are the two types of decision-making statements we can use in Unix/Linux shell:

The if-else statement: If-else statements are useful to take a path from a given set of paths based on the given conditions.

- **if-fi**: The if block is executed if the specified condition is true.

Syntax

```
if [ expression ]
then
  Statement(s) to be executed if expression is true
fi
```

Example

```
#!/bin/sh
a=10
b=20
if [ $a == $b ]
then
  echo "a is equal to b"
fi
if [ $a != $b ]
then
  echo "a is not equal to b"
fi
```

- **if-else-fi**: The if block is executed if the specified condition is true. Otherwise, the else block is executed.

  Syntax:

  ```
  if [ expression ]
  then
    Statement(s) to be executed if expression is true
  else
    Statement(s) to be executed if expression is not true
  fi
  ```

  Example:

  ```
  #!/bin/sh
  a=10
  b=20
  if [ $a == $b ]
  then
    echo "a is equal to b"
  else
    echo "a is not equal to b"
  fi
  ```

- **if-elif-else-fi**: The if block is executed if the condition specified in the if statement is true. Otherwise, the next elif condition is checked. The elif block is executed if the condition specified in the elif statement is true. The process goes on for the remaining elif statements until no condition is satisfied. If none of the conditions of if or elif statements are satisfied, the else block is executed.

Syntax:

```
if [ expression 1 ]
then
  Statement(s) to be executed if expression 1 is true
elif [ expression 2 ]
then
  Statement(s) to be executed if expression 2 is true
elif [ expression 3 ]
then
  Statement(s) to be executed if expression 3 is true
else
  Statement(s) to be executed if no expression is true
fi
```

Example:

```
#!/bin/sh
a=10
b=20
if [ $a == $b ]
then
  echo "a is equal to b"
elif [ $a -gt $b ]
then
  echo "a is greater than b"
elif [ $a -lt $b ]
then
  echo "a is less than b"
else
  echo "None of the condition met"
fi
```

The case-sac statement: Case-sac statements are similar to switches. They are useful when all of the different branches only depend on the value of a single input variable.

Syntax

The basic syntax of the case...esac statement is to give an expression to evaluate and to execute several different statements based on the value of the expression.

The interpreter checks each case against the value of the expression until a match is found. If nothing matches, a default condition will be used.

```
case word in
  pattern1)
    Statement(s) to be executed if pattern1 matches ;;
  pattern2)
    Statement(s) to be executed if pattern2 matches ;;
  pattern3)
    Statement(s) to be executed if pattern3 matches;;
  *)
   Default condition to be executed;;
esac
```

Example:

```
#!/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
```

Operators

Arithmetic Operators

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	Description	Example
-eq	Checks if the value of two operands are equal or not; if yes, then the condition becomes true.	[$a -eq $b] is not true.
-ne	Checks if the value of two operands are equal or not; if values are not equal, then the condition becomes true.	[$a -ne $b] is true.
-gt	Checks if the value of left operand is greater than the value of right operand; if yes, then the condition becomes true.	[$a -gt $b] is not true.
-lt	Checks if the value of left operand is less than the value of right operand; if yes, then the condition becomes true.	[$a -lt $b] is true.
-ge	Checks if the value of left operand is greater than or equal to the value of right operand; if yes, then the condition becomes true.	[$a -ge $b] is not true.
-le	Checks if the value of left operand is less than or equal to the value of right operand; if yes, then the condition becomes true.	[$a -le $b] is true.

Boolean Operators

Operator	Description	Example
!	This is logical negation. This inverts a true condition into false and vice versa.	[! false] is true.
-o	This is logical **OR**. If one of the operands is true, then the condition becomes true.	[$a -lt 20 -o $b -gt 100] is true.
-a	This is logical **AND**. If both the operands are true, then the condition becomes true otherwise false.	[$a -lt 20 -a $b -gt 100] is false.

Looping statements

A loop is a powerful programming tool that enables you to execute a set of commands repeatedly. You will use different loops based on the situation. For example, the while loop executes the given commands until the given condition remains true; the until loop executes until a given condition becomes true.

While Loop

The while loop enables you to execute a set of commands repeatedly until some condition occurs. It is usually used when you need to manipulate the value of a variable repeatedly.

Syntax

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

Example

```
#!/bin/sh
a=0
while [ $a -lt 10 ]
do
  echo $a
  a=`expr $a + 1`
done
```

For loop

The for loop operates on lists of items. It repeats a set of commands for every item in a list.

Syntax

```
for var in word1 word2 ... wordN
do
  Statement(s) to be executed for every word.
done
```

Here var is the name of a variable and word1 to wordN are sequences of characters separated by spaces (words). Each time the for loop executes, the value of the variable var is set to the next word in the list of words, word1 to wordN.

Example:

```
#!/bin/sh
for var in 0 1 2 3 4 5 6 7 8 9
do
  echo $var
done
```

until Loop

The while loop is perfect for a situation where you need to execute a set of commands while some condition is true. Sometimes you need to execute a set of commands until a condition is true.

Syntax

```
until command
do
  Statement(s) to be executed until command is true
done
```

Here the Shell command is evaluated. If the resulting value is false, given statement(s) are executed. If the command is true then no statement will be executed and the program jumps to the next line after the done statement.

Example

```
#!/bin/sh
a=0
until [ ! $a -lt 10 ]
do
   echo $a
   a=`expr $a + 1`
done
```

Arithmetic in Shell script

```
a=10
b=20
val=`expr $a + $b`
echo "a + b : $val"
val=`expr $a - $b`
echo "a - b : $val"
val=`expr $a \* $b`
echo "a * b : $val"
val=`expr $b / $a`
echo "b / a : $val"
val=`expr $b % $a`
echo "b % a : $val"
if [ $a == $b ]
then
   echo "a is equal to b"
fi
if [ $a != $b ]
then
   echo "a is not equal to b"
fi
```

History 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 as he wanted to use Unix 386 Intel computer but couldn't afford it. 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 use 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.

Linux Today

Today, supercomputers, smart phones, desktop, web servers, tablet, laptops and home appliances like washing machines, DVD players, routers, modems, cars, refrigerators, etc use Linux OS.

GNU

- GNU is a Unix-compatible operating system developed by the GNU project, which was started in 1983 by Richard Stallman with the goal of producing non-proprietary software. As such, users may download, modify and redistribute GNU software. GNU is a recursive acronym for GNU's Not Unix!
- GNU has a Unix-like design, but it is available as free software and does not contain any Unix code. GNU is comprised of a collection of software applications, libraries and developer tools, along with a program to allocate resources and communicate with the hardware, or kernel. GNU can be used with other kernels and is often used with a Linux kernel. A GNU/Linux combination is the GNU/Linux operating system. The main components of a GNU system include the following:
  - GNU compiler collection
  - GNU C library
  - GNU Emacs text editor
  - GNOME desktop environment
- GNU programs can be ported to a multitude of other operating systems, including different platforms such as Mac OS X and Microsoft Windows. GNU is sometimes installed on Unix systems as a replacement for proprietary utilities.

GPL Concept

The GNU General Public License (GPL) is a free, copyleft license used primarily for software. The GNU GPL allows users to change and share all versions of a program. GPL is provided through the Free Software Foundation, a non-profit corporation that works to provide free software for the GNU Project. In 1989, Richard Stallman produced the first GPL through the GNU Program. The GNU Program was launched in 1984 for the express purpose of developing operating systems that are similar to Unix, except that they are open source. Under the GPL provisions, owners may sell copies of programs under GPL, or distribute them for free. To do so, licensees must adhere to the designated terms and conditions of the GPLs. Under a GPL, owners are permitted to modify digital materials as well. The GPL is widely used and the most popular free license of its kind.

Open Source

Open source is a term that originally referred to open source software (OSS). Open source software is code that is designed to be publicly accessible—anyone can see, modify, and distribute the code as they see fit.

Open source software is developed in a decentralized and collaborative way, relying on peer review and community production. Open source software is often cheaper, more flexible, and has more longevity than its proprietary peers because it is developed by communities rather than a single author or company.

Open source has become a movement and a way of working that reaches beyond software production. The open source movement uses the values and decentralized production model of open source software to find new ways to solve problems in their communities and industries.

Freeware

Freeware is any copyrighted software, application or program that may be freely downloaded, installed, used and shared. Such programs are available for use at no cost to general end users. Freeware differs from free software, as the latter allows a user to modify source code for republishing or integration with other software.

As small software utilities, freeware is free to use throughout its lifetime because it does not expire after a certain period. Freeware may be used for a desktop, mobile or Web-based utility.

Generally, freeware is a concise and limited version of a larger and paid software program. Vendors publish freeware to provide a limited but free version of software to prospective buyers prior to purchase. Additionally, large independent software vendors (ISV) publish freeware to enhance brand buzz and market reputation

Installation and Configuration of Linux

Step 1. Download the Linux distribution of your choice. If you're new to Linux, consider trying a lightweight and easy to use distribution, such as Ubuntu or Linux Mint. Linux distributions (known as "distros") are typically available for free to download in ISO format. You can find the ISO for the distribution of your choice at the distribution's website. This format needs to be burned to a CD or USB stick before you can use it to install Linux. This will create a Live CD or Live USB.

- **a.** A Live CD or Live USB is a disk that you can boot into, and often contains a preview version of the operating system that can be run directly from the CD or USB stick.
- **b.** Install an image burning program, or use your system's built-in burning tool if you are using Windows 7, 8, or Mac OS X. Pen Drive Linux and UNetBootin are two popular tools for burning ISO files to USB sticks.

Step 2. Boot into the Live CD or Live USB. Most computers are set to boot into the hard drive first, which means you will need to change some settings to boot from your newly-burned CD or USB. Start by rebooting the computer.

- **a.** Once the computer reboots, press the key used to enter the boot menu. The key for your system will be displayed on the same screen as the manufacturer's logo. Typical keys include F12, F2, or Del.
- **b.** For Windows 8 users, hold the Shift key and click restart. This will load the Advanced Startup Options, where you can boot from CD.
- **c.** For Windows 10 users, go to advanced boot in settings and click "Restart Now."
- **d.** If your computer doesn't give you direct access to the boot menu from the manufacturer's splash screen, it's most likely hidden in the BIOS menu. You can access the BIOS menu in the same way that you would get to the boot menu. At the manufacturer splash screen, the key should be listed in one of the bottom corners.
- **e.** Once you're in the boot menu, select your live CD or USB. Once you've changed the settings, save and exit the BIOS setup or boot menu. Your computer will continue with the boot process.

Step 3. Try out the Linux distribution before installing. Most Live CDs and USBs can launch a "live environment", giving you the ability to test it out before making the switch. You won't be able to create files, but you can navigate around the interface and decide if it's right for you.

Step 4. Start the installation process. If you're trying out the distro, you can launch the installation from the application on the desktop. If you decided not to try out the distribution, you can start the installation from the boot menu.

- **a.** You will be asked to configure some basic options, such as language, keyboard layout, and timezone.

Step 5. Create a username and password. You will need to create login information to install Linux. A password will be required to log into your account and perform administrative tasks.

Step 6. Set up the partition. Linux needs to be installed on a separate partition from any other operating systems on your computer if you intend dual booting Linux with another OS. A partition is a portion of the hard drive that is formatted specifically for that operating system. You can skip this step if you don't plan on dual booting.

- **a.** Distros such as Ubuntu will set a recommended partition automatically. You can then adjust this manually yourself. Most Linux installations require at least 20 GB, so be sure to set aside enough room for both the Linux operating system and any other programs you may install and files you may create.

b. If the installation process does not give you automatic partitions, make sure that the partition you create is formatted as Ext4. If the copy of Linux you are installing is the only operating system on the computer, you will most likely have to manually set your partition size.

Step 7. Boot into Linux. Once the installation is finished, your computer will reboot. You will see a new screen when your computer boots up called "GNU GRUB". This is a boot loader that handles Linux installations. Pick your new Linux distro from the list. This screen may not show up if you only have one operating system on your computer. If this screen isn't being presented to you automatically, then you can get it back by hitting shift right after the manufacturer splash screen.

Step 8. Start using Linux. Once your installation is complete and you've verified that your hardware is working, you're ready to start using Linux. Most distros come with several popular programs installed, and you can download many more from their respective file repositories.

Using Ubuntu: Start-up, Shutdown

Ubuntu was initially a terminal-based operating system, but over time Linux slowly introduced the GUI in its operating system. Nowadays, GUI methods are the only way to solve many operating system problems. Well, we are here to teach you some cool methods that can help you shut down your system in Ubuntu like a pro. There are two ways to shutdown Ubuntu Linux. Go to the upper right corner and click the drop-down menu. You'll see the shutdown button here. You can also use the command 'shutdown now'.

LILO Configuration

LILO stands for Linux Loader that is used to load Linux into memory. It can boot operating systems from floppy disks, hard disks, and it does not depend on a specific file system. Lilo handles some tasks such as locate the kernel, identify other supporting programs, load memory and starts the kernel. The configuration file of lilo is located at "/etc/lilo.conf". Lilo reads this configuration file and it tells Lilo where it should place the bootloader.

Linux Booting Process

When LILO loads itself, it displays the name LIO where each word specifies some actions
1) If it displays nothing then it does not load any part of LILO.
2) L: This is the first stage of the bootloader that has been loaded. If the process stops here it denotes that there were problems in the second stage. This may occur due to some incorrect disk parameter specified in the configuration file of lilo or some media problems also.
3) LI: It indicates that the second stage boot loader has been loaded and could not be executed. It can occur due to problems similar to L.
4) LIL: At this stage, the second stage boot loader has been completed in its execution. If it fails, this stage indicates that there were media problems or map file specified in the configuration file has some problems.
5) LIL?: This means that the second stage boot loader loaded at an incorrect address.
6) LIL-: This indicates that the descriptor table is corrupted.
7) LILO: All parts are successfully loaded.

GRUB Configuration

The GRUB (Grand Unified Bootloader) is a bootloader available from the GNU project. A bootloader is very important as it is impossible to start an operating system without it. It is the first program which starts when the program is switched on. The bootloader transfers the control to the operating system kernel.

GRUB Boot Process

The boot process using GRUB requires the GRUB to load itself into memory. This is done in the following steps:

- The stage 1 boot loader is loaded into the memory by the BIOS. This boot loader is also known as the primary boot loader. It exists on 512 bytes or less of disk space within the master boot record. The primary boot loader can load the stage 1.5 or stage 2 boot loader if required.
- The stage 1.5 boot loader is loaded into the memory by the stage 1 boot loader if required. This may be necessary in some cases as some hardware require a middle step before moving on to the stage 2 loader.
- The secondary boot loader is also known as the stage 2 boot loader and it can be loaded into the memory by the primary boot loader. Display of the GRUB menu and command environment are functions performed by the secondary boot loader. This allows the user to look at system parameters and select the operating system to boot.
- The operating system or kernel is loaded into the memory by the secondary boot loader. After that, the control of the machine is transferred to the operating system.

GRUB VS LILO

1) Lilo stands for Linux Loader and GRUB stands for Grand Unified Bootloader.
2) LILO has no interactive command interface, whereas GRUB has a more powerful command interface.
3) LILO does not support booting from a network, whereas GRUB does.
4) GRUB is a boot loader which can be used in Linux, DOS, and other operating systems whereas Lilo is a boot loader for Linux.
5) Lilo has a simpler interface so it is easy to use whereas GRUB is more complicated to use.
6) Lilo is the old default bootloader and GRUB is the new default boot loader.

User Interfaces (GUI and CUI)

Features	CUI	GUI
Full-Form	CUI stands for Character User Interface.	GUI stands for Graphical User Interface.
Interaction	The user interacts with the computer using commands like text.	The user interacts with the system using Graphics like icons, images.
Navigation	Navigation is not easy.	Navigation is easy to use.
Usage	Usage is easy to use.	Usage is difficult, requires expertise.
Speed	It has high speed.	It has a low speed.
Memory Requirement	It has a low memory requirement.	It has a high memory requirement.
Peripherals used	Users interact with the computer system by typing commands into the keyboard.	Users interact with the computer system using a graphical interface, which includes menus and mouse clicks.
Precision	It has high precision.	It has low precision.
Flexibility	It has a little flexible user interface.	It has a highly flexible user interface.
Customize	It is not easily changeable.	It has highly customizable.