JAZAN UNIVERSITY

COMPUTER SCIENCE DEPARTMENT

Operating Systems

(231 COMP- 3) & (COMP-333)
Lab Manual

RED Hat System Admistration-1

Course Coordinator
Syed Ziauddin

Academic Year 2022-2023

8	**Deadlock (Mutual Exclusion)**	154
9	**Page Replacement Algorithm** a. FIFO (First in First Out) **b.** ORA (Optimal Replacement Algorithm)	156

Tentative Breakdown of Lab Sessions

Topics	No of Weeks	Contact Hours
PART -1 **Installation of Red HAT Linux operating system**	1	2
Accessing the command line	1	2
Managing Files From The Command Line, Creating, viewing, and Editing text files	1	2
Managing local linux users and Groups	1	2
Controlling access to files with Linux file system permissions	1	2
Monitoring and managing Linux processes	1	2
PART -2 **CPU Scheduling**	2	4
Mutual Exclusion (Deadlock)	1	2
Page Replacement Algorithm	1	2

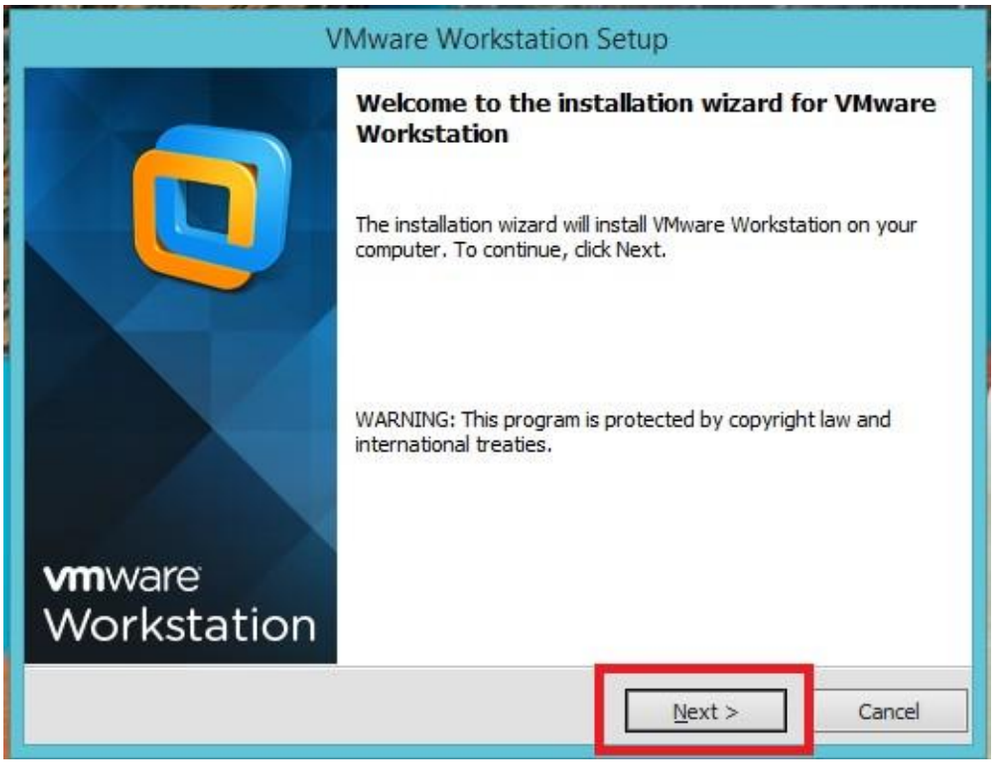
LAB-1 INSTALLATION OF RED HAT LINUX OPERATING SYSTEM

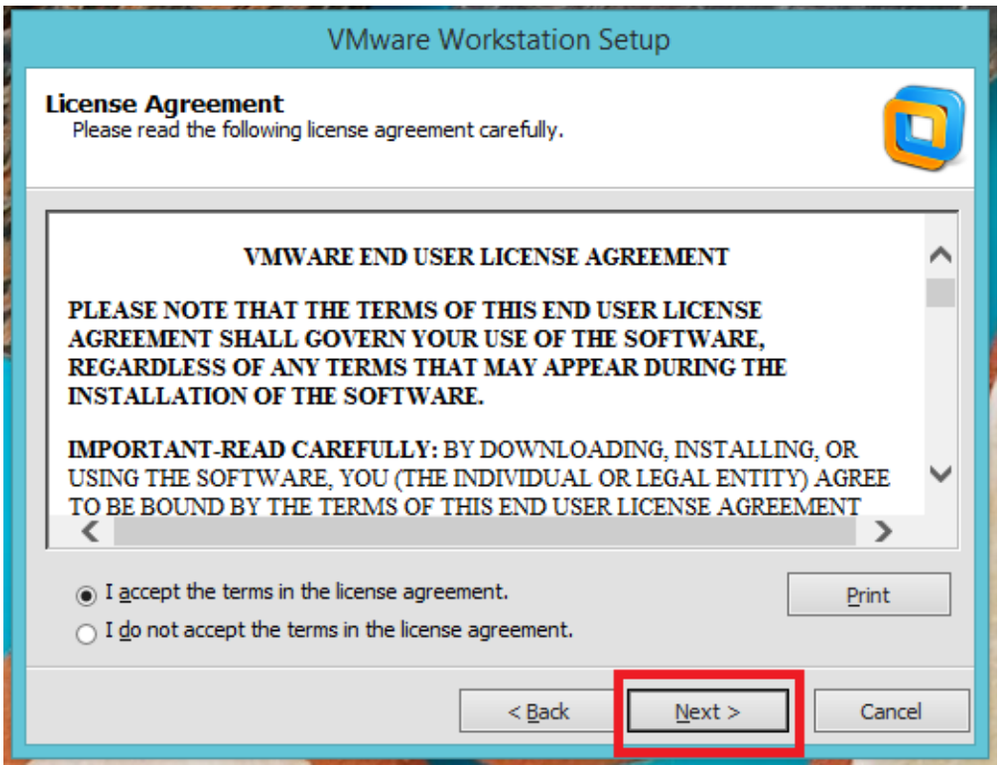
Goal- Installation of Red HAT Linux operating system.

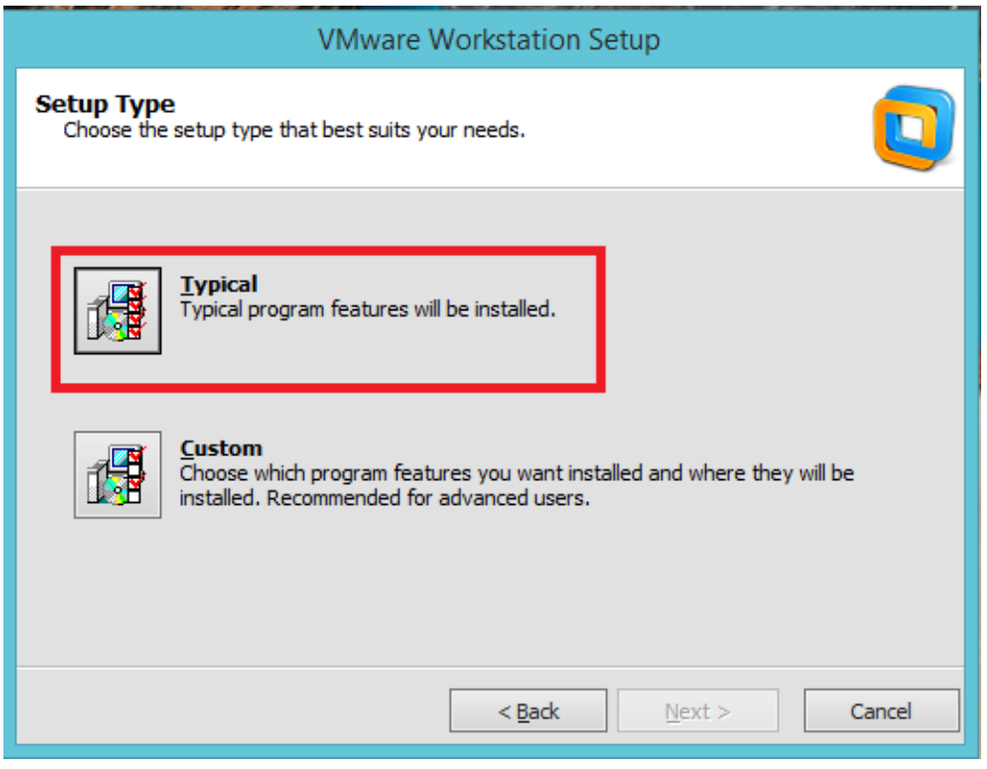
Sections

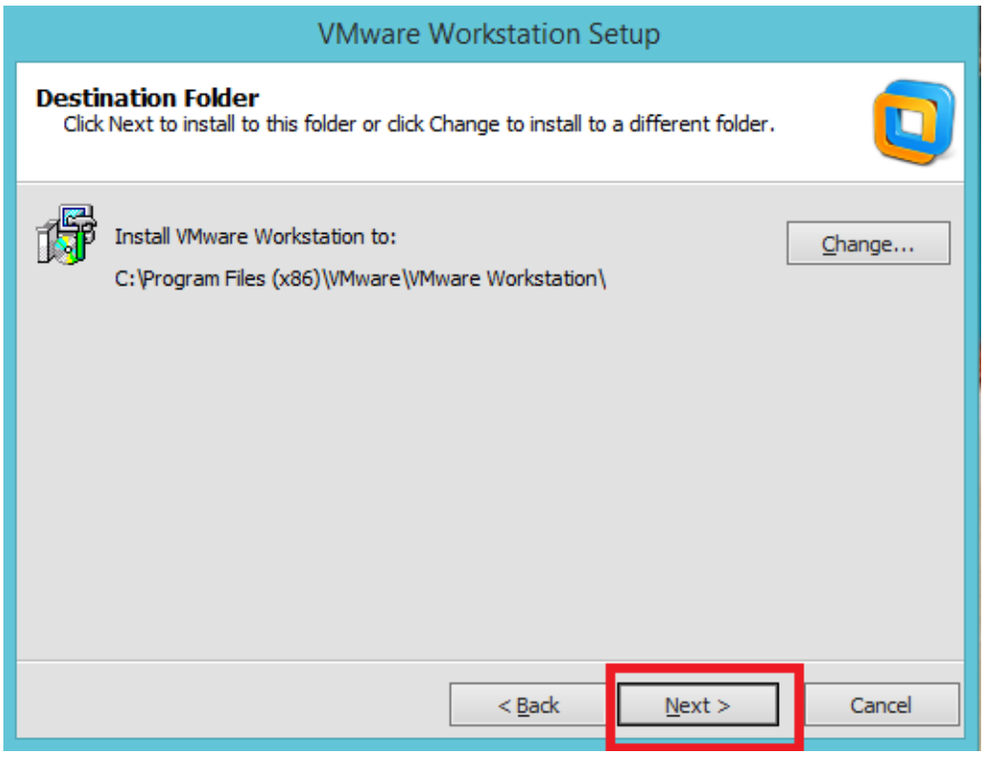
- Partitioning drives
- Configuring boot loader (GRUB/LILO)
- Network configuration
- Setting time zones
- Creating password and user accounts
- Shutting down

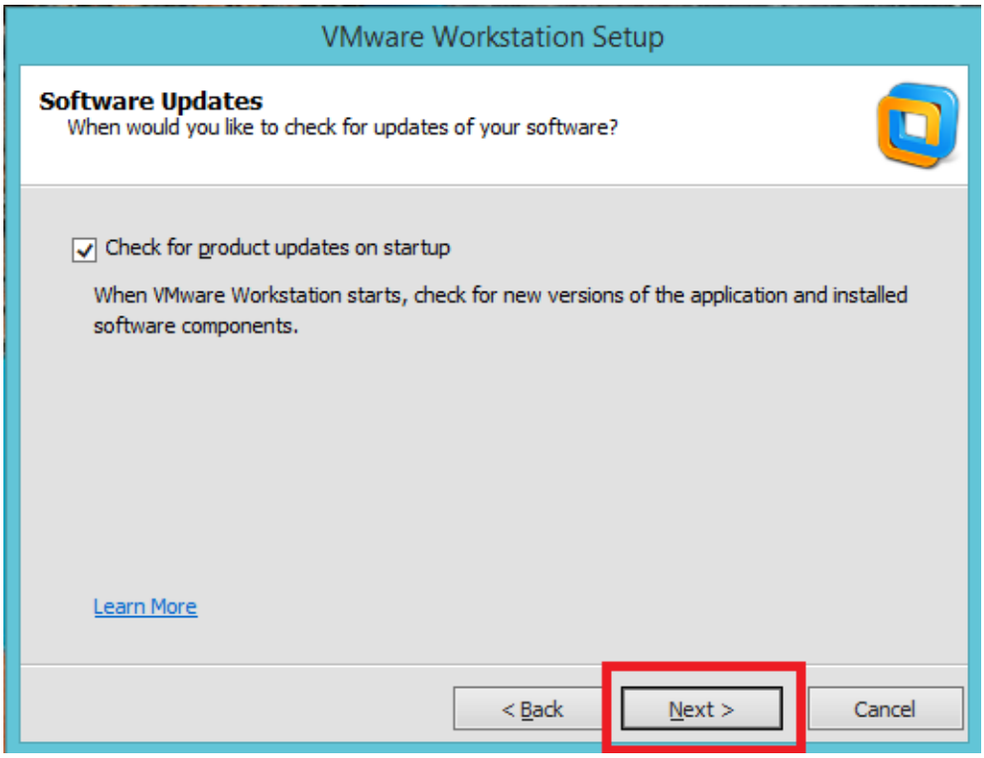
Practical no 1: Installation of Red HAT Linux operating system.

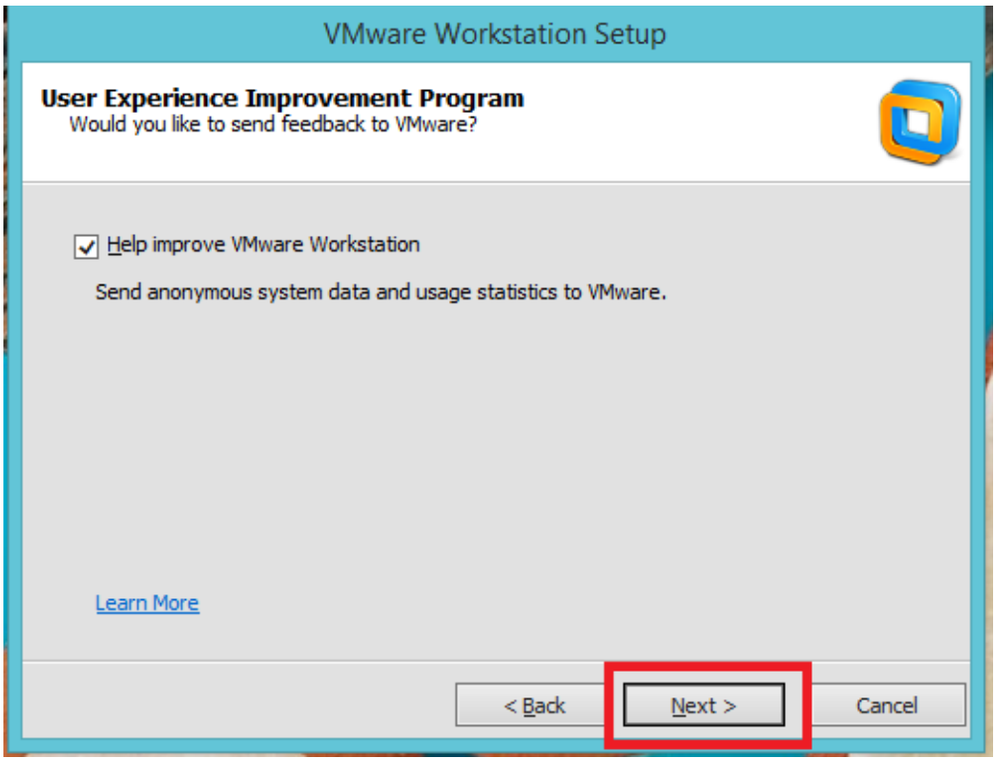


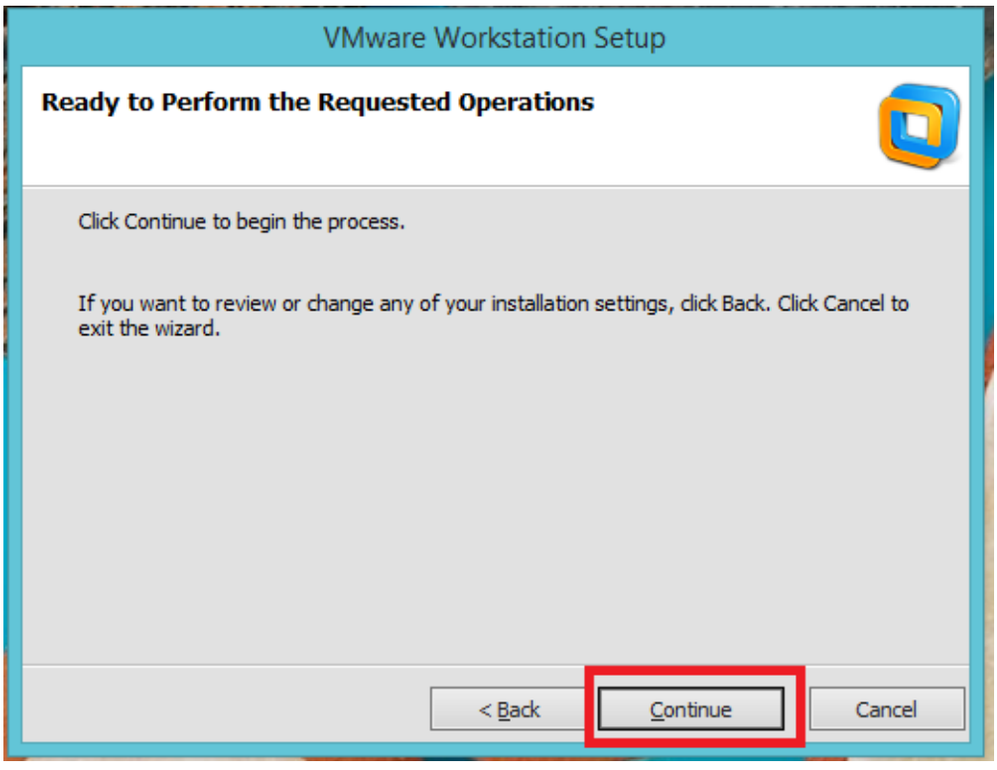

VMware Workstation Setup
License Agreement
Please read the following license agreement carefully.
VMWARE END USER LICENSE AGREEMENT
PLEASE NOTE THAT THE TERMS OF THIS END USER LICENSE AGREEMENT SHALL GOVERN YOUR USE OF THE SOFTWARE, REGARDLESS OF ANY TERMS THAT MAY APPEAR DURING THE INSTALLATION OF THE SOFTWARE.
IMPORTANT-READ CAREFULLY: BY DOWNLOADING, INSTALLING, OR USING THE SOFTWARE, YOU (THE INDIVIDUAL OR LEGAL ENTITY) AGREE TO BE BOUND BY THE TERMS OF THIS END USER LICENSE AGREEMENT
I accept the terms in the license agreement.
I do not accept the terms in the license agreement.
Print
< Back
Next >
Cancel

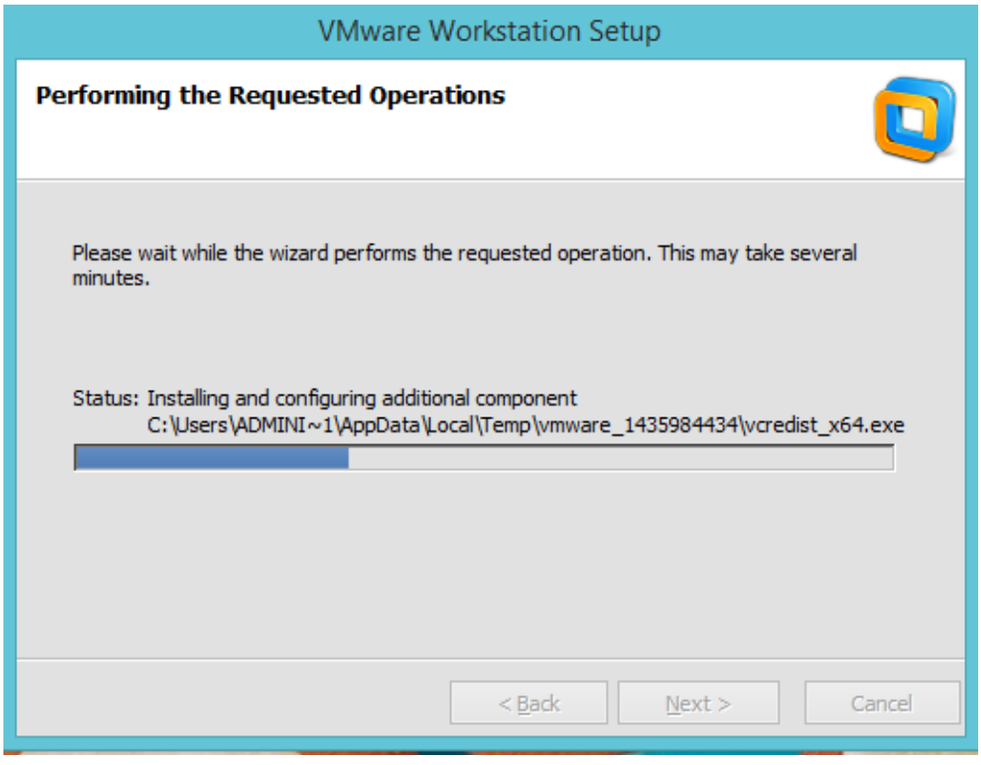

VMware Workstation Setup
Setup Type
Choose the setup type that best suits your needs.
Typical
Typical program features will be installed.
Custom
Choose which program features you want installed and where they will be installed. Recommended for advanced users.
< Back
Next >
Cancel


VMware Workstation Setup
Destination Folder
Click Next to install to this folder or click Change to install to a different folder.
Install VMware Workstation to:
C:\Program Files (x86)\VMware\VMware Workstation\
Change...
< Back
Next >
Cancel

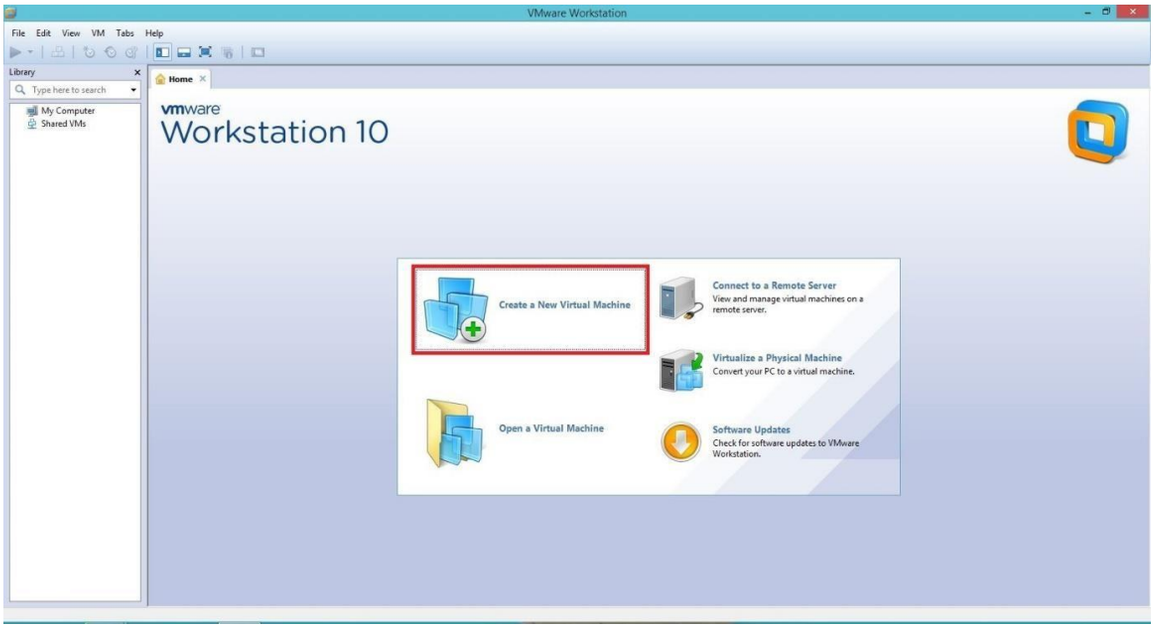

VMware Workstation Setup
Software Updates
When would you like to check for updates of your software?
Check for product updates on startup
When VMware Workstation starts, check for new versions of the application and installed software components.
Learn More
< Back
Next >
Cancel


VMware Workstation Setup
User Experience Improvement Program
Would you like to send feedback to VMware?
Help improve VMware Workstation
Send anonymous system data and usage statistics to VMware.
Learn More
< Back
Next >
Cancel

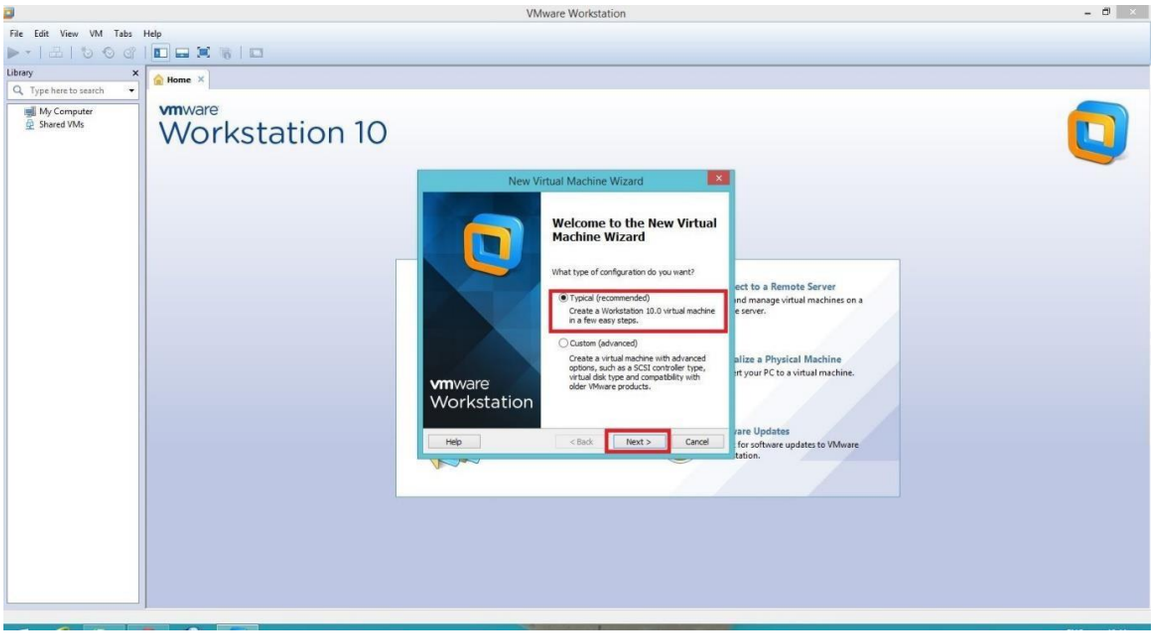

VMware Workstation Setup
Ready to Perform the Requested Operations
Click Continue to begin the process.
If you want to review or change any of your installation settings, click Back. Click Cancel to exit the wizard.
< Back
Continue
Cancel



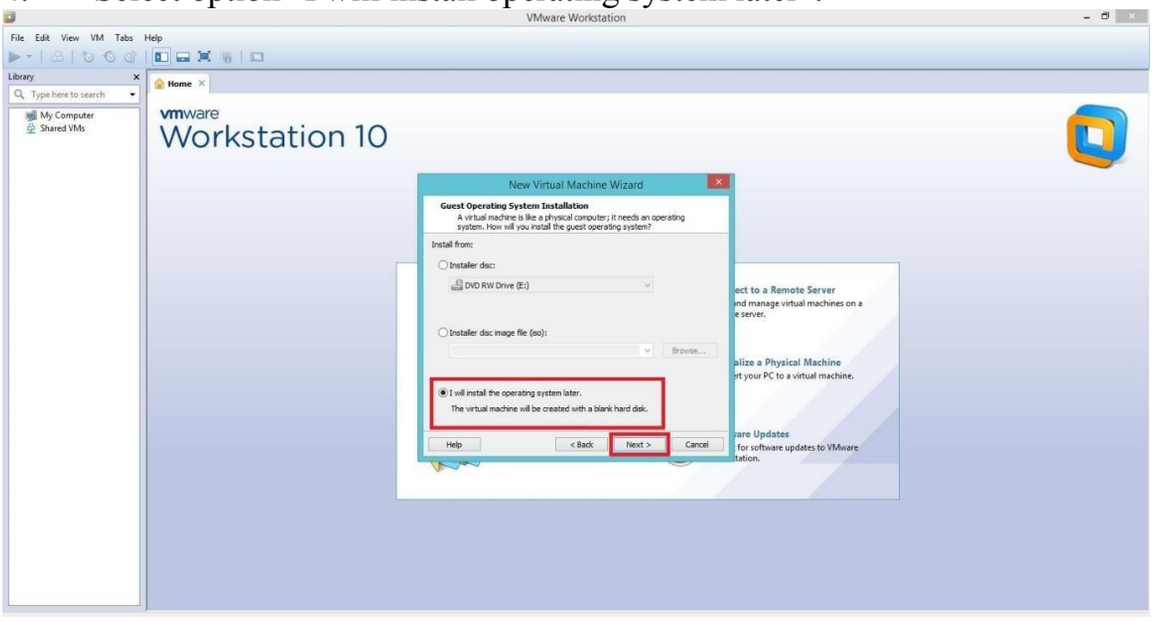
1. Double click on VM VirtualBox icon and Oracle VM VirtualBox Manager will open.
2. Click on New button in the toolbar to create a new virtual machine



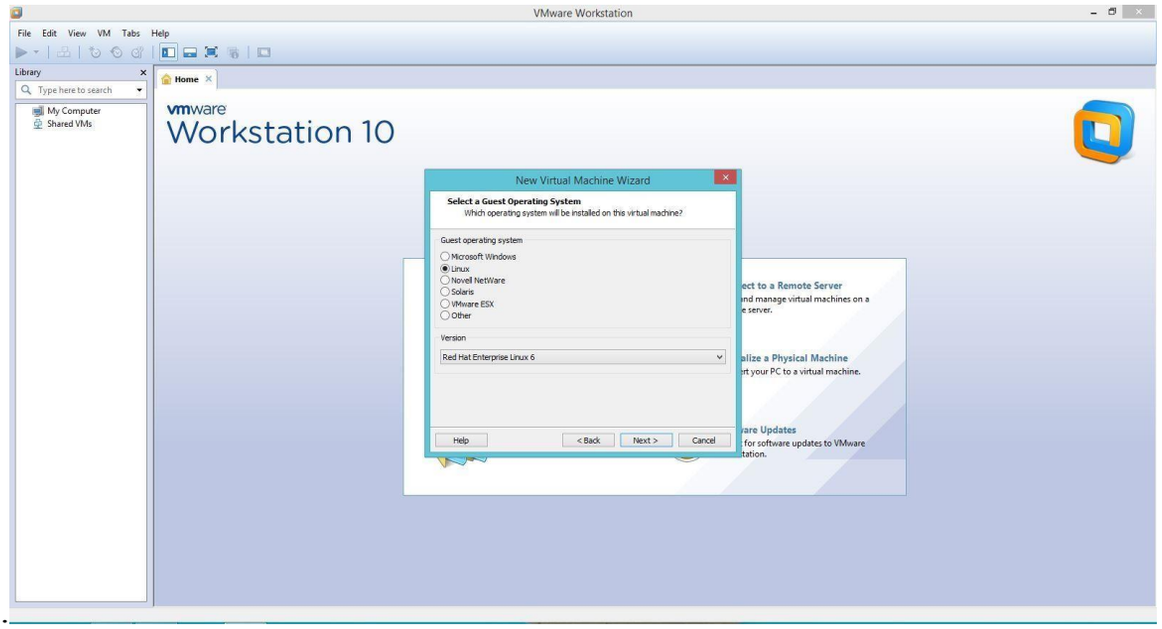
3. Create Virtual Machine Dialog box will open



4. Select option “I will install operating system later”.



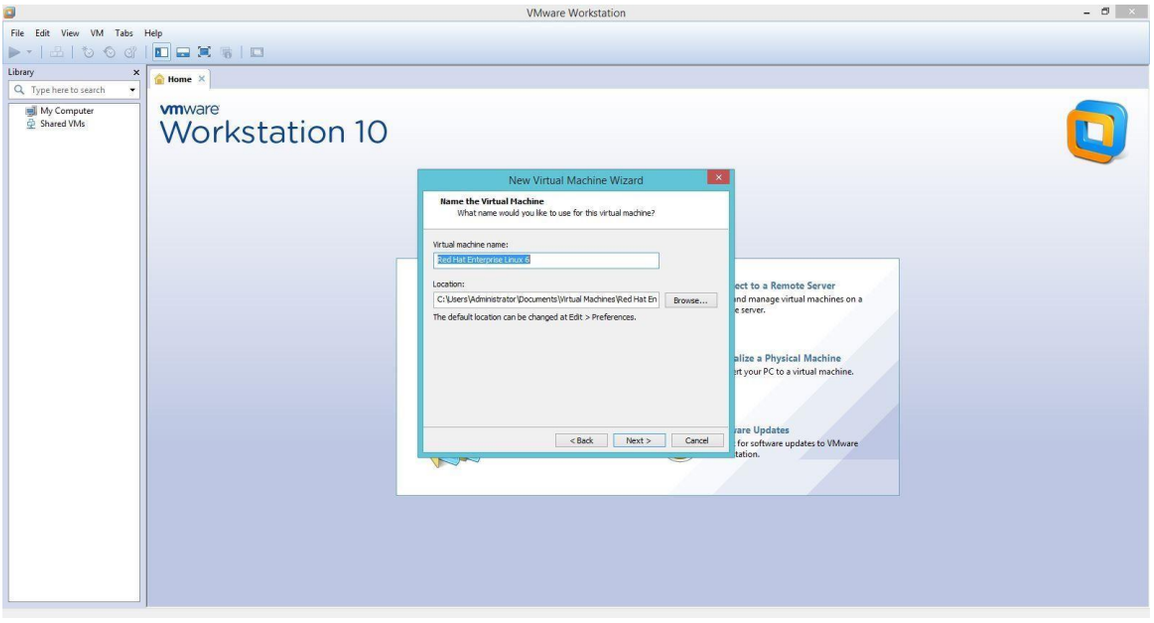
5. Select the operating system as RedHat the Type will automatically get selected as Linux and Version as Red hat



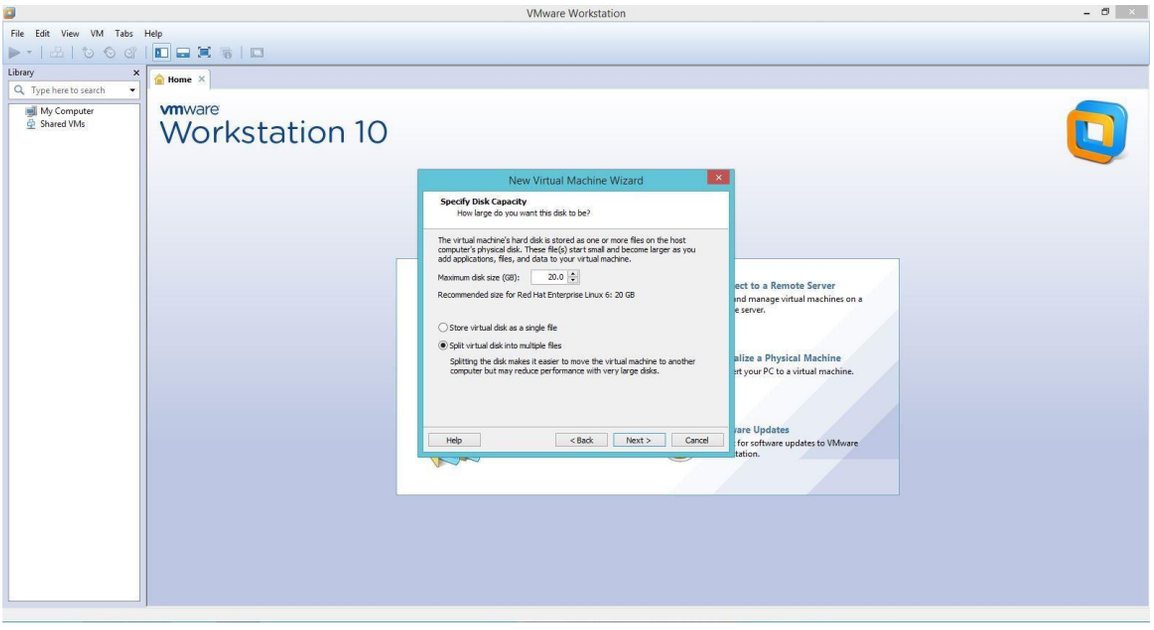
.

(Virtual Box support no of operating system which you can select from, Type drop down menu)

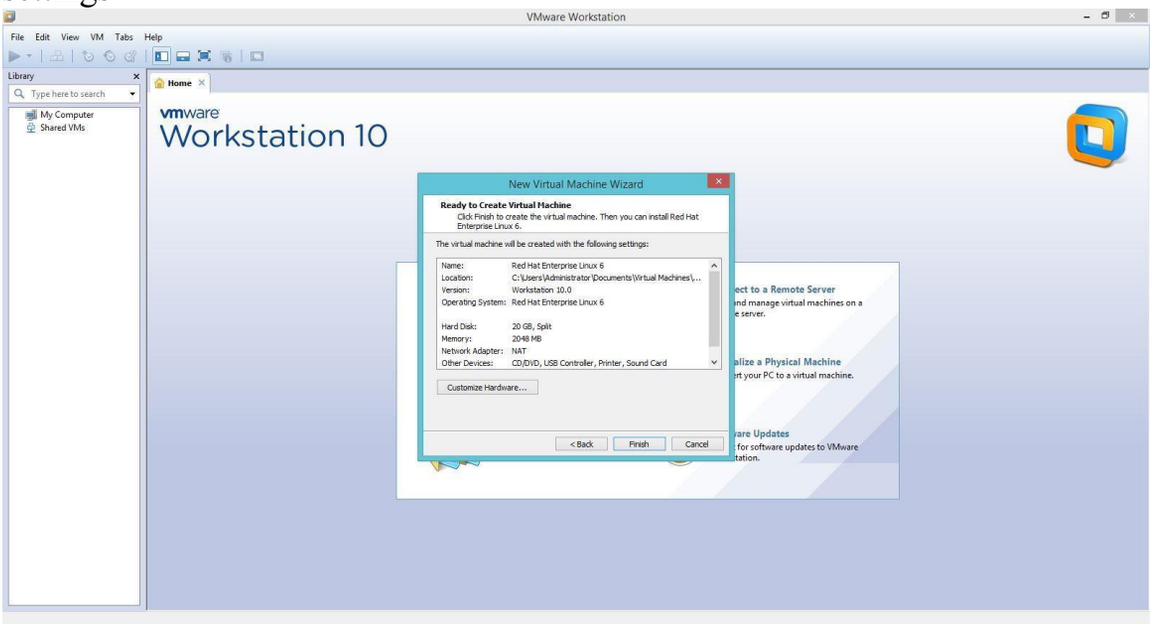
6. Now write the virtual machine name as you want or set it by default “Red Hat Enterprise Linux 6”.



7. Now Select the Hard disk space as 20GB and select Store virtual machine as single machine.



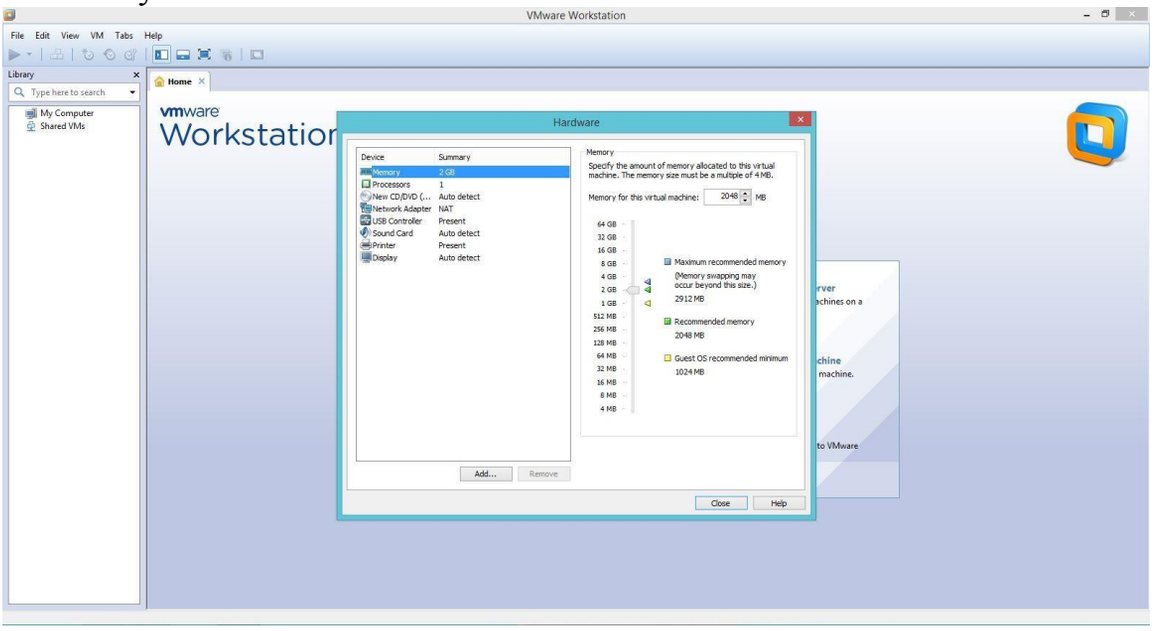
8. Now you get the option that virtual machine is created with the following settings

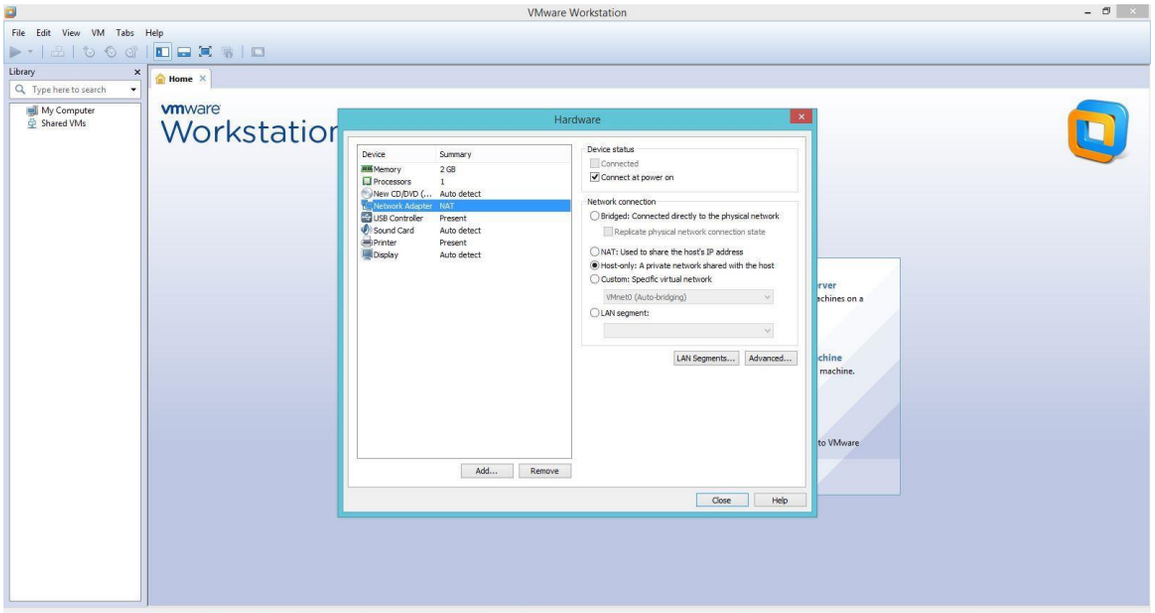


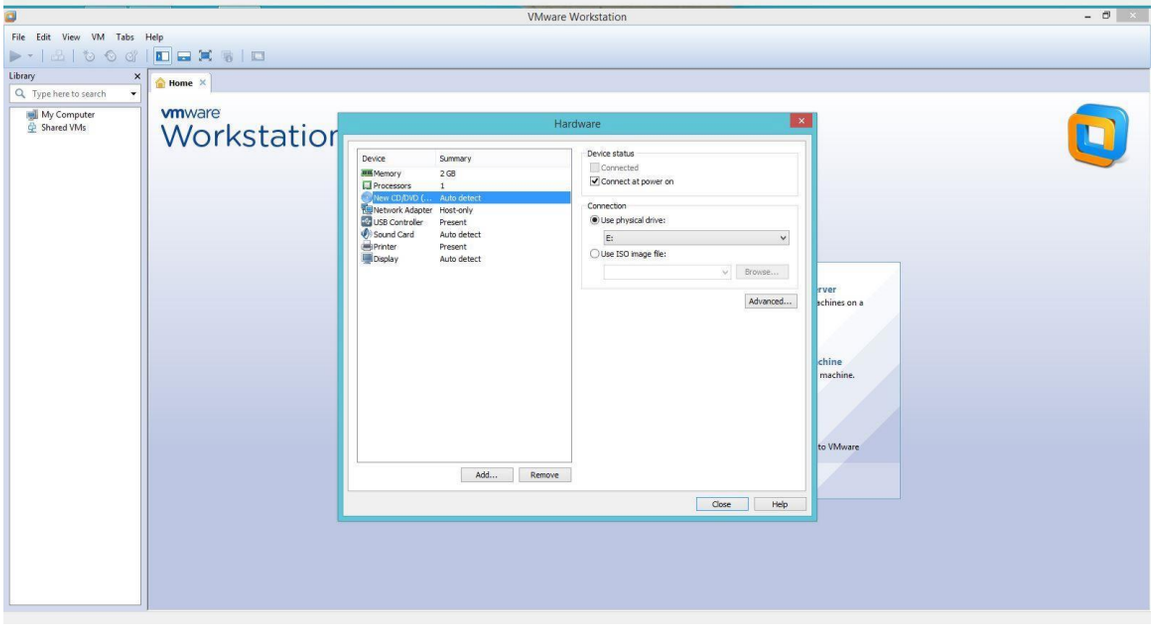
9. Click on “customize Hardware” Select the memory size that is RAM you want to allocate for RedHat virtual machine (1 GB) click Next button.

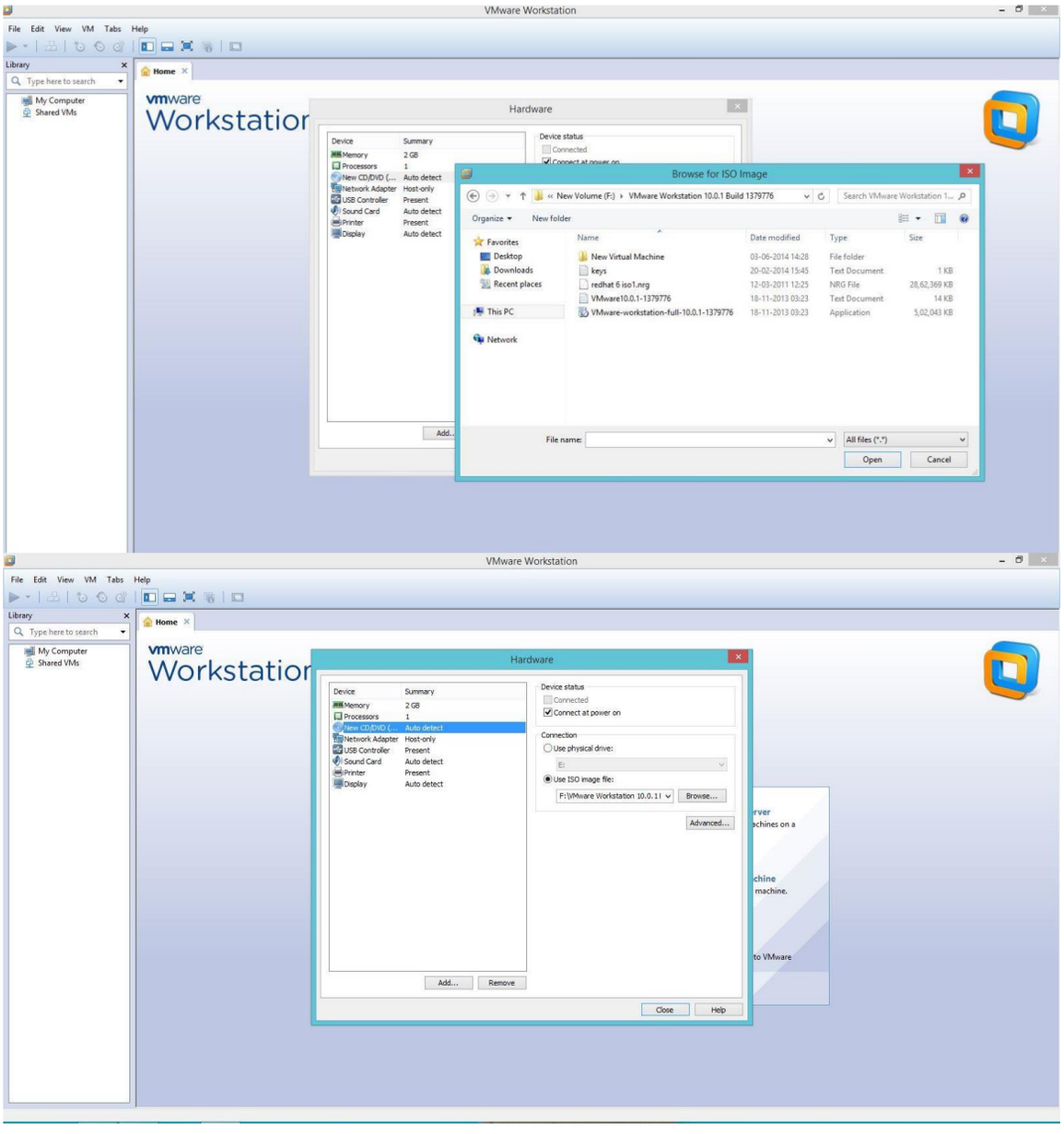
10. Now select create a virtual hard drive to the new machine click create button.

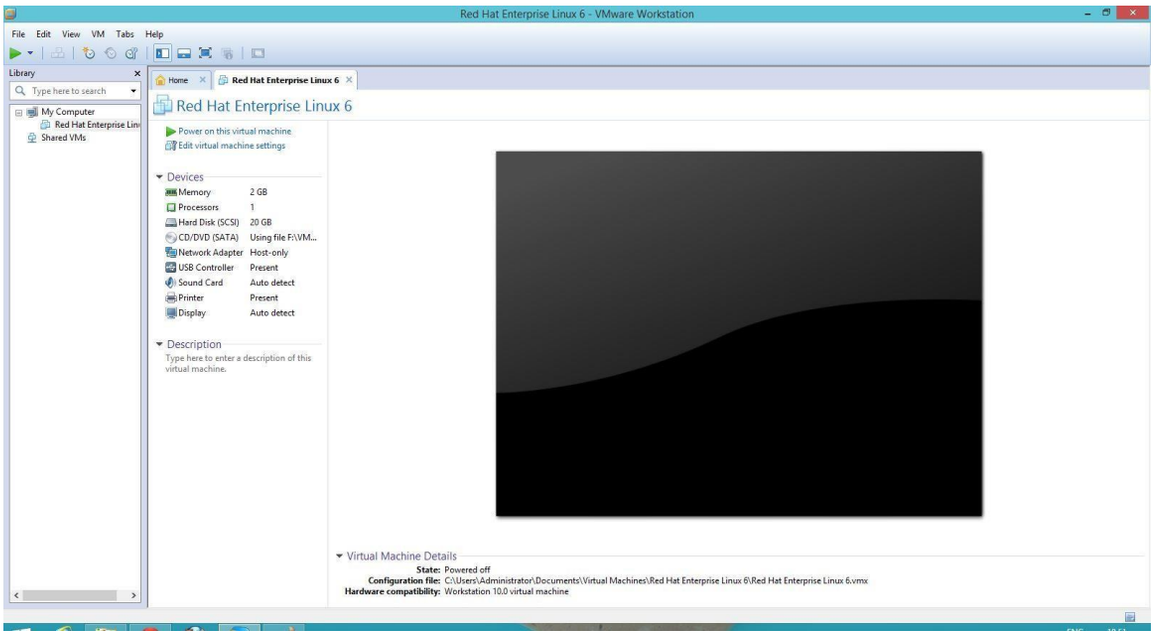
11. Your RedHat Virtual Box operating system drive is created. Now start the RedHat by double-click on it or use Start tab on menu bar.



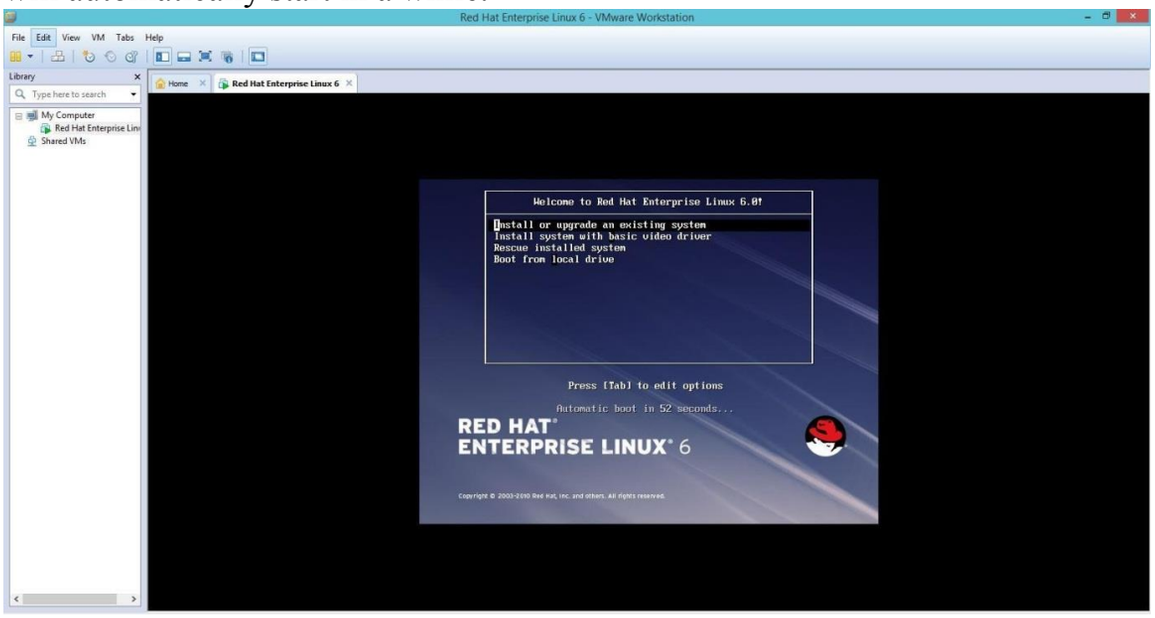

VMware Workstation
File Edit View VM Tabs Help
Library
Type here to search
My Computer
Shared VMs
Home
vmware
Workstation
Hardware
Device
Summary
Memory 2 GB
Processors 1
New CD/DVD (... Auto detect
Network Adapter NAT
USB Controller Present
Sound Card Auto detect
Printer Present
Display Auto detect
Add...
Remove
Device status
Connected
Connect at power on
Network connection
Bridged: Connected directly to the physical network
Replicate physical network connection state
NAT: Used to share the host's IP address
Host-only: A private network shared with the host
Custom: Specific virtual network
VMnet0 (Auto-bridging)
LAN segment:
LAN Segments...
Advanced...
Close
Help


VMware Workstation
File Edit View VM Tabs Help
Library
Type here to search
My Computer
Shared VMs
Home
vmware
Workstation
Hardware
Device
Summary
Memory 2 GB
Processors 1
New CD/DVD (... Auto detect
Network Adapter Host-only
USB Controller Present
Sound Card Auto detect
Printer Present
Display Auto detect
Add...
Remove
Device status
Connected
Connect at power on
Connection
Use physical drive:
E:
Use ISO image file:
Browse...
Advanced...
Close
Help

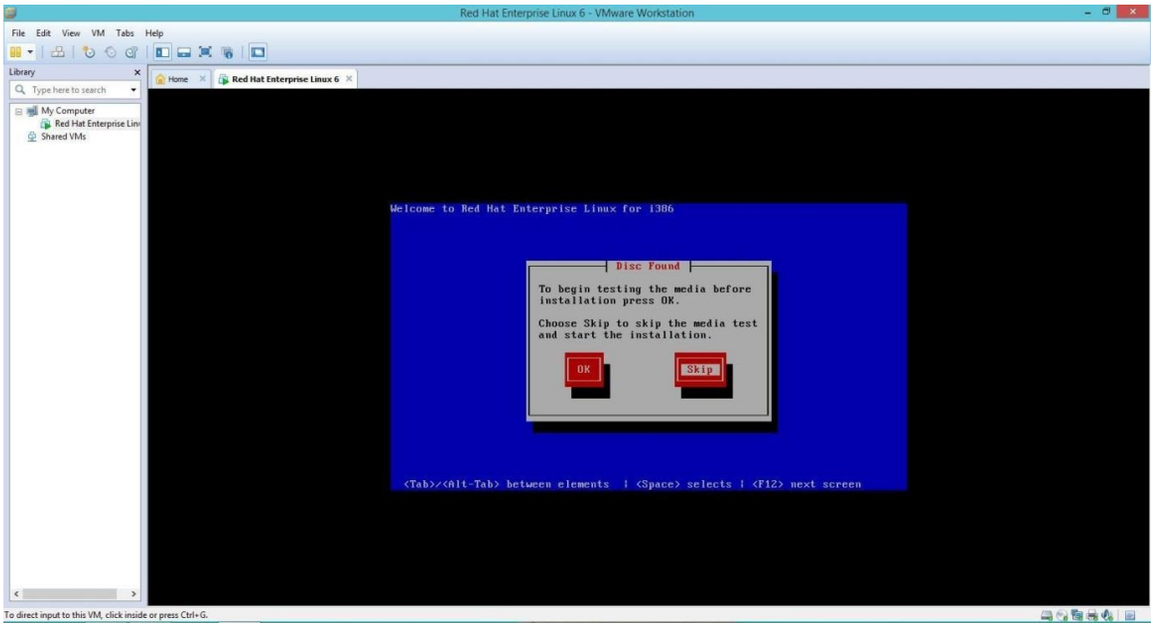

VMware Workstation
File Edit View VM Tabs Help
Library
Type here to search
My Computer
Shared VMs
Home
vmware Workstation
Hardware
Device Summary
Memory 2 GB
Processors 1
New CD/DVD (... Auto detect
Network Adapter Host-only
USB Controller Present
Sound Card Auto detect
Printer Present
Display Auto detect
Device status
Connected
Connect at power on
Browse for ISO Image
« New Volume (F:) › VMware Workstation 10.0.1 Build 1379776
Search VMware Workstation 1...
Organize
New folder
Favorites
Desktop
Downloads
Recent places
This PC
Network
Name
Date modified
Type
Size
New Virtual Machine 03-06-2014 14:28 File folder
keys 20-02-2014 15:45 Text Document 1 KB
redhat 6 iso1.nrg 12-03-2011 12:25 NRG File 28,62,369 KB
VMware10.0.1-1379776 18-11-2013 03:23 Text Document 14 KB
VMware-workstation-full-10.0.1-1379776 18-11-2013 03:23 Application 5,02,043 KB
File name:
All files (*.*)
Open
Cancel
Add...
VMware Workstation
File Edit View VM Tabs Help
Library
Type here to search
My Computer
Shared VMs
Home
vmware Workstation
Hardware
Device Summary
Memory 2 GB
Processors 1
New CD/DVD (... Auto detect
Network Adapter Host-only
USB Controller Present
Sound Card Auto detect
Printer Present
Display Auto detect
Device status
Connected
Connect at power on
Connection
Use physical drive:
E:
Use ISO image file:
F:\VMware Workstation 10.0.1 I
Browse...
Advanced...
Add...
Remove
Close
Help



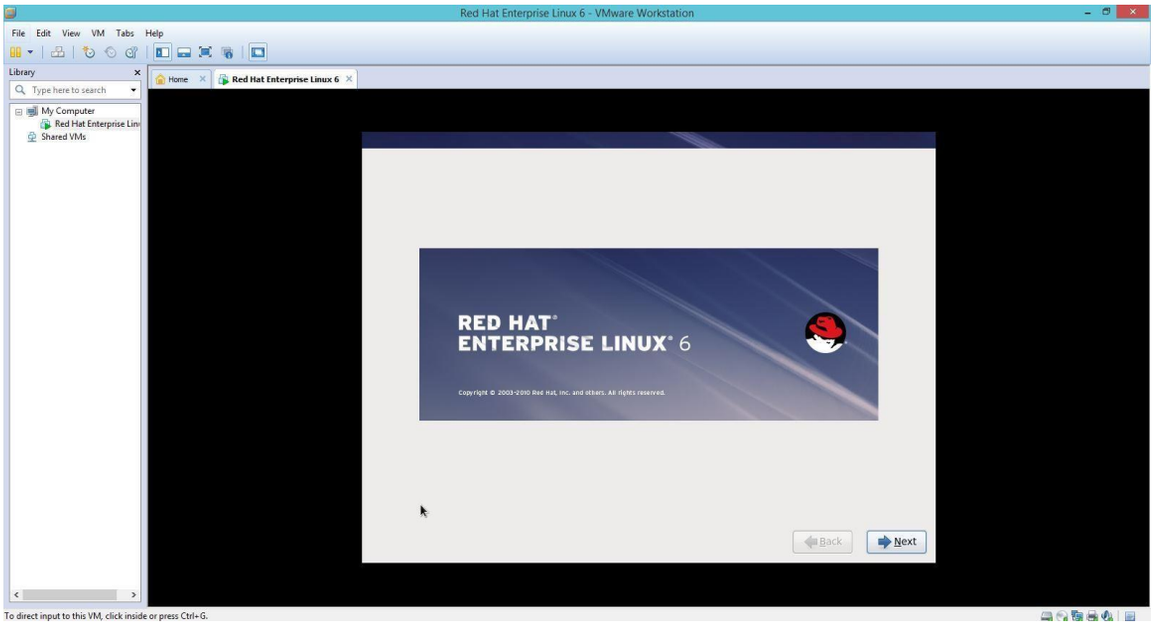
12. Red hat installation starts from here. select " Install or upgrade an existing system " option and press enter. It is a by default graphical installation option or it will automatically start in a while.



13. Here it will prompt for testing media before installation select "Skip" here

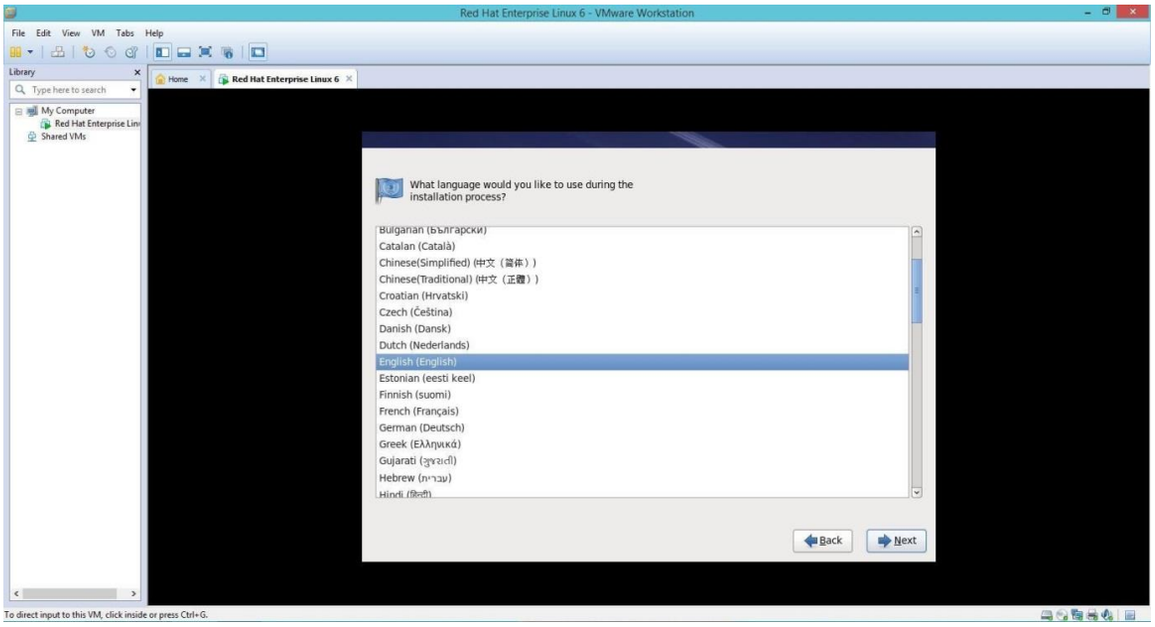


14. Select “Next”



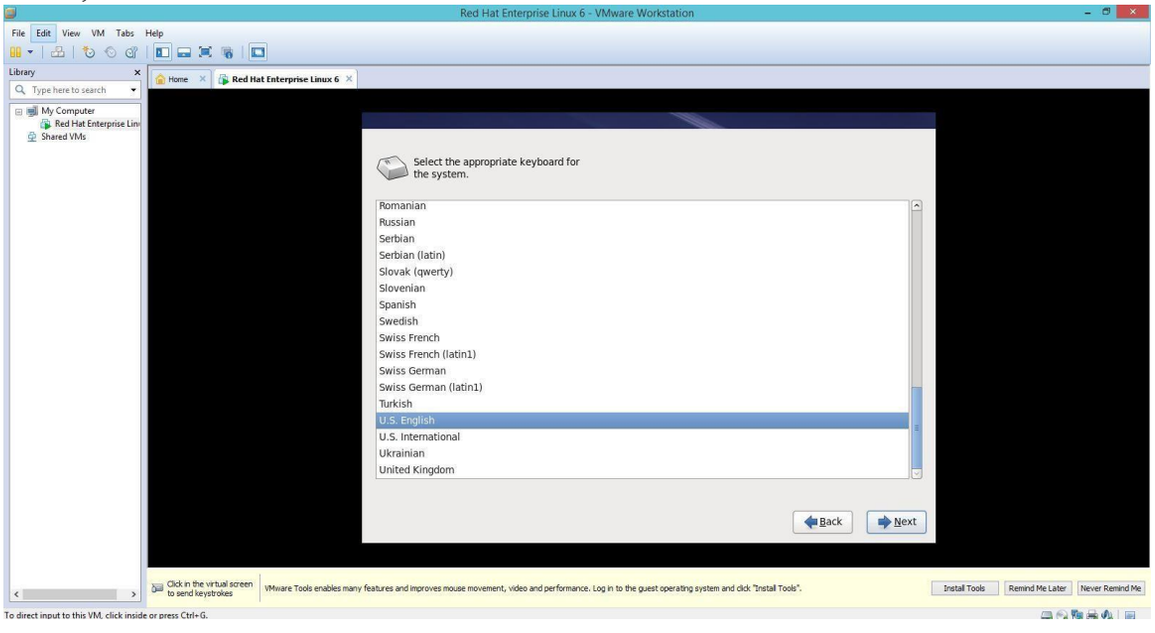
15. Language Selection :--

Using the mouse select a language to use for the installation. The language we select here will become the default language for the operation system once it is installed. Once you select the appropriate language click “Next” button



16. Keyboard configuration :-

Select the correct layout type(for example U.S. english) for the keyboard we should prefer for the installation and as the system default once the selection is made, click “Next” to continue.

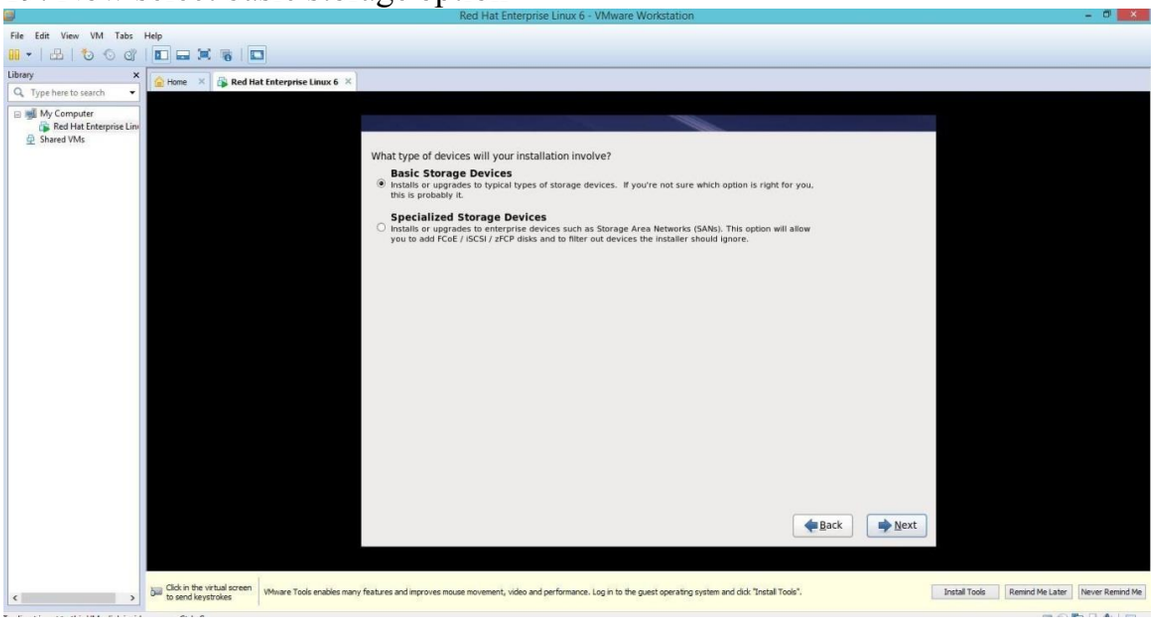


17. Enter the installation number:-

Enter the installation number. This no. will determine the package selection set that is available to the installer. If we choose to skip entering the installation number we will be presented with a basic selection of packages to install later on.

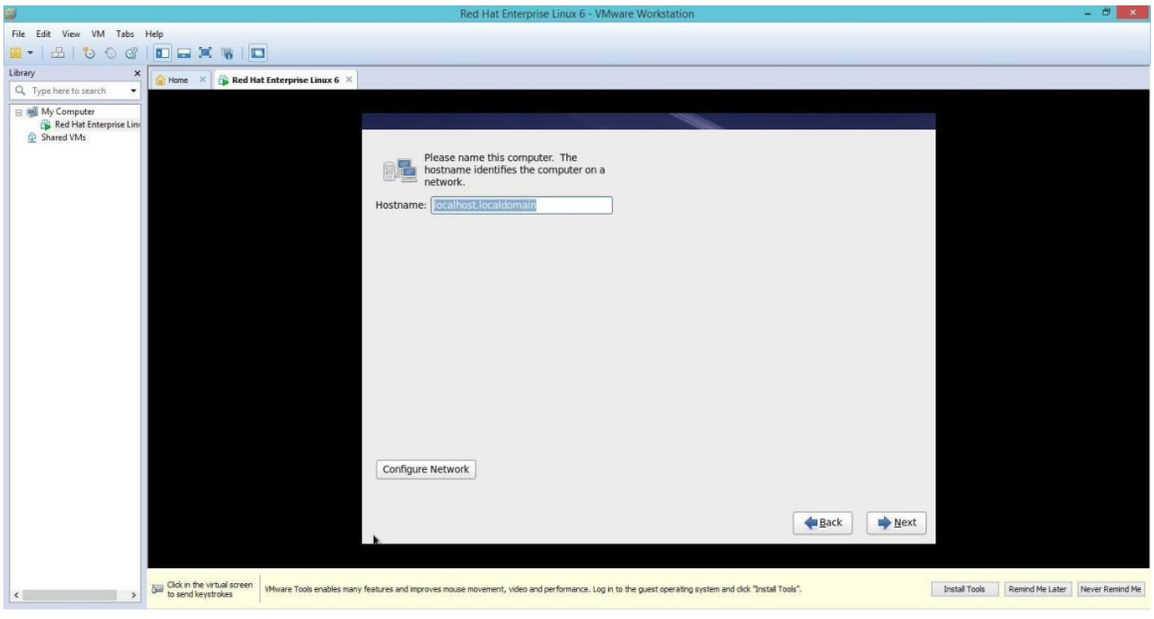
Click on “Skip entering installation number. Then Ok -. Skip-> Yes and then done.

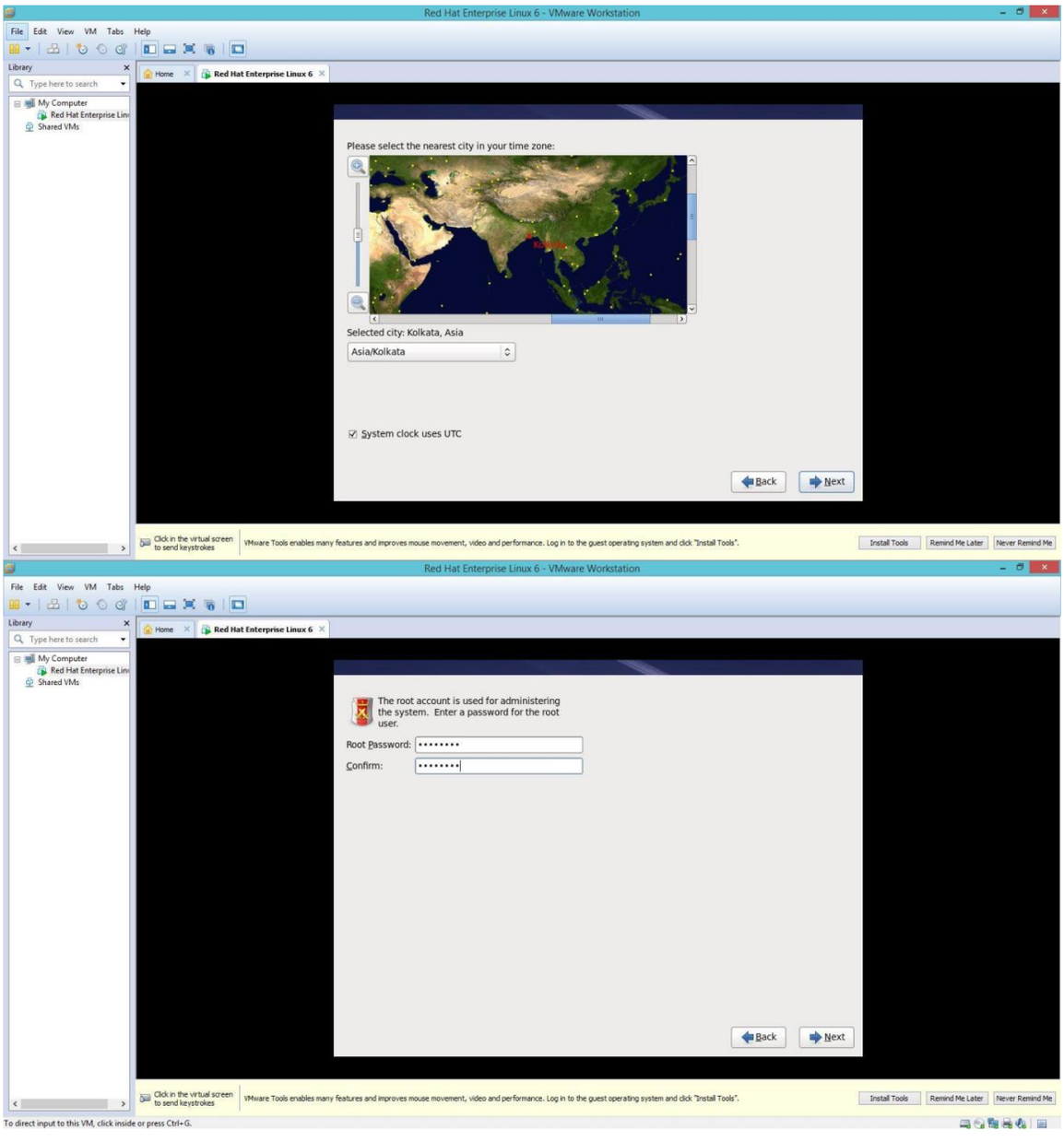
19. Now select basic storage option



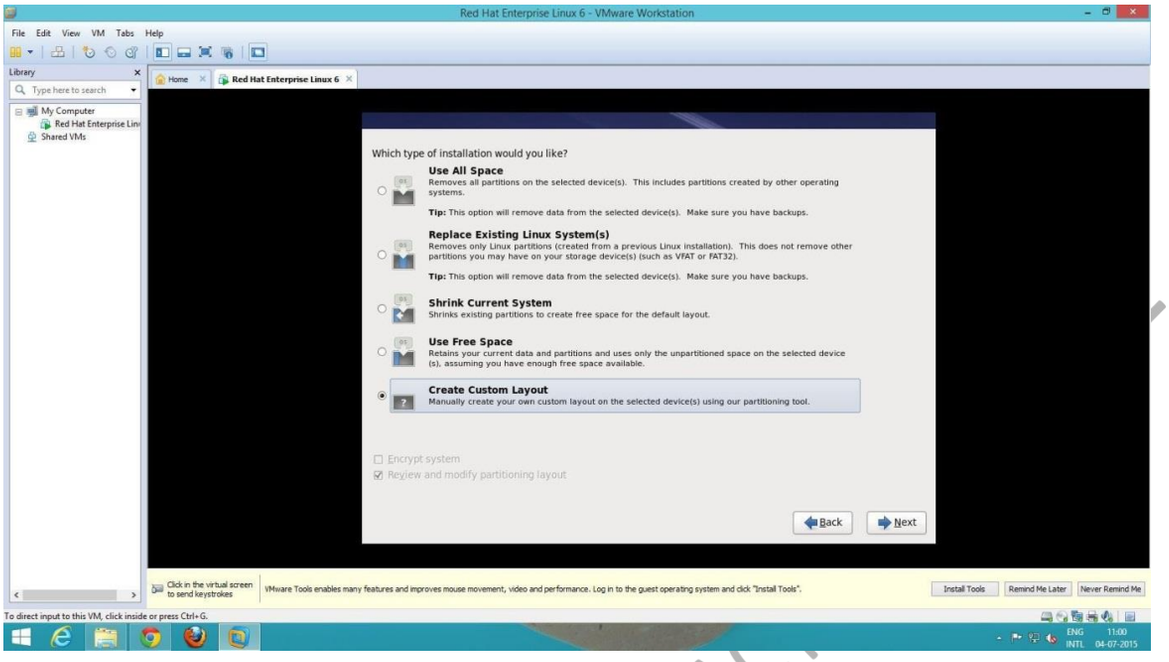
20. Now the system will find the hard disk space and need to re-initialize for creating directories. Select “re-initialize” all option.

21. Now here we assign our Hostname change the hostname as you desire or let it be as localhost.localdomain



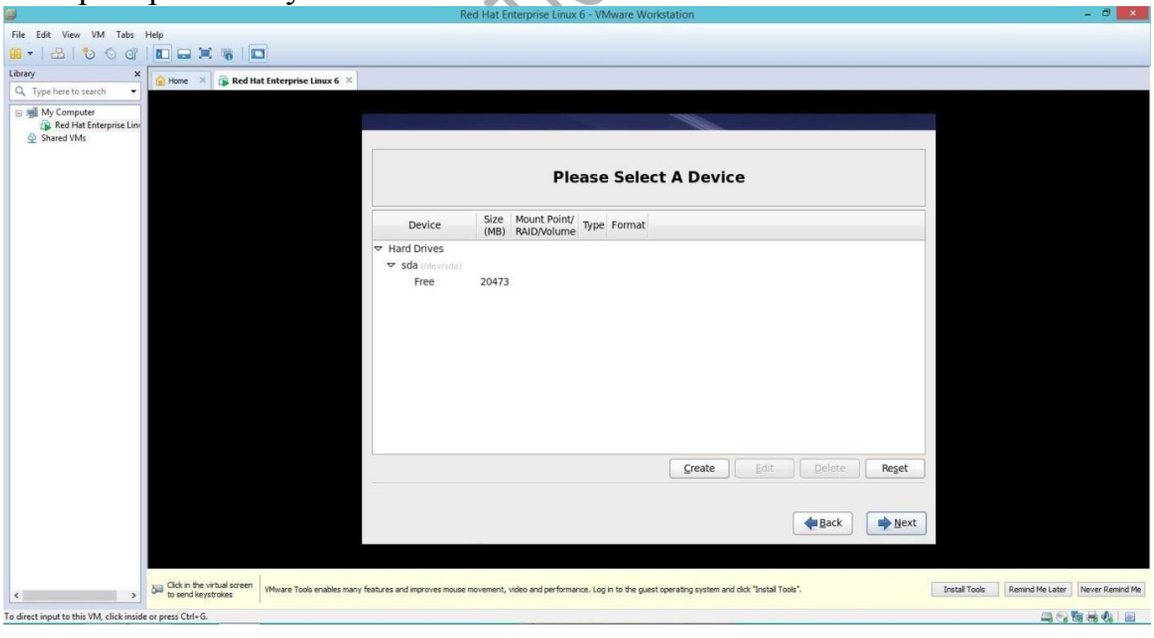


22.. Now select “Create Custom Layout” for manually creating Partitions



23. Disk Partitioning Setup :-

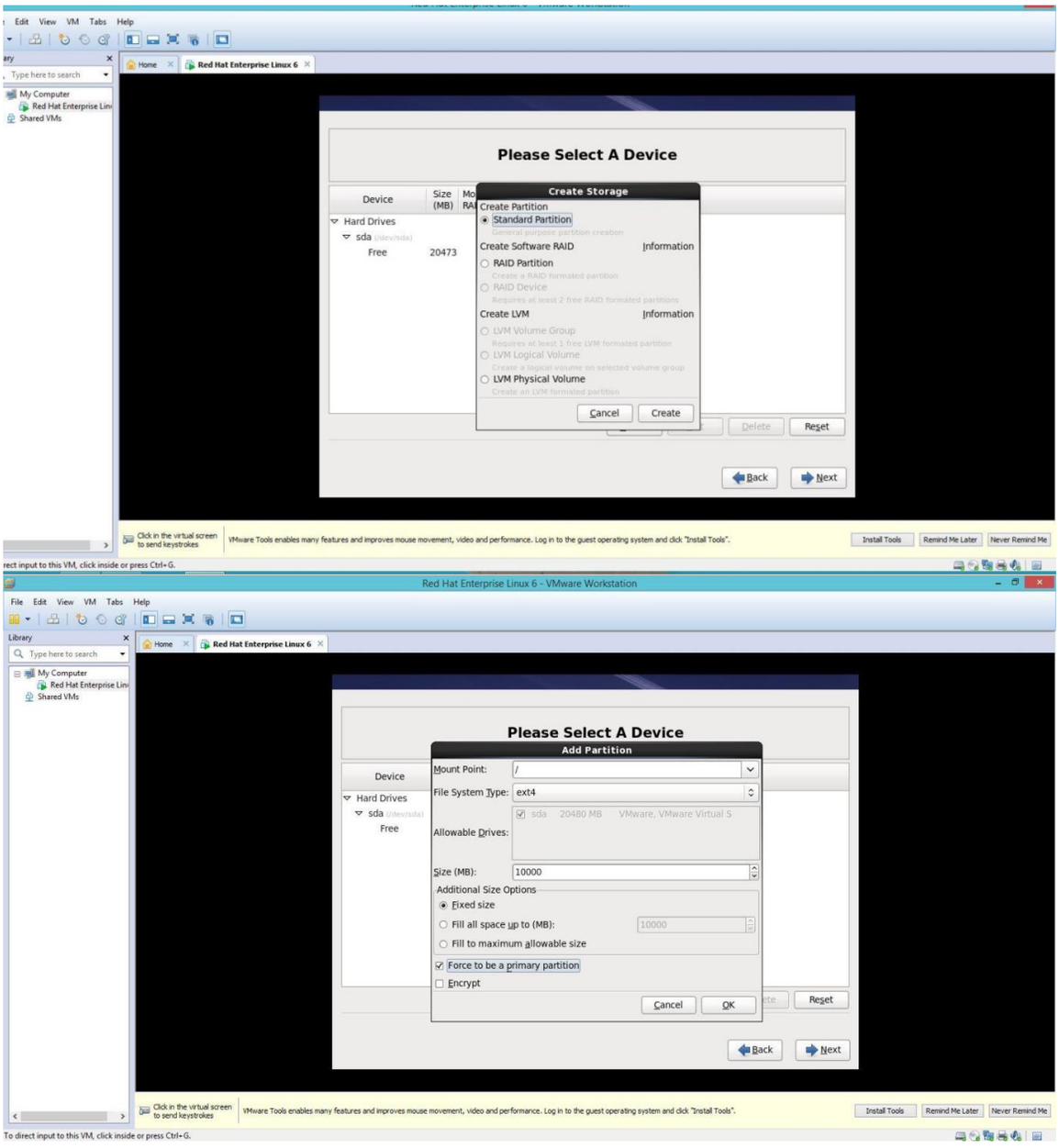
Partitioning allow to divide the hard drive into installed sections where each section behaves as its own hard drive partitioning is particularly useful if we run multiple operation system.

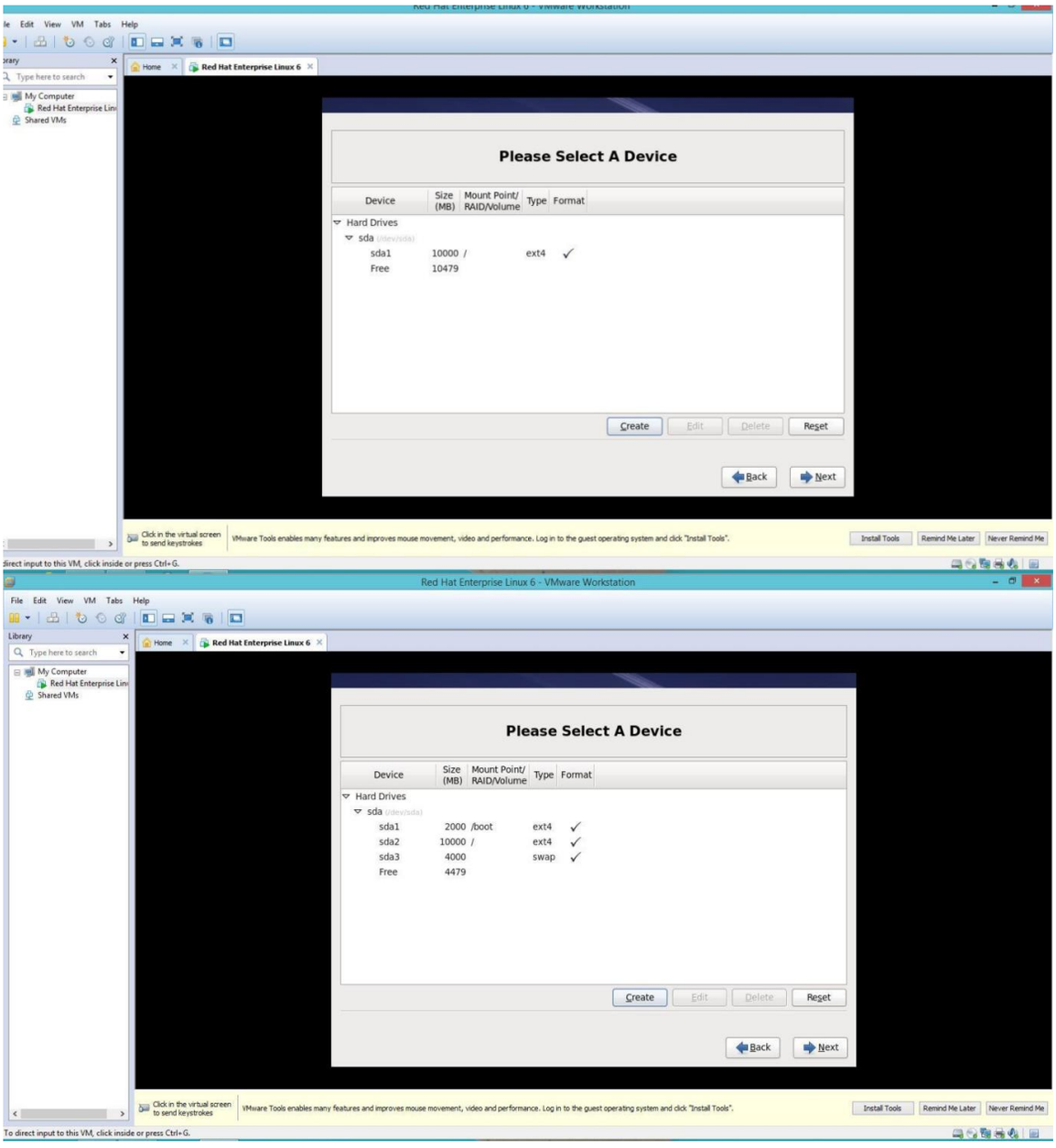


For Root :-

Select the option of create custom layout then create new partitions where mount

point is /(root) of type ext4 click on “force to be primary partitions” and give size as 10000 MB and click Ok





For Swap :-

create new partitions where file system type is swap and size 4000 MB, click Ok

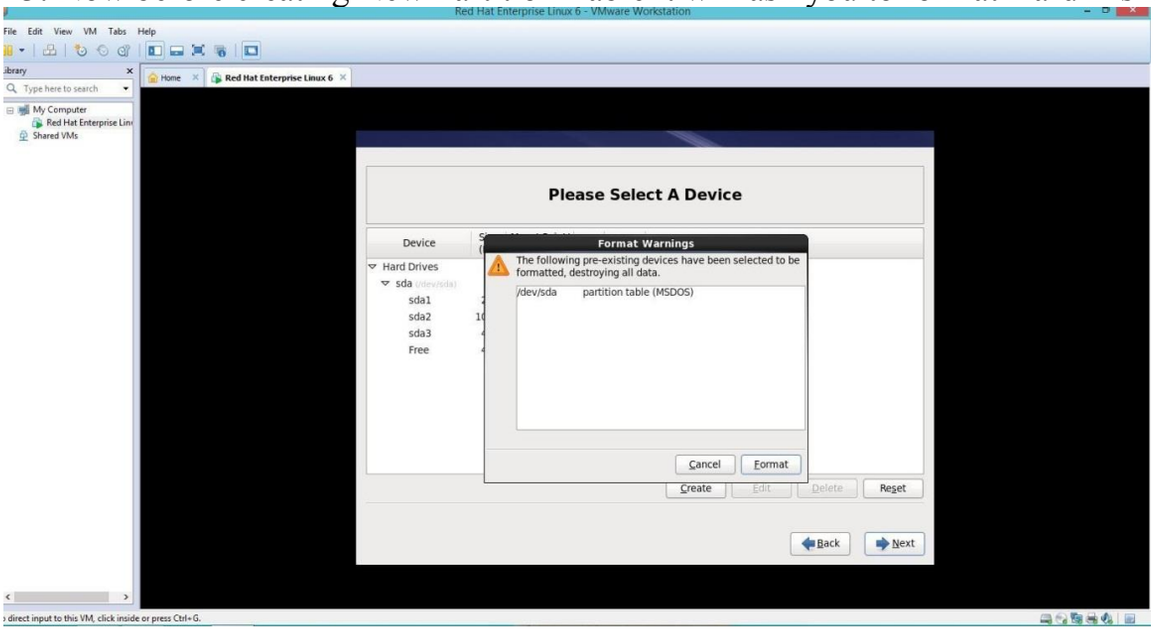
Now partitioning is complete. Click on “next”.

The following is tabular presentation of Disk Partition.

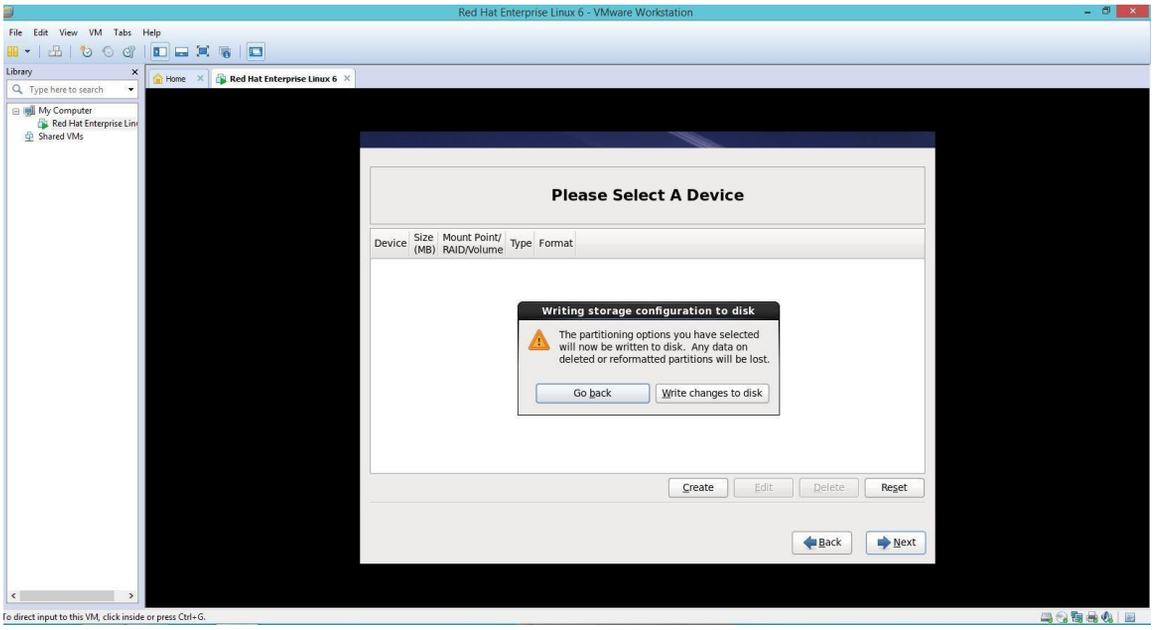
Sr. No	Mount Point	File system type	Size (MB)

1 /(root)Ext3/Ext4 10000 MB
2 - /swap 4000 MB
3 /boot Ext3/Ext4 2000 MB

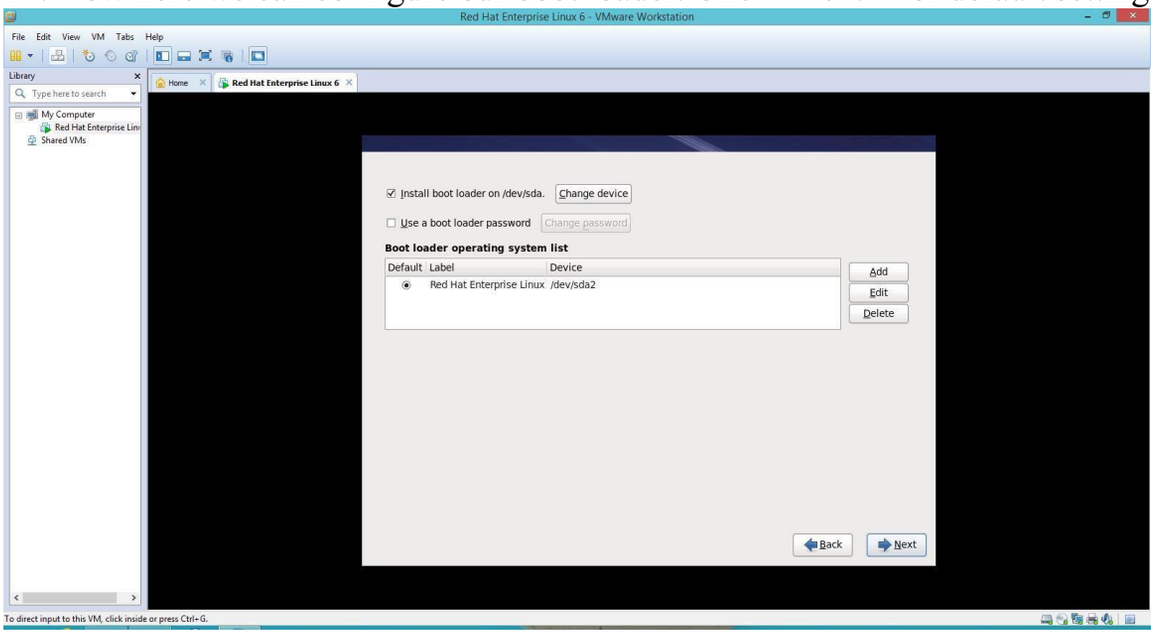
23. Now before creating New Partition Table it will ask you to format Hard Disk.



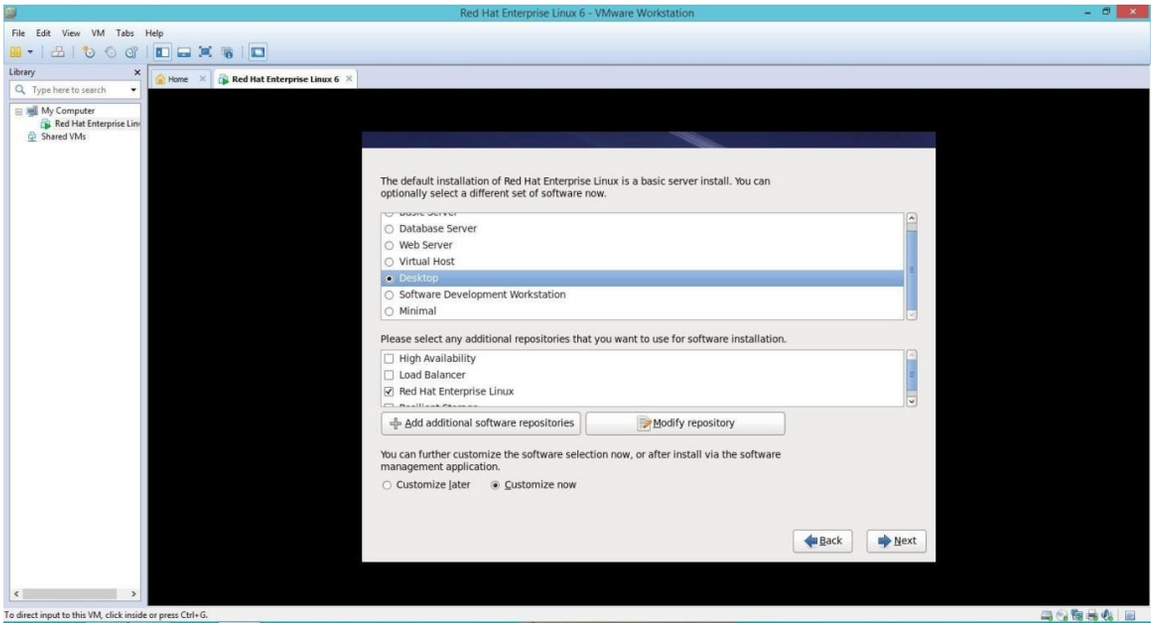
Now it will ask for format and write changes to disk, click on “Write changes to disk”



24. Now here we can configure our boot loader. Click “Next ” for default setting



25. Now it gives you prompt for installation of Software. Select customize now for installation of set of software and click on “Next”



26. Software selection:-

By default, the Red Hat Enterprise Linux installation process loads a selection of software that is
suitable for a system deployed as a basic server. Note that this installation does not include a graphical environment. To include a selection of software suitable for other roles, click the radio button that corresponds to one of the following options:

Basic Server

This option provides a basic installation of Red Hat Enterprise Linux for use on a server.

Database Server

This option provides the MySQL and PostgreSQL databases.

Web server

This option provides the Apache web server.

Enterprise Identity Server Base

This option provides OpenLDAP and Enterprise Identity Management (IPA) to create an
identity and authentication server.

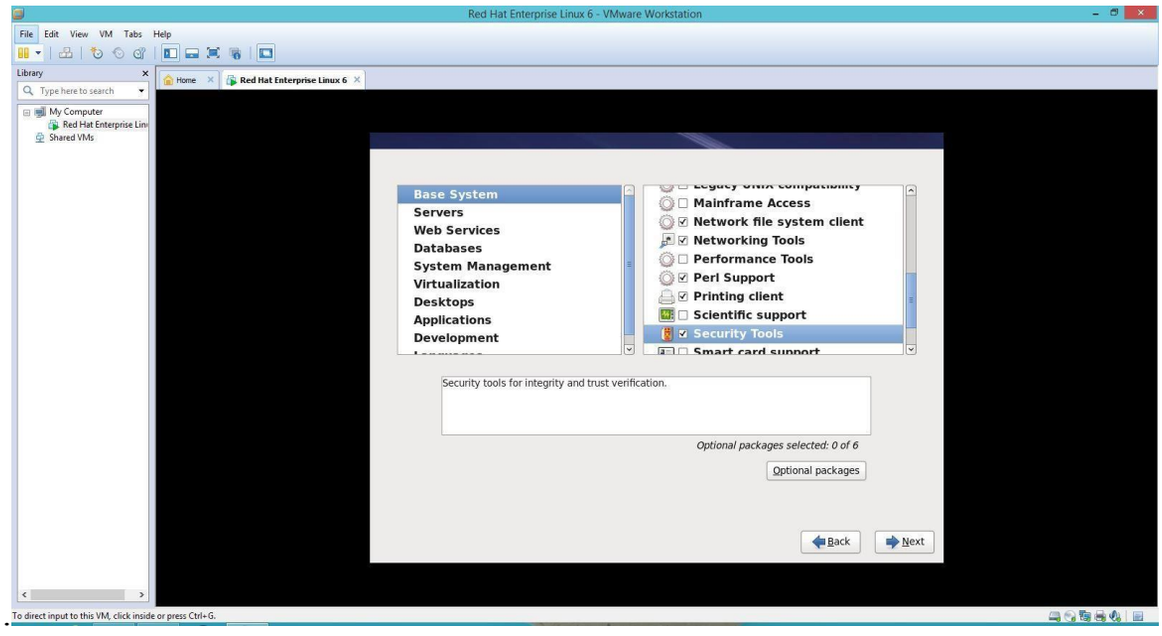
Virtual Host

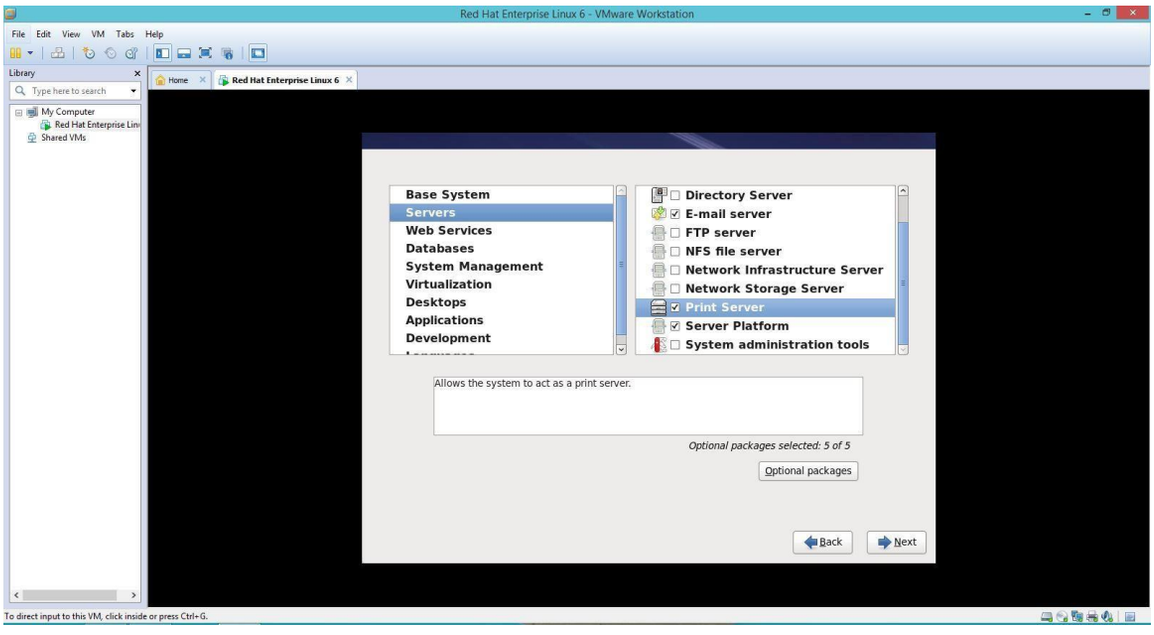
This option provides the KVM and Virtual Machine Manager tools to create a host for virtual
machines.

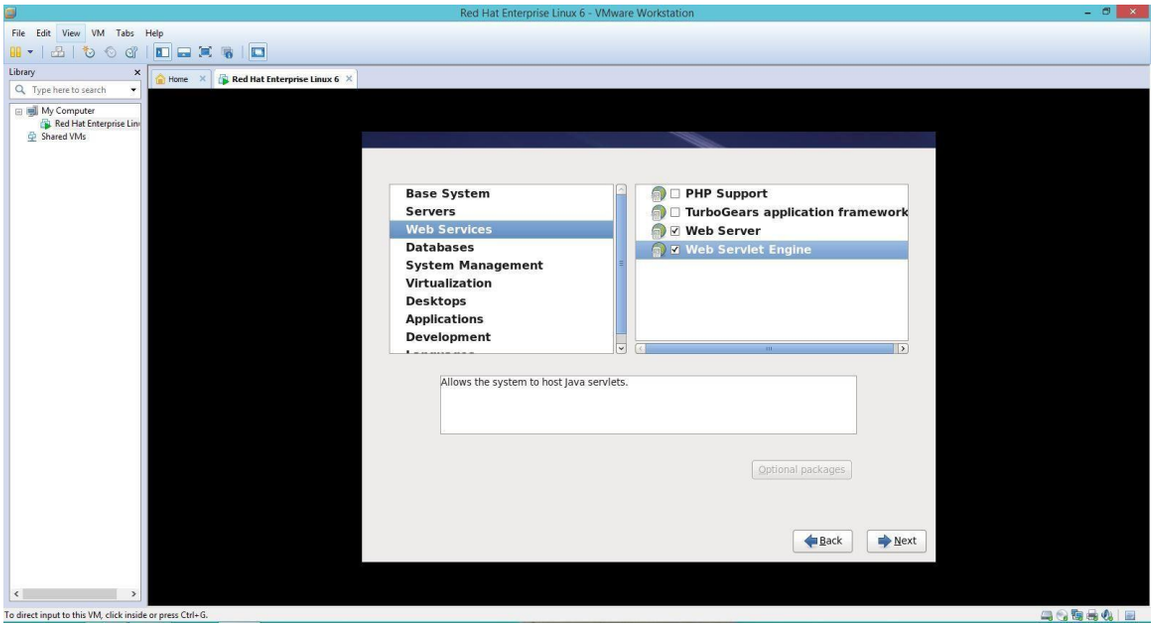
Desktop
This option provides the OpenOffice.org productivity suite, graphical tools such as the GIMP,
and multimedia applications.
Software Development Workstation
This option provides the necessary tools to compile software on your Red Hat Enterprise Linux
system.

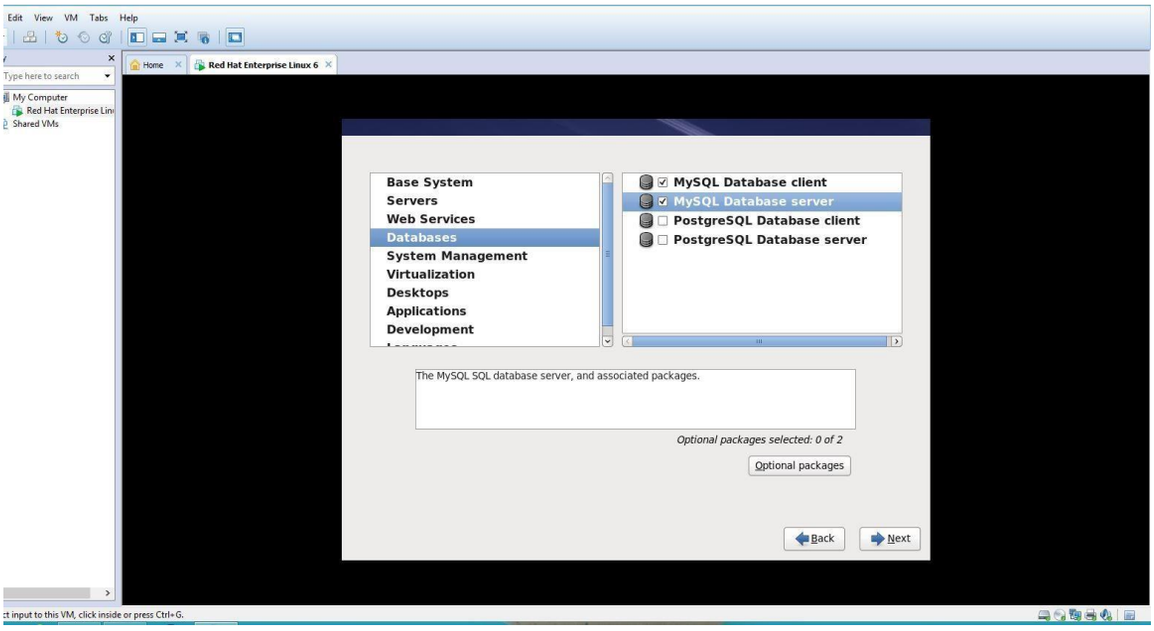
This option provides only the packages essential to run Red Hat Enterprise Linux. A minimal
installation provides the basis for a single-purpose server or desktop appliance and maximizes
performance and security on such an installation.
Click on Customize now and select following software.

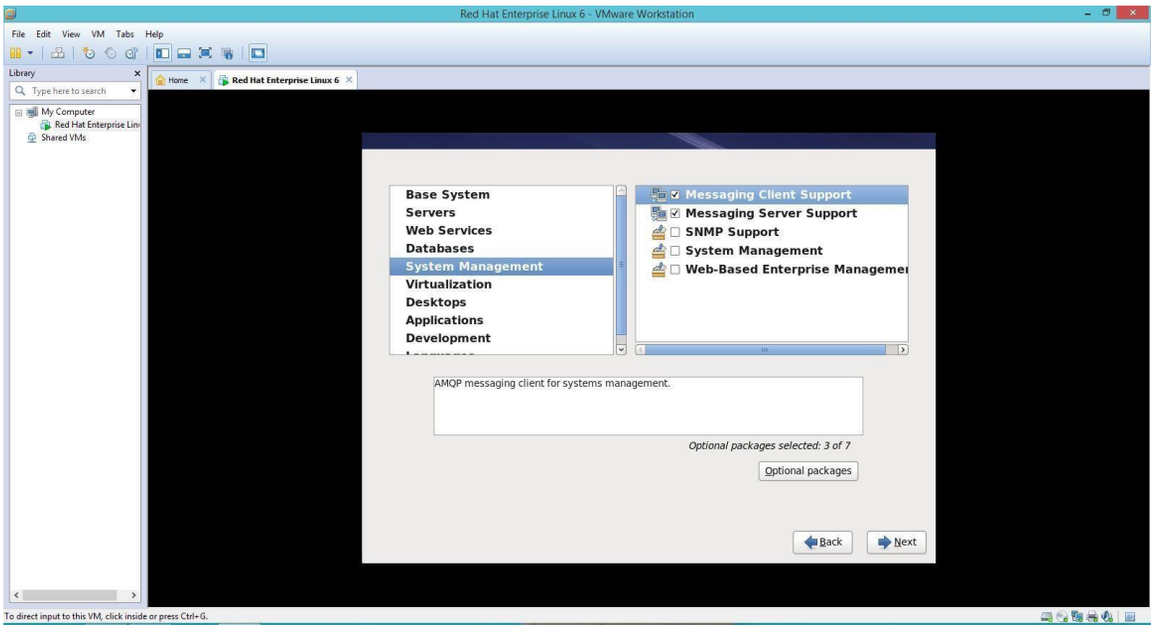
1> Base server -Desktop :-
Desktop
KDE
X-windows
2> Server
3> Web server
4> Database
5> System management

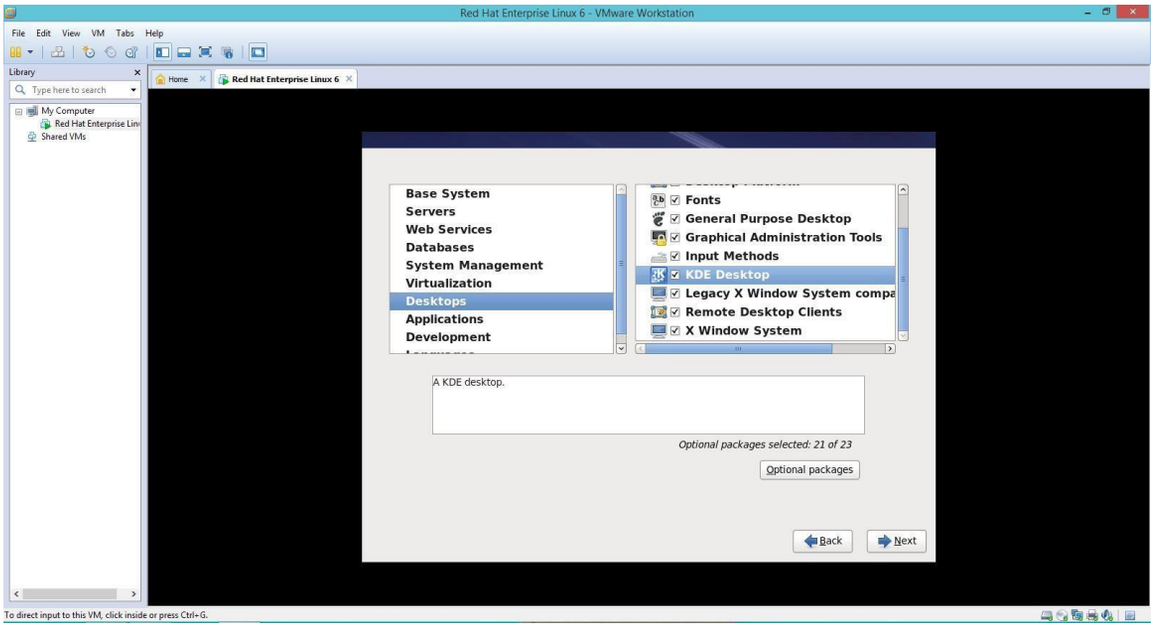

Red Hat Enterprise Linux 6 - VMware Workstation
File Edit View VM Tabs Help
Library
Type here to search
My Computer
Red Hat Enterprise Linux 6
Shared VMs
Home
Red Hat Enterprise Linux 6
Base System
Servers
Web Services
Databases
System Management
Virtualization
Desktops
Applications
Development
Mainframe Access
Network file system client
Networking Tools
Performance Tools
Perl Support
Printing client
Scientific support
Security Tools
Security tools for integrity and trust verification.
Optional packages selected: 0 of 6
Optional packages
Back
Next
To direct input to this VM, click inside or press Ctrl+G.


Red Hat Enterprise Linux 6 - VMware Workstation
File Edit View VM Tabs Help
Library
Type here to search
My Computer
Red Hat Enterprise Linux 6
Shared VMs
Home
Red Hat Enterprise Linux 6
Base System
Servers
Web Services
Databases
System Management
Virtualization
Desktops
Applications
Development
Directory Server
E-mail server
FTP server
NFS file server
Network Infrastructure Server
Network Storage Server
Print Server
Server Platform
System administration tools
Allows the system to act as a print server.
Optional packages selected: 5 of 5
Optional packages
Back
Next
To direct input to this VM, click inside or press Ctrl+G.


Red Hat Enterprise Linux 6 - VMware Workstation
File Edit View VM Tabs Help
Library
Type here to search
My Computer
Red Hat Enterprise Linux 6
Shared VMs
Home
Red Hat Enterprise Linux 6
Base System
Servers
Web Services
Databases
System Management
Virtualization
Desktops
Applications
Development
PHP Support
TurboGears application framework
Web Server
Web Servlet Engine
Allows the system to host Java servlets.
Optional packages
Back
Next
To direct input to this VM, click inside or press Ctrl+G.

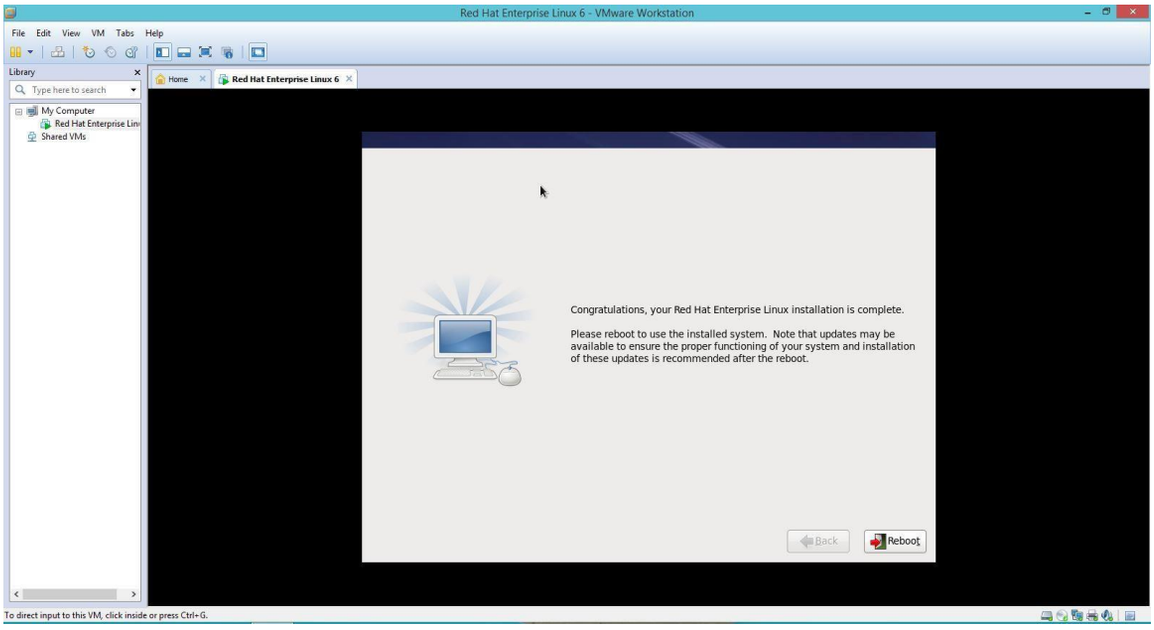

Edit View VM Tabs Help
Type here to search
My Computer
Red Hat Enterprise Linux 6
Shared VMs
Home
Red Hat Enterprise Linux 6
Base System
Servers
Web Services
Databases
System Management
Virtualization
Desktops
Applications
Development
MySQL Database client
MySQL Database server
PostgreSQL Database client
PostgreSQL Database server
The MySQL SQL database server, and associated packages.
Optional packages selected: 0 of 2
Optional packages
Back
Next
ct input to this VM, click inside or press Ctrl+G.



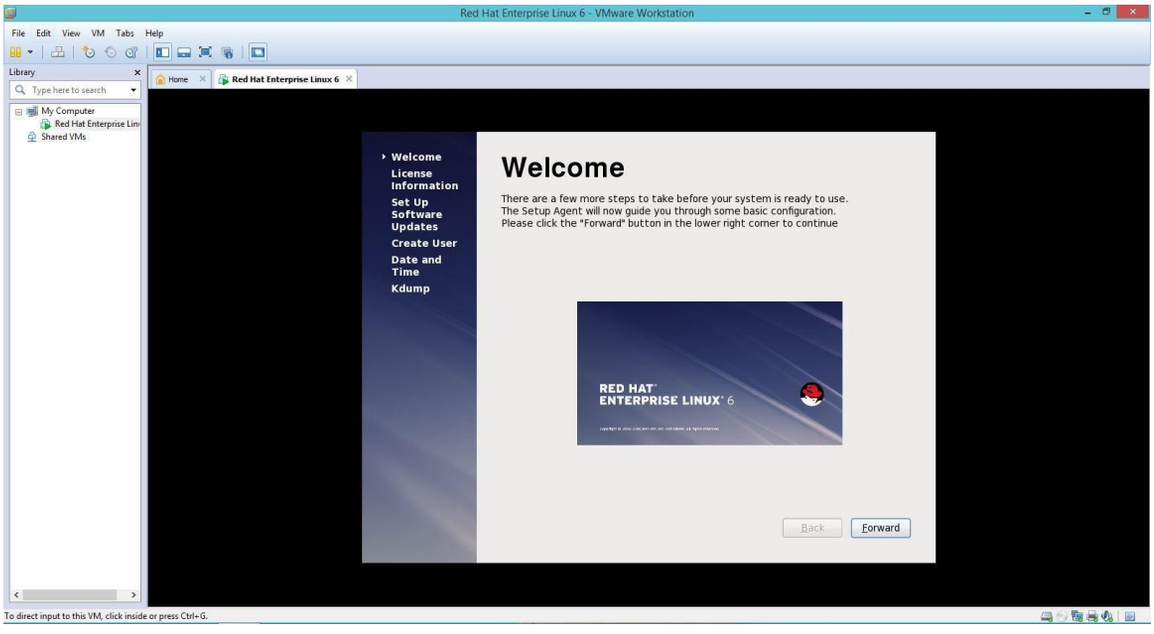


27. Now the next windows showing that it is “Transferring the install image to hard drive” it copy all files to hard drive so installation process get faster

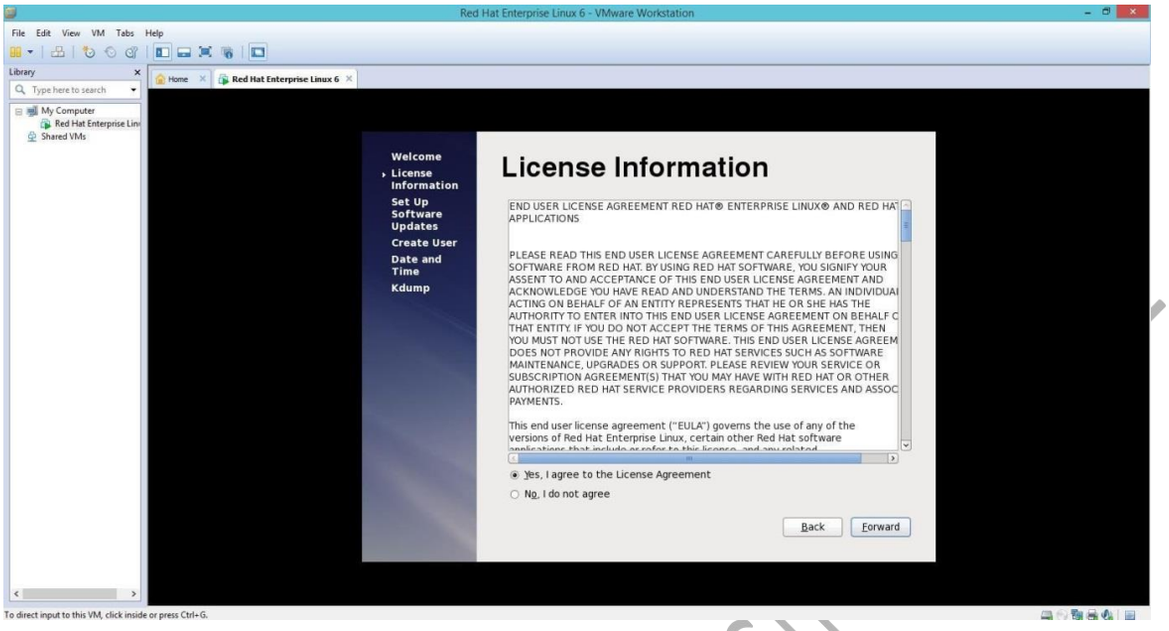
28. Now the installation start from the hard disk files.
29. Installation of Red hat is completed and ask for the reboot. Click on “Reboot”.



30. once the Red hat start it show the window saying few more steps are there for basic configuration. Click on “Forward”.



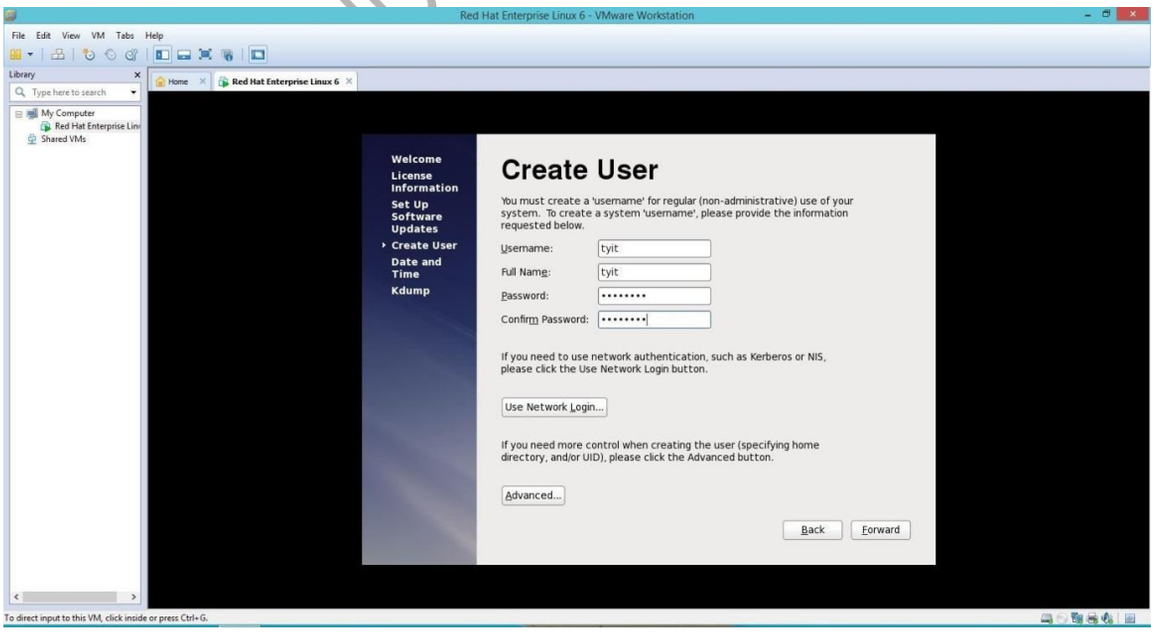
31. Here select “I agree to the license agreement” to proceed and click “Forward”.



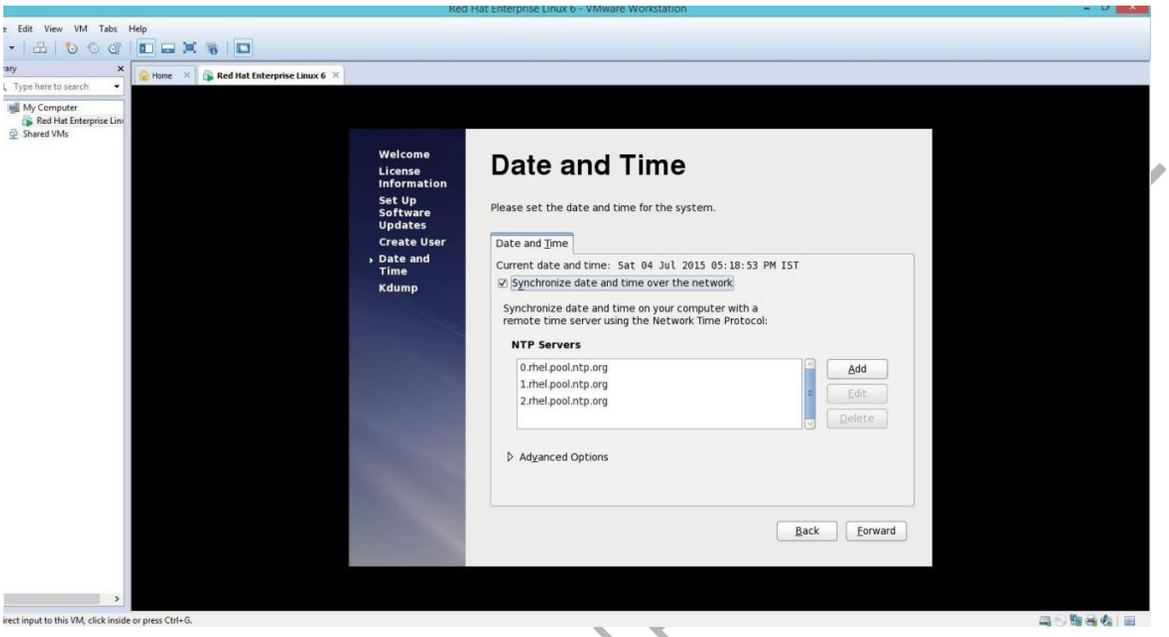
32. Now it asks for software update as we don't have the RHN No. Click "Forward" .

33. Click on "Forward" to finish update setup

35. Now we have to create Normal User for our system. Provide Username and password and click on "forward". The Root user is different from the user we created now. Root user has administrator rights and the user we created is normal user without administrative rights.



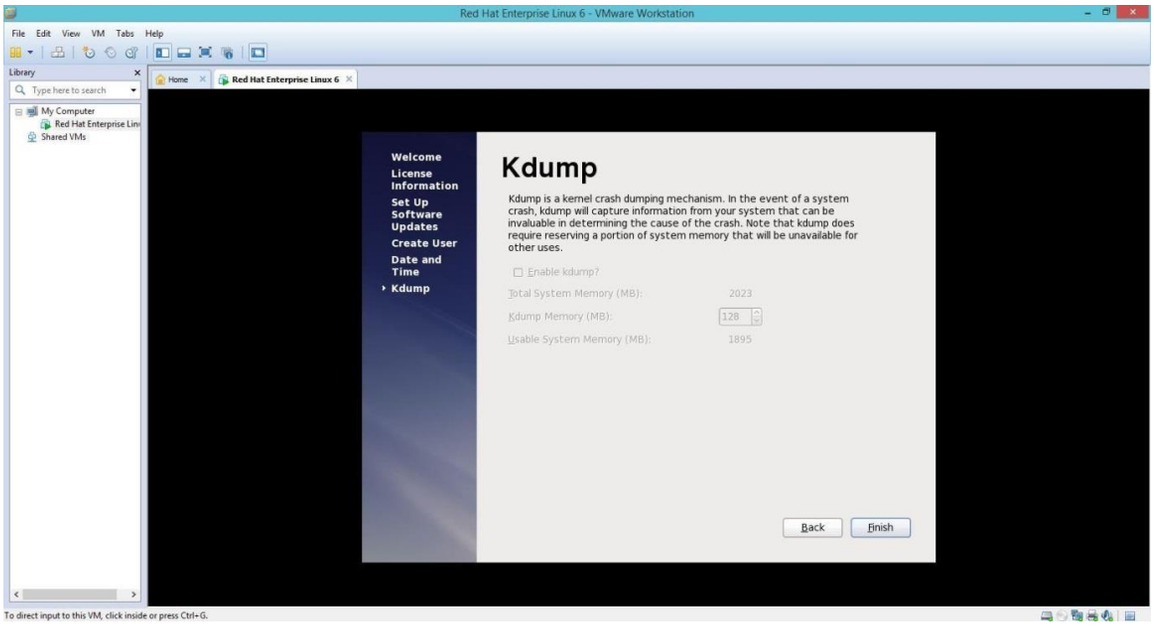
35.Date and Time Zone Configuration
Now select the System date for the window,

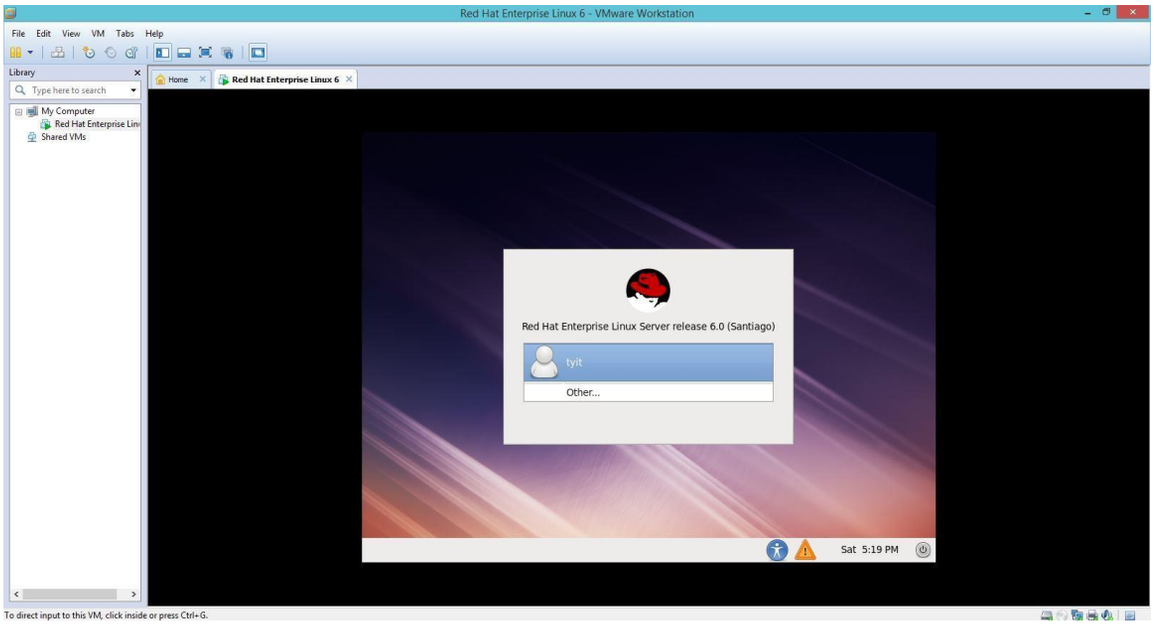


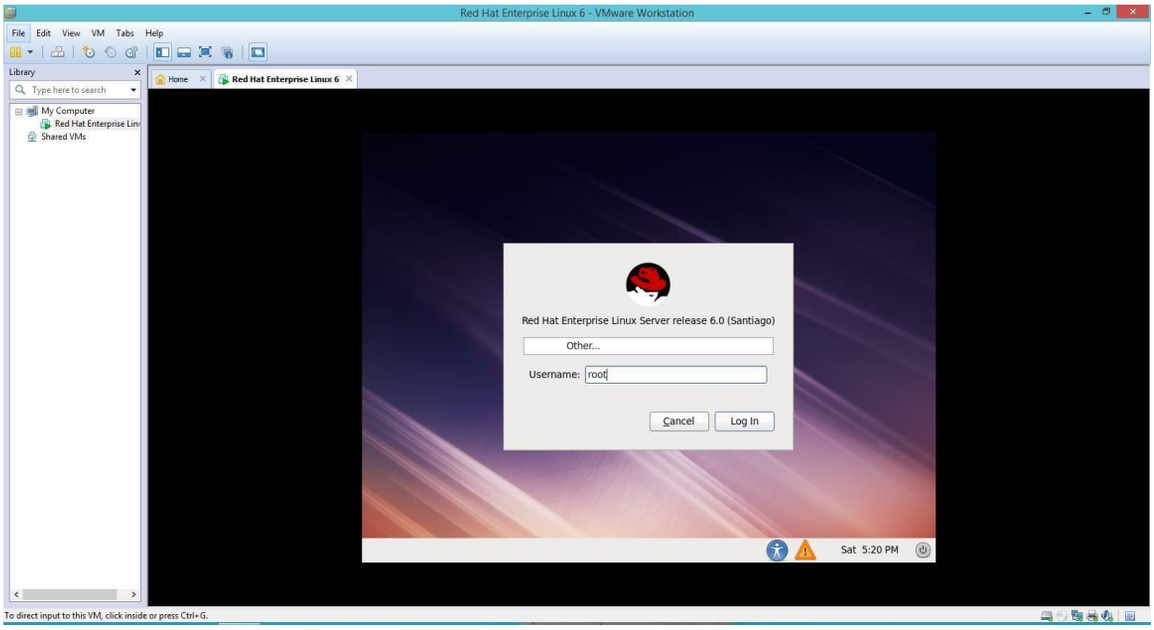
Set your time zone by selecting the city closest to your computer's physical location. Click on the map to
zoom in to a particular geographical region of the world.
From here there are two ways for you to select your time zone:
Using your mouse, click on the interactive map to select a specific city (represented by a yellow dot).
A red X appears indicating your selection.
You can also scroll through the list at the bottom of the screen to select your time zone. Using your mouse, click on a location to highlight your selection.
If Red Hat Enterprise Linux is the only operating system on your computer, select System clock uses UTC. The system clock is a piece of hardware on your computer system. Red Hat Enterprise Linux uses the time zone setting to determine the offset between the local time and UTC on the system clock. This behaviour is standard for systems that use UNIX, Linux, and similar operating systems.
Click Forward to proceed

36. Now it will gives you Error “Insufficient memory to configure kdump”. Click on Ok.

37.. Kdump is used for backup and recovery purpose







39. Now your RedHat Virtual Machine is ready for use. Select the Redhat Virtual Option from VM-Ware Workstation.

Shutting Down

To shut down Red Hat Enterprise Linux, the root user may issue the /sbin/shutdown command. The shutdown man page has a complete list of options, but the two most common uses are:

/sbin/shutdown -h now

and

/sbin/shutdown -r now

After shutting everything down, the -h option halts the machine, and the -r option reboots.

PAM console users can use the reboot and halt commands to shut down the system while in runlevels 1 through 5. For more information about PAM console users, refer to the Red Hat Enterprise Linux Deployment Guide.

If the computer does not power itself down, be careful not to turn off the computer until a message appears indicating that the system is halted.

Failure to wait for this message can mean that not all the hard drive partitions are unmounted, which can lead to file system corruption.

Managing software packages

Searching for software packages

yum allows you to perform a complete set of operations with software packages.

The following section describes how to use **yum** to:

- Search for packages.
- List packages.
- List repositories.
- Display information about the packages.
- List package groups.
- Specify global expressions in yum input.

To search for a package, use:

```
# yum search term
```

Replace *term* with a term related to the package.

Note that `yum search` command returns term matches within the name and summary of the packages. This makes the search faster and enables you to search for packages you do not know the name of, but for which you know a related term.

- To include term matches within package descriptions, use:

  ```
  # yum search --all term
  ```

  Replace *term* with a term you want to search for in a package name, summary, or description.

  Note that `yum search --all` enables a more exhaustive but slower search.

To list information on all installed and available packages, use:

```
# yum list --all
```

To list all packages installed on your system, use:

```
# yum list --installed
```

To list all packages in all enabled repositories that are available to install, use:

```
# yum list --available
```

Installing packages with YUM

- To install a package and all the package dependencies, use:

  ```
  # yum install package-name
  ```

  Replace *package-name* with the name of the package.

- To install multiple packages and their dependencies simultaneously, use:

  ```
  # yum install package-name-1 package-name-2
  ```

  Replace *package-name-1* and *package-name-2* with the names of the packages.

- When installing packages on a *multilib* system (AMD64, Intel 64 machine), you can specify the architecture of the package by appending it to the package name:

  ```
  # yum install package-name.arch
  ```

  Replace *package-name.arch* with the name and architecture of the package.

- If you know the name of the binary you want to install, but not the package name, you can use the path to the binary as an argument:

  ```
  # yum install /usr/sbin/binary-file
  ```

  Replace `/usr/sbin/binary-file` with a path to the binary file.

  yum searches through the package lists, finds the package which provides `/usr/sbin/binary-file`, and prompts you as to whether you want to install it.

- To install a previously-downloaded package from a local directory, use:

  ```
  # yum install /path/
  ```

  Replace */path/* with the path to the package.

Checking for updates with YUM

The following procedure describes how to check the available updates for packages installed on your system using `yum`.

Procedure

- To see which packages installed on your system have available updates, use:

  ```
  # yum check-update
  ```

  The output returns the list of packages and their dependencies that have an update available.

Updating a single package with YUM

Use the following procedure to update a single package and its dependencies using `yum`.

- To update a package, use:

  ```
  # yum update package-name
  ```

  Replace *package-name* with the name of the package.

LAB-2 ACCESSING THE COMMAND LINE

Goal- To log into a Linux system and run simple commands using the shell

Sections • Accessing the Command Line Using the Local Console (and Practice)

- Accessing the Command Line Using the Desktop (and Practice)
- Executing Commands Using the Bash Shell (and Practice)

LAB

ACCESSING THE COMMAND LINE

PERFORMANCE CHECKLIST

In this lab, you will use the Bash shell to efficiently execute commands using shell metacharacters.

RESOURCES:	
Files:	**`/usr/bin/clean-binary-files`**

OUTCOMES

- Practice using shell command line editing and history functions to efficiently execute commands with minor changes.
- Change the password of the **`student`** user to **`T3st1ngT1me`**.
- Execute commands used to identify file types and display parts of text files.

Reset your desktopX system. Perform the following steps on desktopX.

1. Log into your **`desktopX`** system's graphical login screen as **`student`**.
2. Open a terminal window that will provide a **`bash`** prompt.
3. Change **`student`**'s password to **`T3st1ngT1me`**.
4. Display the current time and date.
5. Display the current time in the following format: HH:MM:SS A/PM. Hint: The format string that displays that output is **`%r`**.
6. What kind of file is **`/usr/bin/clean-binary-files`**? Is it readable by humans?
7. Use the **`wc`** command and **`bash`** shortcuts to display the size of **`/usr/bin/clean-binary-files`**.
8. Display the first 10 lines of **`/usr/bin/clean-binary-files`**.
9. Display the last 10 lines at the bottom of the **`/usr/bin/clean-binary-files`** file.
10. Repeat the previous command, but use the **`-n 20`** option to display the last 20 lines in the file. Use command line editing to accomplish this with a minimal amount of keystrokes.
11. Execute the **`date`** command without any arguments to display the current date and time.
12. Use **`bash`** history to display just the time.
13. Finish your session with the **`bash`** shell.

ACCESSING THE COMMAND LINE USING THE LOCAL CONSOLE

OBJECTIVES

After completing this section, students should be able to log into a Linux system on a local text console and run simple commands using the shell.

THE `bash` SHELL

A *command line* is a text-based interface which can be used to input instructions to a computer system. The Linux command line is provided by a program called the *shell*. Over the long history of UNIX-like systems, many shells have been developed. The default shell for users in Red Hat Enterprise Linux is the GNU Bourne–Again Shell (**bash**). Bash is an improved version of one of the most successful shells used on UNIX-like systems, the Bourne Shell (**sh**).

When a shell is used interactively, it displays a string when it is waiting for a command from the user. This is called the *shell prompt*. When a regular user starts a shell, the default prompt ends with a **$** character.

```
[student@desktopX ~]$
```

The **$** is replaced by a **#** if the shell is running as the superuser, **root**. This makes it more obvious that it is a superuser shell, which helps to avoid accidents and mistakes in the privileged account.

```
[root@desktopX ~]#
```

Using **bash** to execute commands can be powerful. The **bash** shell provides a scripting language that can support automation of tasks. The shell has additional capabilities that can simplify or make possible operations that are hard to accomplish efficiently with graphical tools.

NOTE

The **bash** shell is similar in concept to the command line interpreter found in recent versions of Microsoft Windows **cmd.exe**, although **bash** has a more sophisticated scripting language. It is also similar to Windows PowerShell in Windows 7 and Windows Server 2008 R2. Mac OS X administrators who use the Macintosh's Terminal utility may be pleased to note that **bash** is the default shell in Mac OS X.

VIRTUAL CONSOLES

Users access the **bash** shell through a *terminal*. A terminal provides a keyboard for user input and a display for output. On text-based installations, this can be the Linux machine's *physical console*, the hardware keyboard and display. Terminal access can also be configured through serial ports.

Another way to access a shell is from a *virtual console*. A Linux machine's physical console supports multiple virtual consoles which act like separate terminals. Each virtual console supports an independent login session.

If the graphical environment is available, it will run on the *first* virtual console in Red Hat Enterprise Linux 7. Five additional text login prompts are available on consoles two through six (or one through five if the graphical environment is turned off). With a graphical environment running, access a text login prompt on a virtual console by pressing **Ctrl+Alt** and pressing a function key (**F2** through **F6**). Press **Ctrl+Alt+F1** to return to the first virtual console and the graphical desktop.

IMPORTANT

In the pre-configured virtual images delivered by Red Hat, login prompts have been disabled in the virtual consoles.

NOTE

In Red Hat Enterprise Linux 5 and earlier, the first *six* virtual consoles always provided text login prompts. When the graphical environment was launched, it ran on virtual console seven (accessed through **Ctrl+Alt+F7**).

SHELL BASICS

Commands entered at the shell prompt have three basic parts:

- *Command* to run
- *Options* to adjust the behavior of the command
- *Arguments*, which are typically targets of the command

The *command* is the name of the program to run. It may be followed by one or more *options*, which adjust the behavior of the command or what it will do. Options normally start with one or two dashes (**-a** or **--all**, for example) to distinguish them from arguments. Commands may also be followed by one or more *arguments*, which often indicate a target that the command should operate on.

For example, the command line **usermod -L morgan** has a command (**usermod**), an option (**-L**), and an argument (**morgan**). The effect of this command is to lock the password on user morgan's account.

To use a command effectively, a user needs to know what options and arguments it takes and in what order it expects them (the *syntax* of the command). Most commands have a **--help** option. This causes the command to print a description of what it does, a "usage statement" that describes the command's syntax, and a list of the options it accepts and what they do.

Usage statements may seem complicated and difficult to read. They become much simpler to understand once a user becomes familiar with a few basic conventions:

- Square brackets, `[]`, surround optional items.
- Anything followed by `...` represents an arbitrary-length list of items of that type.
- Multiple items separated by pipes, `|`, means only *one* of them can be specified.
- Text in angle brackets, **<>**, represents variable data. For example, **<filename>** means “insert the filename you wish to use here”. Sometimes these variables are simply written in capital letters (e.g., FILENAME).

Consider the first usage statement for the **date** command:

```
[student@desktopX ~]$ date --help
date [OPTION]... [+FORMAT]
```

This indicates that **date** can take an optional list of options (**[OPTION]...**), followed by an optional format string, prefixed with a plus character, **+**, that defines how the current date should be displayed (**[+FORMAT]**). Since both of these are optional, **date** will work even if it is not given options or arguments (it will print the current date and time using its default format).

NOTE

The **man** page for a command has a SYNOPSIS section that provides information about the command's syntax. The **man-pages**(7) man page describes how to interpret all the square brackets, vertical bars, and so forth that users see in SYNOPSIS or a usage message.

When a user is finished using the shell and wants to quit, there are a couple of ways to end the session. The **exit** command terminates the current shell session. Another way to finish a session is by pressing **Ctrl+d**.

REFERENCES

intro(1), **bash**(1), **console**(4), **pts**(4), and **man-pages**(7) man pages

Note: Some details of the console(4) man page, involving init(8) and inittab(5), are outdated.

SOLUTION

LOCAL CONSOLE ACCESS TERMS

Match the following items to their counterparts in the table.

DESCRIPTION	TERM
The interpreter that executes commands typed as strings.	Shell
The visual cue that indicates an interactive shell is waiting for the user to type a command.	Prompt
The name of a program to run.	Command
The part of the command line that adjusts the behavior of a command.	Option
The part of the command line that specifies the target that the command should operate on.	Argument
The hardware display and keyboard used to interact with a system.	Physical console
One of multiple logical consoles that can each support an independent login session.	Virtual console
An interface that provides a display for output and a keyboard for input to a shell session.	Terminal

ACCESSING THE COMMAND LINE USING THE DESKTOP

OBJECTIVES

After completing this section, students should be able to log into the Linux system using the GNOME 3 desktop environment to run commands from a shell prompt in a terminal program.

THE GNOME DESKTOP ENVIRONMENT

The *desktop environment* is the graphical user interface on a Linux system. The default desktop environment in Red Hat Enterprise Linux 7 is provided by GNOME 3. It provides an integrated desktop for users and a unified development platform on top of a graphical framework provided by the X Window System.

The GNOME Shell provides the core user interface functions for the GNOME desktop environment. The **`gnome-shell`** application is highly customizable. By default, RHEL 7 users use the "GNOME Classic" theme for **`gnome-shell`**, which is similar to the GNOME 2 desktop environment. Another available option is the "modern" GNOME 3 theme used by the upstream GNOME project. Either theme can be selected persistently at login by selecting the gear icon next to the Sign In button when entering the user's password.

The first time a new user logs in, an initial setup program runs to help them configure basic account settings. The GNOME Help application is then started on the Getting Started with GNOME screen. This screen includes videos and documentation to help orient new users to the GNOME 3 environment. GNOME Help can be quickly started by pressing **`F1`** in **`gnome-shell`**, by selecting Applications → Documentation → Help, or by running the **`yelp`** command.

STARTING A TERMINAL

To get a shell prompt in GNOME, start a graphical terminal application such as GNOME Terminal. There are several ways to do this. Here are the three most commonly used methods:

- Select Applications → Utilities → Terminal.
- On an empty desktop, right-click, or press the **Menu** key, and select Open in Terminal from the context menu that appears.
- From the Activities Overview, select Terminal from the dash (either from the favorites area or by finding it with either the grid button (inside Utilities grouping) or the search field at the top of the windows overview).

When a terminal window is opened, a shell prompt displays for the user that started the graphical terminal program. The shell prompt and the terminal window's title bar will indicate the current user name, host name, and working directory.

LOCKING THE SCREEN OR LOGGING OUT

Locking the screen, or logging out entirely, can be done from the menu for the user's name on the far right side of the top bar.

To lock the screen, select (User) → Lock or press **Ctrl+Alt+L**. The screen will lock if the graphical session is idle for a few minutes.

A lock screen curtain will appear that shows the system time and the name of the logged-in user. To unlock the screen, press **Enter** or **Space** to raise the lock screen curtain, then enter the user's password on the lock screen.

To log out and end the current graphical login session, select (User) → Log Out from the top bar. A dialog window will appear, giving the option to Cancel the log out within 60 seconds, or confirm the Log Out action.

POWERING OFF OR REBOOTING THE SYSTEM

To shut down the system, select (User) → Power Off from the top bar or press **Ctrl+Alt+Del**. In the dialog that appears, the user can choose to Power Off, Restart the machine, or Cancel the operation. If the user does not make a choice in this dialog, the system will automatically shut down after 60 seconds.

REFERENCES

GNOME Help

- **yelp**

GNOME Help: *Getting Started with GNOME*

- **yelp help:gnome-help/getting-started**

LAB 2.1

GUIDED EXERCISE

THE GNOME 3 DESKTOP ENVIRONMENT

In this lab, you will log in through the graphical display manager as a regular user to become familiar with the GNOME Classic desktop environment provided by GNOME 3.

OUTCOME

A basic orientation to the GNOME 3 desktop environment.

Access the graphical login screen of **desktopX.example.com**.

IMPORTANT

There are two virtual machines available for lab exercises, a desktop machine (generically called **desktopX**) and a server (generically called **serverX**).

Take care to keep straight which virtual machine an exercise wants you to use.

Do each of the following tasks on the **desktopX** machine.

1. Log in as **student** using the password **student**.
   1.1. At the GNOME login screen, click the **student** user account. Enter **student** when prompted for the password.
   1.2. Click Sign In once the password has been typed in.

2. Change the password for **student** from **student** to **55TurnK3y**.
   2.1. The simplest approach is to open GNOME Terminal and use the **passwd** command at the shell prompt.

   On the empty desktop, press the **Menu** key or right-click with the mouse to open the context menu.
   2.2. Select Open in Terminal.
   2.3. In the terminal window that appears, type **passwd** at the shell prompt. Follow the instructions provided by the program to change the **student** password from **student** to **55TurnK3y**.

3. Log out.
   3.1. Select the student → Log Out menu item.
   3.2. Click the Log Out button in the confirmation window that appears.

4. Log back in as **student** with the new password of **55TurnK3y**.
   4.1. At the GNOME login screen, click the **student** user account. Enter **55TurnK3y** when prompted for the password.
   4.2. Click Sign In once the password has been typed in.

5. Determine how to shut down **desktopX** from the graphical interface, but Cancel the operation without shutting down the system.

EXECUTING COMMANDS USING THE BASH SHELL

OBJECTIVES

After completing this section, students should be able to save time running commands from a shell prompt using Bash shortcuts.

BASIC COMMAND SYNTAX

The GNU Bourne-Again Shell (**bash**) is a program that interprets commands typed in by the user. Each string typed into the shell can have up to three parts: the command, options (that begin with a - or --), and arguments. Each word typed into the shell is separated from each other with spaces. Commands are the names of programs that are installed on the system. Each command has its own options and arguments.

The **Enter** key is pressed when a user is ready to execute a command. Each command is typed on a separate line and the output from each command displays before the shell displays a prompt. If a user wants to type more than one command on a single line, a semicolon, **;**, can be used as a command separator. A semicolon is a member of a class of characters called *metacharacters* that has special meanings for **bash**.

NOTE

The command **ps** can accept options without **-** or **--** .This will be covered in Chapter 7.

EXAMPLES OF SIMPLE COMMANDS

The **date** command is used to display the current date and time. It can also be used by the superuser to set the system clock. An argument that begins with a plus sign (+) specifies a format string for the date command.

```
[student@desktopX ~]$ date
Sat Apr  5 08:13:50 PDT 2014
[student@desktopX ~]$ date +%R
08:13
[student@desktopX ~]$ date +%x
04/05/2014
```

The **passwd** command changes a user's own password. The original password for the account must be specified before a change will be allowed. By default, **passwd** is configured to require a strong password, consisting of lowercase letters, uppercase letters, numbers, and symbols, and is not based on a dictionary word. The superuser can use the **passwd** command to change other users' passwords.

```
[student@desktopX ~]$ passwd
Changing password for user student.
Changing password for student.
(current) UNIX password: old_password
New password: new_password
Retype new password: new_password
```

```
passwd: all authentication tokens updated successfully.
```

Linux does not require file name extensions to classify files by type. The **file** command scans the beginning of a file's contents and displays what type it is. The files to be classified are passed as arguments to the command.

```
[student@desktopX ~]$ file /etc/passwd
/etc/passwd: ASCII text
[student@desktopX ~]$ file /bin/passwd
/bin/passwd: setuid ELF 64-bit LSB shared object, x86-64, version 1
(SYSV), dynamically linked (uses shared libs), for GNU/Linux 2.6.32,
BuildID[sha1]=0x91a7160a019b7f5f754264d920e257522c5bce67, stripped
[student@desktopX ~]$ file /home
/home: directory
```

The **head** and **tail** commands display the beginning and end of a file respectively. By default, these commands display 10 lines, but they both have a **-n** option that allows a different number of lines to be specified. The file to display is passed as an argument to these commands.

```
[student@desktopX ~]$ head /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
[student@desktopX ~]$ tail -n 3 /etc/passwd
gdm:x:42:42::/var/lib/gdm:/sbin/nologin
gnome-initial-setup:x:993:991::/run/gnome-initial-setup/:/sbin/nologin
tcpdump:x:72:72::/:/sbin/nologin
```

The **wc** command counts lines, words, and characters in a file. It can take a **-l**, **-w**, or **-c** option to display only the lines, words, or characters, respectively.

```
[student@desktopX ~]$ wc /etc/passwd
  39   70 2005 /etc/passwd
  [student@desktopX ~]$ wc -l /etc/passwd ; wc -l /etc/group
  39 /etc/passwd
  63 /etc/group
[student@desktopX ~]$ wc -c /etc/group /etc/hosts
 843 /etc/group
 227 /etc/hosts
1070 total
```

TAB COMPLETION

Tab completion allows a user to quickly complete commands or file names once they have typed enough at the prompt to make it unique. If the characters typed are not unique, pressing the **Tab** key twice displays all commands that begin with the characters already typed.

```
[student@desktopX ~]$ pas<Tab><Tab>
passwd       paste        pasuspender
[student@desktopX ~]$ pass<Tab>
[student@desktopX  ~]$  passwd
Changing password for user student.
Changing password for student.
(current) UNIX password:
```

Tab completion can be used to complete file names when typing them as arguments to commands. When **Tab** is pressed, it will complete the file name as much as it can. Pressing **Tab** a second time causes the shell to list all of the files that are matched by the current pattern. Type additional characters until the name is unique, then use tab completion to finish off the command line.

```
[student@desktopX ~]$ ls /etc/pas<Tab>[student@desktopX ~]$ ls /etc/passwd<Tab>
passwd   passwd-
```

Arguments and options can be matched with tab completion for many commands. The **useradd** command is used by the superuser, **root**, to create additional users on the system. It has many options that can be used to control how that command behaves. Tab completion following a partial option can be used to complete the option without a lot of typing.

```
[root@desktopX ~]# useradd --<Tab><Tab>
--base-dir         --groups           --no-log-init      --shell
--comment          --help             --non-unique       --skel
--create-home      --home-dir         --no-user-group    --system
--defaults         --inactive         --password         --uid
--expiredate       --key              --root             --user-group
--gid              --no-create-home   --selinux-user
[root@desktopX ~]# useradd --
```

COMMAND HISTORY

The **history** command displays a list of previously executed commands prefixed with a command number.

The exclamation point character, **!**, is a metacharacter that is used to expand previous commands without having to retype them. **!*number*** expands to the command matching the number specified. **!*string*** expands to the most recent command that begins with the string specified.

```
[student@desktopX ~]$ history
...Output omitted...
   23  clear
   24  who
   25  pwd
   26  ls /etc
   27  uptime
   28  ls -l
   29  date
   30  history
[student@desktopX ~]$ !ls
ls -l
total 0
drwxr-xr-x. 2 student student 6 Mar 29 21:16 Desktop
...Output omitted...
```

```
[student@desktopX ~]$ !26
ls /etc
abrt                    hosts                   pulse
adjtime                 hosts.allow             purple
aliases                 hosts.deny              qemu-ga
...Output omitted...
```

The arrow keys can be used to navigate through previous command lines in the shell's history. **Up Arrow** edits the previous command in the history list. **Down Arrow** edits the next command in the history list. Use this key when the **Up Arrow** has been pressed too many times. **Left Arrow** and **Right Arrow** move the cursor left and right in the current command line being edited.

The **Esc+.** key combination causes the shell to copy the last word of the previous command on the current command line where the cursor is. If used repeatedly, it will continue to go through earlier commands.

EDITING THE COMMAND LINE

When used interactively, **bash** has a command line-editing feature. This allows the user to use text editor commands to move around within and modify the current command being typed. Using the arrow keys to move within the current command and to step through the command history was introduced earlier in this session. More powerful editing commands are introduced in the following table.

Useful command line-editing shortcuts

SHORTCUT	DESCRIPTION
Ctrl+a	Jump to the beginning of the command line.
Ctrl+e	Jump to the end of the command line.
Ctrl+u	Clear from the cursor to the beginning of the command line.
Ctrl+k	Clear from the cursor to the end of the command line.
Ctrl+Left Arrow	Jump to the beginning of the previous word on the command line.
Ctrl+Right Arrow	Jump to the end of the next word on the command line.
Ctrl+r	Search the history list of commands for a pattern.

There are several other command line-editing commands available, but these are the most useful commands for beginning users. The other commands can be found in the **bash**(1) man page.

REFERENCES

bash(1), **date**(1), **file**(1), **head**(1), **passwd**(1), **tail**(1), and **wc**(1) man pages

SOLUTION

BASH COMMANDS AND KEYBOARD SHORTCUTS

Match the following Bash shortcuts to their descriptions in the table.

DESCRIPTION	SHELL COMMAND
Jump to the beginning of the previous word on the command line.	**Ctrl+Left Arrow**
Separate commands on the same line.	**;**
Clear from the cursor to the end of the command line.	**Ctrl+k**
Re-execute a recent command by matching the command name.	**!*string***
Shortcut used to complete commands, file names, and options.	**Tab**
Re-execute a specific command in the history list.	**!*number***
Jump to the beginning of the command line.	**Ctrl+a**
Display the list of previous commands.	**history**
Copy the last argument of previous commands.	**Esc+.**

LAB-3 MANAGING FILES FROM THE COMMAND LINE

Goal- To copy, move, create, delete, and organize files while working from the Bash shell prompt.

Sections

- The Linux File System Hierarchy (and Practice)
- Locating Files by Name (and Practice)
- Managing Files Using Command-Line Tools (and Practice)
- Matching File Names Using Path Name Expansion (and Practice)
- Creating Viewing and Editing Text files

LAB

MANAGING FILES WITH SHELL EXPANSION

PERFORMANCE CHECKLIST

In this lab, you will create, move, and remove files and folders using a variety of file name matching shortcuts.

OUTCOMES

Familiarity and practice with many forms of wildcards for locating and using files.

Perform the following steps on serverX unless directed otherwise. Log in as **`student`** and begin the lab in the home directory.

1. To begin, create sets of empty practice files to use in this lab. If an intended shell expansion shortcut is not immediately recognized, students are expected to use the solution to learn and practice. Use shell tab completion to locate file path names easily.

   Create a total of 12 files with names **`tv_seasonX_episodeY.ogg`**. Replace *X* with the season number and *Y* with that season's episode, for two seasons of six episodes each.

2. As the author of a successful series of mystery novels, your next bestseller's chapters are being edited for publishing. Create a total of eight files with names **`mystery_chapterX.odf`**. Replace *X* with the numbers 1 through 8.

3. To organize the TV episodes, create two subdirectories named **`season1`** and **`season2`** under the existing **`Videos`** directory. Use one command.

4. Move the appropriate TV episodes into the season subdirectories. Use only two commands, specifying destinations using relative syntax.

5. To organize the mystery book chapters, create a two-level directory hierarchy with one command. Create **`my_bestseller`** under the existing **`Documents`** directory, and **`chapters`** beneath the new **`my_bestseller`** directory.

6. Using one command, create three more subdirectories directly under the **`my_bestseller`** directory. Name these subdirectories **`editor`**, **`plot_change`**, and **`vacation`**. The *create parent* option is not needed since the **`my_bestseller`** parent directory already exists.

7. Change to the **`chapters`** directory. Using the home directory shortcut to specify the source files, move all book chapters into the **`chapters`** directory, which is now your current directory. What is the simplest syntax to specify the destination directory?

8. The first two chapters are sent to the editor for review. To remember to not modify these chapters during the review, move those two chapters only to the **`editor`** directory. Use relative syntax starting from the **`chapters`** subdirectory.

9. Chapters 7 and 8 will be written while on vacation. Move the files from **`chapters`** to **`vacation`**. Use one command without wildcard characters.

10. With one command, change the working directory to the season 2 TV episodes location, then copy the first episode of the season to the **`vacation`** directory.

11. With one command, change the working directory to **`vacation`**, then list its files. Episode 2 is also needed. Return to the **`season2`** directory using the *previous working directory* shortcut.

This will succeed if the last directory change was accomplished with one command. Copy the episode 2 file into **vacation**. Return to **vacation** using the shortcut again.

12. Chapters 5 and 6 may need a plot change. To prevent these changes from modifying original files, copy both files into **plot_change**. Move up one directory to **vacation**'s parent directory, then use one command from there.

13. To track changes, make three backups of chapter 5. Change to the **plot_change** directory. Copy **mystery_chapter5.odf** as a new file name to include the full date (Year-Mo-Da). Make another copy appending the current timestamp (as the number of seconds since the *epoch*) to ensure a unique file name. Also make a copy appending the current user (*$USER*) to the file name. See the solution for the syntax of any you are unsure of (like what arguments to pass to the **date** command).

    Note, we could also make the same backups of the chapter 6 files too.

14. The plot changes were not successful. Delete the **plot_change** directory. First, delete all of the files in the **plot_change** directory. Change directory up one level because the directory cannot be deleted while it is the working directory. Try to delete the directory using the **rm** command *without* the *recursive* option. This attempt should fail. Now use the **rmdir** command, which will succeed.

15. When the vacation is over, the **vacation** directory is no longer needed. Delete it using the **rm** command with the *recursive* option.

    When finished, return to the home directory.

THE LINUX FILE SYSTEM HIERARCHY

OBJECTIVES

After completing this section, students should be able to understand fundamental file system layout, organization, and the location of key file types.

THE FILE SYSTEM HIERARCHY

All files on a Linux system are stored on file systems which are organized into a single *inverted* tree of directories, known as a *file system hierarchy*. This tree is inverted because the root of the tree is said to be at the *top* of the hierarchy, and the branches of directories and subdirectories stretch *below* the root.

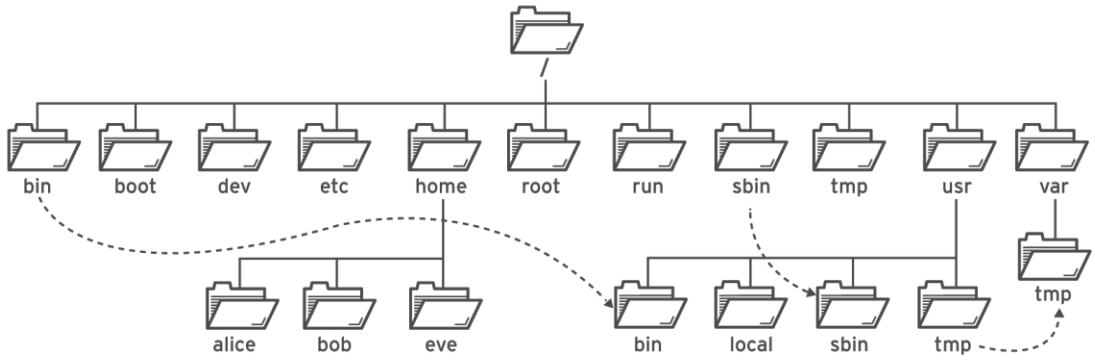


Figure 2.1: Significant file system directories in Red Hat Enterprise Linux 7

The directory `/` is the root directory at the top of the file system hierarchy. The `/` character is also used as a *directory separator* in file names. For example, if `etc` is a subdirectory of the `/` directory, we could call that directory `/etc`. Likewise, if the `/etc` directory contained a file named `issue`, we could refer to that file as `/etc/issue`.

Subdirectories of `/` are used for standardized purposes to organize files by type and purpose. This makes it easier to find files. For example, in the root directory, the subdirectory `/boot` is used for storing files needed to boot the system.

NOTE

The following terms are encountered in describing file system directory contents:

- *static* is content that remains unchanged until explicitly edited or reconfigured.
- *dynamic* or *variable* is content typically modified or appended by active processes.
- *persistent* is content, particularly configuration settings, that remain after a reboot.
- *runtime* is process- or system-specific content or attributes cleared during reboot.

The following table lists some of the most important directories on the system by name and purpose.

Important Red Hat Enterprise Linux directories

LOCATION	PURPOSE
/usr	Installed software, shared libraries, include files, and static read-only program data. Important subdirectories include: - **/usr/bin**: *User commands*. - **/usr/sbin**: *System administration commands*. - **/usr/local**: *Locally customized software*.
/etc	Configuration files specific to this system.
/var	Variable data specific to this system that should persist between boots. Files that dynamically change (e.g. databases, cache directories, log files, printer-spooled documents, and website content) may be found under **/var**.
/run	Runtime data for processes started since the last boot. This includes process ID files and lock files, among other things. The contents of this directory are recreated on reboot. (This directory consolidates **/var/run** and **/var/lock** from older versions of Red Hat Enterprise Linux.)
/home	*Home directories* where regular users store their personal data and configuration files.
/root	Home directory for the administrative superuser, root.
/tmp	A world-writable space for temporary files. Files which have not been accessed, changed, or modified for 10 days are deleted from this directory automatically. Another temporary directory exists, **/var/tmp**, in which files that have not been accessed, changed, or modified in more than 30 days are deleted automatically.
/boot	Files needed in order to start the boot process.
/dev	Contains special *device files* which are used by the system to access hardware.

IMPORTANT

In Red Hat Enterprise Linux 7, four older directories in **/** now have identical contents as their counterparts located in **/usr**:

- **/bin** and **/usr/bin**.
- **/sbin** and **/usr/sbin**.
- **/lib** and **/usr/lib**.
- **/lib64** and **/usr/lib64**.

In older versions of Red Hat Enterprise Linux, these were distinct directories containing different sets of files. In RHEL 7, the directories in **/** are symbolic links to the matching directories in **/usr**.

REFERENCES

hier(7) man page

Filesystem Hierarchy Standard
http://www.pathname.com/fhs

SOLUTION

FILE SYSTEM HIERARCHY

Match the following items to their counterparts in the table.

DIRECTORY PURPOSE	LOCATION
This directory contains static, persistent system configuration data.	/etc
This is the system's root directory.	/
User home directories are located under this directory.	/home
This is the root account's home directory.	/root
This directory contains dynamic configuration data, such as FTP and websites.	/var
Regular user commands and utilities are located here.	/usr/bin
System administration binaries, for root use, are here.	/usr/sbin
Temporary files are stored here.	/tmp
Contains dynamic, non-persistent application runtime data.	/run
Contains installed software programs and libraries.	/usr

NAVIGATING PATHS

The **pwd** command displays the full path name of the current location, which helps determine appropriate syntax for reaching files using relative path names. The **ls** command lists directory contents for the specified directory or, if no directory is given, for the current directory.

```
[student@desktopX ~]$pwd
/home/student
[student@desktopX ~]$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
[student@desktopX ~]$
```

Use the **cd** command to change directories. With a working directory of **/home/student**, relative path syntax is shortest to reach the **Videos** subdirectory. The **Documents** subdirectory is then reached using absolute path syntax.

```
[student@desktopX ~]$cd Videos[student@desktopX Videos]$pwd
/home/student/Videos
[student@desktopX Videos]$ cd /home/student/Documents
[student@desktopX Documents]$ pwd
/home/student/Documents
[student@desktopX Documents]$ cd
[student@desktopX ~]$ pwd
/home/student
[student@desktopX ~]$
```

The shell program prompt displays, for brevity, only the last component of the current directory path. For **/home/student/Videos**, only **Videos** displays. At any time, return to the user's home directory using **cd** without specifying a destination. The prompt displays the *tilde* (**~**) character when the user's current directory is their home directory.

The **touch** command normally updates a file's timestamp to the current date and time without otherwise modifying it. This is useful for creating empty files, which can be used for practice, since "touching" a file name that does not exist causes the file to be created. Using **touch**, practice files are created in the **Documents** and **Videos** subdirectories.

```
[student@desktopX ~]$
touch Videos/blockbuster1.ogg
[student@desktopX ~]$
touch Videos/blockbuster2.ogg
[student@desktopX ~]$
touch Documents/thesis_chapter1.odf
[student@desktopX ~]$
touch Documents/thesis_chapter2.odf
[student@desktopX ~]$
```

The **`ls`** command has multiple options for displaying attributes on files. The most common and useful are **`-l`** (long listing format), **`-a`** (all files, includes *hidden* files), and **`-R`** (recursive, to include the contents of all subdirectories).

```
[student@desktopX ~]$ls -l
total 15
drwxr-xr-x.  2 student student 4096 Feb  7 14:02 Desktop
drwxr-xr-x.  2 student student 4096 Jan  9 15:00 Documents
drwxr-xr-x.  3 student student 4096 Jan  9 15:00 Downloads
drwxr-xr-x.  2 student student 4096 Jan  9 15:00 Music
drwxr-xr-x.  2 student student 4096 Jan  9 15:00 Pictures
drwxr-xr-x.  2 student student 4096 Jan  9 15:00 Public
drwxr-xr-x.  2 student student 4096 Jan  9 15:00 Templates
drwxr-xr-x.  2 student student 4096 Jan  9 15:00 Videos
[student@desktopX ~]$ ls -la
total 15









[student@desktopX ~]$
```

The two special directories at the top of the listing refer to the current directory (`.`) and the *parent* directory (`..`). These special directories exist in every directory on the system. Their usefulness will become apparent when file management commands are practiced.

IMPORTANT

File names beginning with a dot (`.`) indicate files *hidden* from normal view using **`ls`** and other commands. This is *not* a security feature. Hidden files keep necessary user configuration files from cluttering home directories. Many commands process hidden files only with specific command-line options, preventing one user's configuration from being accidentally copied to other directories or users.

To protect file *contents* from improper viewing requires the use of *file permissions*.

```
[student@desktopX ~]$ls -R
.:
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos

./Desktop:

./Documents:
thesis_chapter1.odf  thesis_chapter2.odf

./Downloads:

./Music:

./Pictures:

./Public:

./Templates:

./Videos:
blockbuster1.ogg  blockbuster2.ogg
[student@desktopX ~]$
```

The **`cd`** command has many options. A few are so useful as to be worth practicing early and using often. The command **`cd -`** changes directory to the directory where the user was *previous* to the current directory. Watch as this user takes advantage of this behavior to alternate between two directories, useful when processing a series of similar tasks.

```
[student@desktopX ~]$cd Videos[student@desktopX Videos]$pwd
/home/student/Videos
[student@desktopX Videos]$ cd /home/student/Documents
[student@desktopX Documents]$ pwd
/home/student/Documents
[student@desktopX Documents]$ cd -
[student@desktopX Videos]$ pwd
/home/student/Videos
[student@desktopX Videos]$ cd -
[student@desktopX Documents]$ pwd
/home/student/Documents
[student@desktopX Documents]$ cd -
```

```
[student@desktopX Videos]$ pwd
/home/student/Videos
[student@desktopX Videos]$ cd
[student@desktopX ~]$
```

The **cd ..** command uses the **..** hidden directory to move up one level to the *parent* directory, without needing to know the exact parent name. The other hidden directory (**.**) specifies the *current directory* on commands in which the current location is either the source or destination argument, avoiding the need to type out the directory's absolute path name.

```
[student@desktopX Videos]$pwd
/home/student/Videos
[student@desktopX Videos]$ cd .
[student@desktopX Videos]$ pwd
/home/student/Videos
[student@desktopX Videos]$ cd ..
[student@desktopX ~]$ pwd
/home/student
[student@desktopX ~]$ cd ..
[student@desktopX home]$ pwd
/home
[student@desktopX home]$ cd ..
[student@desktopX /]$ pwd
/
[student@desktopX /]$ cd
[student@desktopX ~]$ pwd
/home/student
[student@desktopX ~]$
```

REFERENCES

info libc 'file name resolution' (*GNU C Library Reference Manual*)

- Section 11.2.2 File name resolution

bash(1), **cd**(1), **ls**(1), **pwd**(1), **unicode**(7), and **utf-8**(7) man pages

UTF-8 and Unicode
http://www.utf-8.com/

SOLUTION

LOCATING FILES AND DIRECTORIES

Match the following items to their counterparts in the table.

ACTION TO ACCOMPLISH	COMMAND
List the current user's home directory (long format) in simplest syntax, when it is not the current location.	ls -l
Return to the current user's home directory.	cd
Determine the absolute path name of the current location.	pwd
Return to the most previous working directory.	cd -
Move up two levels from the current location.	cd ../..
List the current location (long format) with hidden files.	ls -al
Move to the binaries location, from any current location.	cd /bin
Move up to the parent of the current location.	cd ..
Move to the binaries location, from the root directory.	cd bin

MANAGING FILES USING COMMAND-LINE TOOLS

OBJECTIVES

After completing this section, students should be able to create, copy, link, move, and remove files and subdirectories in various directories.

COMMAND-LINE FILE MANAGEMENT

File management involves creating, deleting, copying, and moving files. Additionally, directories can be created, deleted, copied, and moved to help organize files logically. When working at the command line, file management requires awareness of the current working directory to choose either absolute or relative path syntax as most efficient for the immediate task.

File management commands

ACTIVITY	SINGLE SOURCE (NOTE)	MULTIPLE SOURCE (NOTE)
Copy file	cp file1 file2	cp file1 file2 file3 dir (4)
Move file	mv file1 file2 (1)	mv file1 file2 file3 dir (4)
Remove file	rm file1	rm -f file1 file2 file3 (5)
Create directory	mkdir dir	mkdir -p par1/par2/dir (6)
Copy directory	cp -r dir1 dir2 (2)	cp -r dir1 dir2 dir3 dir4 (4)
Move directory	mv dir1 dir2 (3)	mv dir1 dir2 dir3 dir4 (4)
Remove directory	rm -r dir1 (2)	rm -rf dir1 dir2 dir3 (5)
Remove empty directory	rmdir dir1	rmdir -p dir1/dir2/dir3
Note:	(1) (2) The result is a rename. (3) The "recursive" option is required to process a source directory. If dir2 exists, the result is a move. If dir2 doesn't exist, the result is a rename. (5) The last argument must be a directory. Use caution with "force" option; you will not be prompted to confirm your action. Use caution with "create parent" option; typing errors are not caught.	

Create directories

The **`mkdir`** command creates one or more directories or subdirectories, generating errors if the file name already exists or when attempting to create a directory in a parent directory that doesn't exist. The **`-p`** *parent* option creates missing parent directories for the requested destination. Be cautious when using **`mkdir -p`**, since accidental spelling mistakes create unintended directories without generating error messages.

In the following example, a user attempts to use **mkdir** to create a subdirectory named **Watched** in the existing **Videos** directory, but mistypes the directory name.

```
[student@desktopX ~]$mkdir Video/Watched
mkdir: cannot create directory `Video/Watched': No such file or directory
```

The **mkdir** failed because **Videos** was misspelled and the directory **Video** does not exist. If the user had used **mkdir** with the **-p** option, there would be no error and the user would end up with two directories, **Videos** and **Video**, and the **Watched** subdirectory would be created in the wrong place.

```
[student@desktopX ~]$mkdir Videos/Watched[student@desktopX ~]$cd
 Documents[student@desktopX Documents]$mkdir ProjectX ProjectY[student@desktopX
 Documents]$mkdir -p Thesis/Chapter1 Thesis/Chapter2 Thesis/
Chapter3[student@desktopX Documents]$cd[student@desktopX ~]$ls -R Videos Documents
Documents:
ProjectX  ProjectY  Thesis  thesis_chapter1.odf  thesis_chapter2.odf

Documents/ProjectX:

Documents/ProjectY:

Documents/Thesis:
Chapter1  Chapter2  Chapter3

Documents/Thesis/Chapter1:

Documents/Thesis/Chapter2:

Documents/Thesis/Chapter3:

Videos:
blockbuster1.ogg  blockbuster2.ogg  Watched

Videos/Watched:

[student@desktopX ~]$
```

The last **mkdir** created three Chapter*N* subdirectories with one command. The **-p** *parent* option created the missing parent directory **Thesis**.

Copy files

The **cp** command copies one or more files to become new, independent files. Syntax allows copying an existing file to a new file in the current or another directory, or copying multiple files into another directory. In any destination, new file names must be unique. If the new file name is not unique, the copy command will overwrite the existing file.

```
[student@desktopX ~]$cd Videos[student@desktopX Videos]$cp blockbuster1.ogg
 blockbuster3.ogg[student@desktopX Videos]$ls -l
total 0
-rw-rw-r--. 1 student student    0 Feb  8 16:23 blockbuster1.ogg
-rw-rw-r--. 1 student student    0 Feb  8 16:24 blockbuster2.ogg
-rw-rw-r--. 1 student student    0 Feb  8 19:02 blockbuster3.ogg
drwxrwxr-x. 2 student student 4096 Feb  8 23:35 Watched
```

```
[student@desktopX Videos]$
```

When copying multiple files with one command, the last argument must be a directory. Copied files retain their original names in the new directory. Conflicting file names that exist at a destination may be overwritten. To protect users from accidentally overwriting directories with contents, multiple file **cp** commands ignore directories specified as a source. Copying non-empty directories, with contents, requires the **-r** *recursive* option.

```
[student@desktopX Videos]$cd ../Documents[student@desktopX Documents]$cp
 thesis_chapter1.odf thesis_chapter2.odf Thesis ProjectX
cp: omitting directory `Thesis'
[student@desktopX Documents]$ cp -r Thesis ProjectX
[student@desktopX Documents]$ cp thesis_chapter2.odf Thesis/Chapter2/
[student@desktopX Documents]$ ls -R
.:
ProjectX  ProjectY  Thesis  thesis_chapter1.odf  thesis_chapter2.odf

./ProjectX:
Thesis  thesis_chapter1.odf  thesis_chapter2.odf

./ProjectX/Thesis:

./ProjectY:

./Thesis:
Chapter1  Chapter2  Chapter3

./Thesis/Chapter1:

./Thesis/Chapter2:
thesis_chapter2.odf

./Thesis/Chapter3:
[student@desktopX Documents]$
```

In the first **cp** command, **Thesis** failed to copy, but **thesis_chapter1.odf** and **thesis_chapter2.odf** succeeded. Using the **-r** *recursive* option, copying **Thesis** succeeded.

Move files

The **mv** command renames files in the same directory, or relocates files to a new directory. File contents remain unchanged. Files moved to a different file system require creating a new file by copying the source file, then deleting the source file. Although normally transparent to the user, large files may take noticeably longer to move.

```
[student@desktopX Videos]$cd ../Documents[student@desktopX Documents]$ls -l
total 0
-rw-rw-r--. 1 student student    0 Feb  8 16:24 thesis_chapter1.odf
-rw-rw-r--. 1 student student    0 Feb  8 16:24 thesis_chapter2.odf
[student@desktopX Documents]$ mv thesis_chapter2.odf thesis_chapter2_reviewed.odf
[student@desktopX Documents]$ mv thesis_chapter1.odf Thesis/Chapter1
[student@desktopX Documents]$ ls -lR
.:
total 16
drwxrwxr-x. 2 student student 4096 Feb 11 11:58 ProjectX
```

```
drwxrwxr-x. 2 student student 4096 Feb 11 11:55 ProjectY
drwxrwxr-x. 5 student student 4096 Feb 11 11:56 Thesis
-rw-rw-r--. 1 student student    0 Feb 11 11:54 thesis_chapter2_reviewed.odf

./ProjectX:
total 0
-rw-rw-r--. 1 student student 0 Feb 11 11:58 thesis_chapter1.odf
-rw-rw-r--. 1 student student 0 Feb 11 11:58 thesis_chapter2.odf

./ProjectX/Thesis:
total 0

./ProjectY:
total 0

./Thesis:
total 12
drwxrwxr-x. 2 student student 4096 Feb 11 11:59 Chapter1
drwxrwxr-x. 2 student student 4096 Feb 11 11:56 Chapter2
drwxrwxr-x. 2 student student 4096 Feb 11 11:56 Chapter3

./Thesis/Chapter1:
total 0
-rw-rw-r--. 1 student student 0 Feb 11 11:54 thesis_chapter1.odf

./Thesis/Chapter2:
total 0
-rw-rw-r--. 1 student student 0 Feb 11 11:54 thesis_chapter2.odf

./Thesis/Chapter3:
total 0
[student@desktopX Documents]$
```

The first **mv** command is an example of renaming a file. The second causes the file to be relocated to another directory.

Remove files and directories

Default syntax for **rm** deletes files, but not directories. Deleting a directory, and potentially many subdirectories and files below it, requires the **-r** *recursive* option. There is no command-line undelete feature, nor a trash bin from which to restore.

```
[student@desktopX Documents]$pwd
/home/student/Documents
[student@desktopX Documents]$ rm thesis_chapter2_reviewed.odf
[student@desktopX Documents]$ rm Thesis/Chapter1
rm: cannot remove `Thesis/Chapter1': Is a directory
[student@desktopX Documents]$ rm -r Thesis/Chapter1
[student@desktopX Documents]$ ls -l Thesis
total 8
drwxrwxr-x. 2 student student 4096 Feb 11 12:47 Chapter2
drwxrwxr-x. 2 student student 4096 Feb 11 12:48 Chapter3
[student@desktopX Documents]$ rm -ri Thesis
rm: descend into directory `Thesis'? y
rm: descend into directory `Thesis/Chapter2'? y
rm: remove regular empty file `Thesis/Chapter2/thesis_chapter2.odf'? y
```

```
rm: remove directory `Thesis/Chapter2'? y
rm: remove directory `Thesis/Chapter3'? y
rm: remove directory `Thesis'? y
[student@desktopX Documents]$
```

After **rm** failed to delete the **Chapter1** directory, the **-r** *recursive* option succeeded. The last **rm** command parsed into each subdirectory first, individually deleting contained files before removing each now-empty directory. Using **-i** will interactively prompt for each deletion. This is essentially the opposite of **-f** which will force the deletion without prompting the user.

The **rmdir** command deletes directories only if empty. Removed directories cannot be undeleted.

```
[student@desktopX Documents]$pwd
/home/student/Documents
[student@desktopX Documents]$ rmdir  ProjectY
[student@desktopX Documents]$ rmdir  ProjectX
rmdir: failed to remove `ProjectX': Directory not empty
[student@desktopX Documents]$ rm -r ProjectX
[student@desktopX Documents]$ ls -lR
.:
total 0
[student@desktopX Documents]$
```

The **rmdir** command failed to delete non-empty **ProjectX**, but **rm -r** succeeded.

REFERENCES

cp(1), **mkdir**(1), **mv**(1), **rm**(1), and **rmdir**(1) man pages

GUIDED EXERCISE

COMMAND-LINE FILE MANAGEMENT

In this lab, you will practice efficient techniques for creating and organizing files using directories and file copies.

OUTCOMES

Students will practice creating, rearranging, and deleting files.

Log into your student account on serverX. Begin in your home directory.

1. In your home directory, create sets of empty practice files to use for the remainder of this lab. If the intended command is not immediately recognized, students are expected to use the guided solution to see and practice how the task is accomplished. Use the shell tab completion to locate and complete path names more easily.

   Create six files with names of the form **songX.mp3**.

   Create six files with names of the form **snapX.jpg**.

   Create six files with names of the form **filmX.avi**.

   In each set, replace X with the numbers 1 through 6.

```
[student@serverX ~]$
touch song1.mp3 song2.mp3 song3.mp3 song4.mp3 song5.mp3 song6.mp3
[student@serverX ~]$
touch snap1.jpg snap2.jpg snap3.jpg snap4.jpg snap5.jpg snap6.jpg
[student@serverX ~]$
touch film1.avi film2.avi film3.avi film4.avi film5.avi film6.avi
[student@serverX ~]$
ls -l
```

2. From your home directory, move the song files into your **Music** subdirectory, the snapshot files into your **Pictures** subdirectory, and the movie files into your **Videos** subdirectory.

   When distributing files from one location to many locations, first change to the directory containing the *source* files. Use the simplest path syntax, absolute or relative, to reach the destination for each file management task.

```
[student@serverX ~]$
mv song1.mp3 song2.mp3 song3.mp3 song4.mp3 song5.mp3 song6.mp3 Music
[student@serverX ~]$
mv snap1.jpg snap2.jpg snap3.jpg snap4.jpg snap5.jpg snap6.jpg
Pictures
[student@serverX ~]$
mv film1.avi film2.avi film3.avi film4.avi film5.avi film6.avi Videos
[student@serverX ~]$
ls -l Music Pictures Videos
```

3. In your home directory, create three subdirectories for organizing your files into projects. Call these directories **friends**, **family**, and **work**. Create all three with one command.

   You will use these directories to rearrange your files into projects.

```
[student@serverX ~]$
mkdir friends family work
[student@serverX ~]$
ls -l
```

4. You will collect some of the new files into the project directories for family and friends. Use as many commands as needed. You do not have to use only one command as in the example. For each project, first change to the project directory, then copy the source files into this directory. You are making copies, since you will keep the originals after giving these projects to family and friends.

   Copy files (all types) containing numbers 1 and 2 to the friends folder.

   Copy files (all types) containing numbers 3 and 4 to the family folder.

   When collecting files from multiple locations into one location, change to the directory that will contain the *destination* files. Use the simplest path syntax, absolute or relative, to reach the source for each file management task.

```
[student@serverX ~]$
cd friends
[student@serverX friends]$
cp ~/Music/song1.mp3 ~/Music/song2.mp3 ~/Pictures/snap1.jpg ~/
Pictures/snap2.jpg ~/Videos/film1.avi ~/Videos/film2.avi .
[student@serverX friends]$
ls -l
[student@serverX friends]$
cd ../family
[student@serverX family]$
cp ~/Music/song3.mp3 ~/Music/song4.mp3 ~/Pictures/snap3.jpg ~/
Pictures/snap4.jpg ~/Videos/film3.avi ~/Videos/film4.avi .
[student@serverX family]$
ls -l
```

5. For your work project, you will create additional copies.

```
[student@serverX family]$ cd ../work[student@serverX work]$ cp ~/Music/song5.mp3
 ~/Music/song6.mp3 ~/Pictures/snap5.jpg ~/Pictures/snap6.jpg ~/Videos/film5.avi ~/
Videos/film6.avi .[student@serverX work]$ ls -l
```

6. Your projects are now done. Time to clean up the projects.

   Change to your home directory. Attempt to delete both the family and friends projects with a single **rmdir** command.

```
[student@serverX work]$ cd[student@serverX ~]$ rmdir family friends
rmdir: failed to remove `family': Directory not empty
rmdir: failed to remove `friends': Directory not empty
```

Using the **rmdir** command should fail since both directories are non-empty.

7. Use another command that will succeed in deleting both the family and friends folders.

```
[student@serverX ~]$
rm -r family friends
[student@serverX ~]$
ls -l
```

8. Delete all the files in the work project, but do not delete the work directory.

```
[student@serverX ~]$
cd work
[student@serverX work]$
rm song5.mp3 song6.mp3 snap5.jpg snap6.jpg film5.avi film6.avi
[student@serverX work]$
ls -l
```

9. Finally, from your home directory, use the **rmdir** command to delete the work directory. The command should succeed now that it is empty.

```
[student@serverX work]$
cd
[student@serverX ~]$
rmdir work
[student@serverX ~]$
ls -l
```

MATCHING FILE NAMES USING PATH NAME EXPANSION

OBJECTIVES

After completing this section, students should be able to use meta-characters and expansion techniques to improve file management processing efficiency.

FILE GLOBBING: PATH NAME EXPANSION

The Bash shell has a path name-matching capability historically called *globbing*, abbreviated from the "global command" file path expansion program of early UNIX. The Bash globbing feature, commonly called *pattern matching* or "wildcards", makes managing large numbers of files easier. Using *meta-characters* that "expand" to match file and path names being sought, commands perform on a focused set of files at once.

Pattern matching

Globbing is a shell command-parsing operation that expands a wildcard pattern into a list of matching path names. Command-line meta-characters are replaced by the match list prior to command execution. Patterns, especially square-bracketed character classes, that do not return matches display the original pattern request as literal text. The following are common meta-characters and pattern classes.

PATTERN	MATCHES
*	Any string of zero or more characters.
?	Any single character.
~	The current user's home directory.
~username	User *username*'s home directory.
~+	The current working directory.
~-	The previous working directory.
[*abc...*]	Any one character in the enclosed class.
[!*abc...*]	Any one character *not* in the enclosed class.
[^*abc...*]	Any one character *not* in the enclosed class.
[[:alpha:]]	Any alphabetic character.[1]
[[:lower:]]	Any lower-case character.[1]
[[:upper:]]	Any upper-case character.[1]
[[:alnum:]]	Any alphabetic character or digit.[1]
[[:punct:]]	Any printable character not a space or alphanumeric.[1]
[[:digit:]]	Any digit, `0-9`.[1]

PATTERN	MATCHES
[[:space:]]	Any one whitespace character; may include tabs, newline, or carriage returns, and form feeds as well as space.[1]
Note	[1] pre-set POSIX character class; adjusts for current locale.

A sample set of files is useful to demonstrate expansion.

```
[student@desktopX ~]$mkdir glob; cd glob[student@desktopX glob]$touch alfa bravo
 charlie delta echo able baker cast dog easy[student@desktopX glob]$ls
able  alfa  baker  bravo  cast  charlie  delta  dog  easy  echo
[student@desktopX glob]$
```

First, simple pattern matches using ***** and **?**.

```
[student@desktopX glob]$ls a*
able  alfa
[student@desktopX glob]$ ls *a*
able  alfa  baker  bravo  cast  charlie  delta  easy
[student@desktopX glob]$ ls [ac]*
able  alfa  cast  charlie
[student@desktopX glob]$ ls ????
able  alfa  cast  easy  echo
[student@desktopX glob]$ ls ?????
baker  bravo  delta
[student@desktopX glob]$
```

Tilde expansion

The tilde character (**~**), when followed by a slash delimiter, matches the current user's home directory. When followed by a string of characters up to a slash, it will be interpreted as a username, if one matches. If no username matches, then an actual tilde followed by the string of characters will be returned.

```
[student@desktopX glob]$ls ~/glob
able  alfa  baker  bravo  cast  charlie  delta  dog  easy  echo
[student@desktopX glob]$ echo ~/glob
/home/student/glob
[student@desktopX glob]$
```

Brace expansion

Brace expansion is used to generate discretionary strings of characters. Braces contain a comma-separated list of strings, or a sequence expression. The result includes the text preceding or following the brace definition. Brace expansions may be nested, one inside another.

```
[student@desktopX glob]$echo {Sunday,Monday,Tuesday,Wednesday}.log
Sunday.log Monday.log Tuesday.log Wednesday.log
[student@desktopX glob]$ echo file{1..3}.txt
file1.txt file2.txt file3.txt
[student@desktopX glob]$ echo file{a..c}.txt
filea.txt fileb.txt filec.txt
[student@desktopX glob]$ echo file{a,b}{1,2}.txt
```

```
filea1.txt filea2.txt fileb1.txt fileb2.txt
[student@desktopX glob]$ echo file{a{1,2},b,c}.txt
filea1.txt filea2.txt fileb.txt filec.txt
[student@desktopX glob]$
```

Command substitution

Command substitution allows the output of a command to replace the command itself. Command substitution occurs when a command is enclosed with a beginning dollar sign and parenthesis, **$(*command*)**, or with backticks, **`*command*`**. The form with backticks is older, and has two disadvantages: 1) it can be easy to visually confuse backticks with single quote marks, and 2) backticks cannot be nested inside backticks. The **$(*command*)** form can nest multiple command expansions inside each other.

```
[student@desktopX glob]$echo Today is `date +%A`.
Today is Wednesday.
[student@desktopX glob]$ echo The time is $(date +%M) minutes past $(date +%l%p).
The time is 26 minutes past 11AM.
[student@desktopX glob]$
```

Protecting arguments from expansion

Many characters have special meaning in the Bash shell. To ignore meta-character special meanings, *quoting* and *escaping* are used to protect them from shell expansion. The backslash (****) is an escape character in Bash, protecting the single following character from special interpretation. To protect longer character strings, single (**'**) or double quotes (**"**) are used to enclose strings.

Use double quotation marks to suppress globbing and shell expansion, but still allow command and variable substitution. Variable substitution is conceptually identical to command substitution, but may use optional brace syntax.

```
[student@desktopX glob]$host=$(hostname -s); echo $host
desktopX
[student@desktopX glob]$ echo "***** hostname is ${host} *****"
***** hostname is desktopX *****
[student@desktopX glob]$ echo Your username variable is \$USER.
Your username variable is $USER.
[student@desktopX glob]$
```

Use single quotation marks to interpret *all* text literally. Observe the difference, on both screen and keyboard, between the single quote (**'**) and the command substitution backtick (**`**). Besides suppressing globbing and shell expansion, quotations direct the shell to additionally suppress command and variable substitution. The question mark is a meta-character that also needed protection from expansion.

```
[student@desktopX glob]$echo "Will variable $host evaluate to $(hostname -s)?"
Will variable desktopX evaluate to desktopX?
[student@desktopX glob]$ echo 'Will variable $host evaluate to $(hostname -s)?'
Will variable $host evaluate to $(hostname -s)?
[student@desktopX glob]$
```

SOLUTION

PATH NAME EXPANSION

Match the following items to their counterparts in the table.

REQUESTED MATCH TO FIND	PATTERNS
Only filenames beginning with "b"	b*
Only filenames ending in "b"	*b
Only filenames containing a "b"	*b*
Only filenames where first character is not "b"	[!b]*
Only filenames at least 3 characters in length	???*
Only filenames that contain a number	*[[:digit:]]*
Only filenames that begin with an upper-case letter	[[:upper:]]*

SOLUTION

MANAGING FILES WITH SHELL EXPANSION

PERFORMANCE CHECKLIST

In this lab, you will create, move, and remove files and folders using a variety of file name matching shortcuts.

OUTCOMES

Familiarity and practice with many forms of wildcards for locating and using files.

Perform the following steps on serverX unless directed otherwise. Log in as **student** and begin the lab in the home directory.

1. To begin, create sets of empty practice files to use in this lab. If an intended shell expansion shortcut is not immediately recognized, students are expected to use the solution to learn and practice. Use shell tab completion to locate file path names easily.

   Create a total of 12 files with names **tv_seasonX_episodeY.ogg**. Replace *X* with the season number and *Y* with that season's episode, for two seasons of six episodes each.

```
[student@serverX ~]$
touch tv_season{1..2}_episode{1..6}.ogg
[student@serverX ~]$
```

```
[student@serverX ~]$
touch mystery_chapter{1..8}.odf
[student@serverX ~]$
ls -l
```

2. As the author of a successful series of mystery novels, your next bestseller's chapters are being edited for publishing. Create a total of eight files with names **mystery_chapterX.odf**. Replace *X* with the numbers 1 through 8.

3. To organize the TV episodes, create two subdirectories named **season1** and **season2** under the existing **Videos** directory. Use one command.

```
[student@serverX ~]$
mkdir Videos/season{1..2}
[student@serverX ~]$
ls -lR
```

CREATING, VIEWING, AND EDITING TEXT FILES

Goal- To create, view, and edit text files from command output or in an editor.

Sections

- Redirecting Output to a File or Program (and Practice)
- Editing Text Files from the Shell Prompt (and Practice)
- Editing Text Files with a Graphical Editor (and Practice)

LAB

CREATING, VIEWING, AND EDITING TEXT FILES

PERFORMANCE CHECKLIST

In this lab, you will edit a file using Vim's visual mode to simplify repetitive edits.

OUTCOMES

Familiarity with the utilities and techniques required to perform file editing. The final edited file will be a list of selected files and tabular data.

Perform the following steps on serverX unless directed otherwise. Log in as **`student`** and begin in the student's home directory.

1. Redirect a long listing of all content in student's home directory, including hidden directories and files, into a file named **`editing_final_lab.txt`**. Your home directory files may not exactly match those shown in the example graphics. This lab edits arbitrary lines and columns. The important outcome is to practice the visual selection process.
2. Edit the file using Vim, to take advantage of *visual mode*.
3. Remove the first three lines, since those lines are not normal file names. Enter line-based visual mode with upper case **`V`**.
4. Remove the permission columns for group and other on the first line. In this step, enter visual mode with lower case **`v`**, which allows selecting characters on a single line only.
5. Remove the permission columns for group and other on the remaining lines. This step will use a more efficient block selection visual mode to avoid having to repeat the single line edit multiple times. This time, enter visual mode with the control sequence **`Ctrl+v`**, which allows selecting a block of characters on multiple lines.
6. Remove the *group owner* column, leaving only one "student" column on all lines. Use the same block selection technique as the last step.
7. Remove the time column, but leave the month and day on all lines. Again, use the block selection visual mode.
8. Remove the **`Desktop`** and **`Public`** rows. This time, enter visual mode with upper case **`V`**, which automatically selects full lines.
9. Save and exit. Make a backup, using the date (in seconds) to create a unique file name.
10. Mail the file contents as the message, not an attachment, to the user student.
11. Append a dashed line to the file to recognize the beginning of newer content.
12. Append a full process listing, but only for processes owned by the current user **`student`** and running on the currently used terminal. View the process listing and send the listing to the file with one command line.
13. Confirm that the process listing is at the bottom of the lab file.

REDIRECTING OUTPUT TO A FILE OR PROGRAM

OBJECTIVES

After completing this section, students should be able to:

- Describe the technical terms standard input, standard output, and standard error.
- Use redirection characters to control output to files.
- Use piping to control output to other programs.

STANDARD INPUT, STANDARD OUTPUT, AND STANDARD ERROR

A running program, or *process*, needs to read input from somewhere and write output to the screen or to files. A command run from the shell prompt normally reads its input from the keyboard and sends its output to its terminal window.

A process uses numbered channels called *file descriptors* to get input and send output. All processes will have at least three file descriptors to start with. *Standard input* (channel 0) reads input from the keyboard. *Standard output* (channel 1) sends normal output to the terminal. *Standard error* (channel 2) sends error messages to the terminal. If a program opens separate connections to other files, it may use higher-numbered file descriptors.

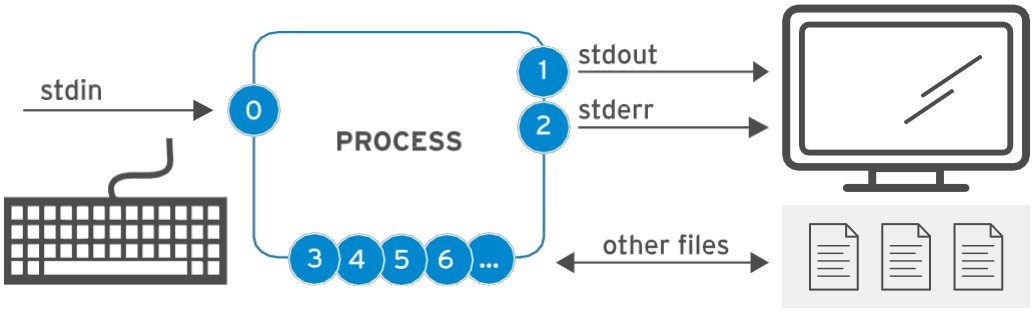


Figure 4.1: Process I/O channels (file descriptors)

Channels (File Descriptors)

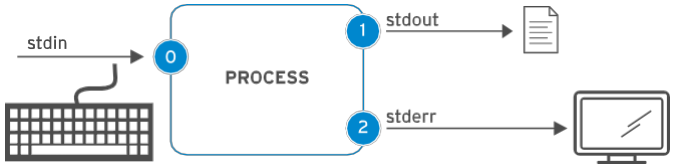
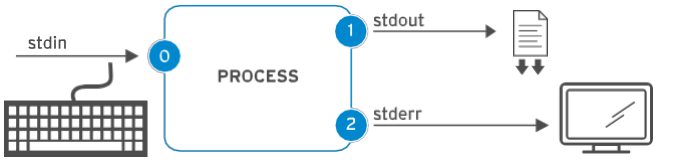
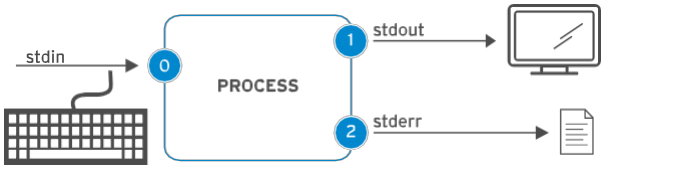
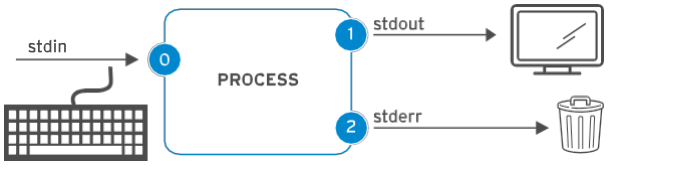
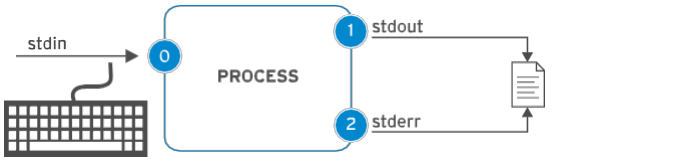
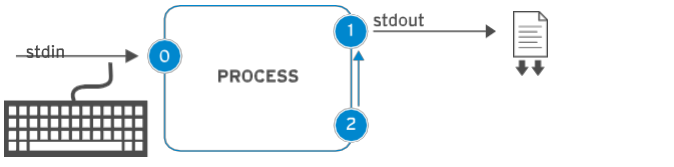
NUMBER	CHANNEL NAME	DESCRIPTION	DEFAULT CONNECTION	USAGE
0	`stdin`	Standard input	Keyboard	read only
1	`stdout`	Standard output	Terminal	write only
2	`stderr`	Standard error	Terminal	write only
3+	***`filename`***	Other files	*none*	read and/or write

REDIRECTING OUTPUT TO A FILE

I/O *redirection* replaces the default channel destinations with file names representing either output files or devices. Using redirection, process output and error messages normally sent to the terminal window can be captured as file contents, sent to a device, or discarded.

Redirecting **`stdout`** suppresses process output from appearing on the terminal. As seen in the following table, redirecting *only* **`stdout`** does not suppress **`stderr`** error messages from displaying on the terminal. If the file does not exist, it will be created. If the file does exist and the redirection is not one that appends to the file, the file's contents will be overwritten. The special file **`/dev/null`** quietly discards channel output redirected to it and is always an empty file.

Output Redirection Operators

USAGE	EXPLANATION	VISUAL AID
>file	redirect **`stdout`** to overwrite a file	
>>file	redirect **`stdout`** to append to a file	
2>file	redirect **`stderr`** to overwrite a file	
2>/ dev/null	discard **`stderr`** error messages by redirecting to **`/dev/null`**	
>file* 2>&1** ***&>file	redirect **`stdout`** and **`stderr`** to overwrite the same file	
>>file* 2>&1** ***&>>file	redirect **`stdout`** and **`stderr`** to append to the same file	

IMPORTANT

The order of redirection operations is important. The following sequence redirects standard output to **file** and then redirects standard error to the same place as standard output (**file**).

```
> file 2>&1
```

However, the next sequence does redirection in the opposite order. This redirects standard error to the default place for standard output (the terminal window, so no change) and *then* redirects only standard output to **file**.

```
2>&1 > file
```

Because of this, some people prefer to use the merging redirection operators:

&>file	instead of	**>file 2>&1**
&>>file	instead of	**>>file 2>&1** (in Bash 4 / RHEL 6 and later)

However, other system administrators and programmers who also use other shells related to **bash** ("Bourne-compatible shells") for scripting commands think that the newer merging redirection operators should be avoided, because they are not standardized or implemented in all of those shells and have other limitations.

The authors of this course take a neutral stance on this topic, and both syntaxes are likely to be encountered in the field.

Examples for output redirection

Many routine administration tasks are simplified by using redirection. Use the previous table to assist while considering the following examples:

- Save a timestamp for later reference.

```
[student@desktopX ~]$date > /tmp/saved-timestamp
```

- Copy the last 100 lines from a log file to another file.

```
[student@desktopX ~]$tail -n 100 /var/log/dmesg > /tmp/last-100-boot-messages
```

- Concatenate four files into one.

```
[student@desktopX ~]$cat file1 file2 file3 file4 > /tmp/all-four-in-one
```

- List the home directory's hidden and regular file names into a file.

```
[student@desktopX ~]$ls -a > /tmp/my-file-names
```

- Append output to an existing file.

```
[student@desktopX ~]$echo "new line of information" >> /tmp/many-lines-of-
information[student@desktopX ~]$diff previous-file current-file >> /tmp/tracking-
changes-made
```

- In the next examples, errors are generated since normal users are denied access to system directories. Redirect errors to a file while viewing normal command output on the terminal.

```
[student@desktopX ~]$find /etc -name passwd 2> /tmp/errors
```

- Save process output and error messages to separate files.

```
[student@desktopX ~]$find /etc -name passwd > /tmp/output 2> /tmp/errors
```

- Ignore and discard error messages.

```
[student@desktopX ~]$find /etc -name passwd > /tmp/output 2> /dev/null
```

- Store output and generated errors together.

```
[student@desktopX ~]$find /etc -name passwd &> /tmp/save-both
```

- Append output and generated errors to an existing file.

```
[student@desktopX ~]$find /etc -name passwd >> /tmp/save-both 2>&1
```

CONSTRUCTING PIPELINES

A *pipeline* is a sequence of one or more commands separated by `|`, the *pipe* character. A pipe connects the standard output of the first command to the standard input of the next command.

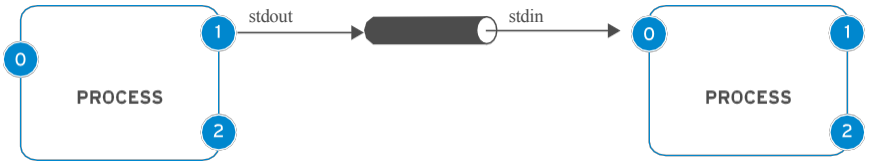


Figure 4.8: Process I/O piping

Pipelines allow the output of a process to be manipulated and formatted by other processes before it is output to the terminal. One useful mental image is to imagine that data is "flowing" through the pipeline from one process to another, being altered in slight ways by each command in the pipeline through which it passes.

NOTE

Pipelines and I/O redirection both manipulate standard output and standard input. *Redirection* sends standard output to or gets standard input from *files*. *Pipes* send standard output to or get standard input from another *process*.

Pipeline examples

This example takes the output of the **`ls`** command and uses **`less`** to display it on the terminal one screen at a time.

```
[student@desktopX ~]$ls -l /usr/bin | less
```

The output of the **ls** command is piped to **wc -l**, which counts the number of lines received from **ls** and prints that to the terminal.

```
[student@desktopX ~]$ls | wc -l
```

In this pipeline, **head** will output the first 10 lines of output from **ls -t**, with the final result redirected to a file.

```
[student@desktopX ~]$ls -t | head -n 10 > /tmp/ten-last-changed-files
```

Pipelines, redirection, and tee

When redirection is combined with a pipeline, the shell first sets up the entire pipeline, then it redirects input/output. What this means is that if output redirection is used in the *middle* of a pipeline, the output will go to the file and not to the next command in the pipeline.

In this example, the output of the **ls** command will go to the file, and **less** will display nothing on the terminal.

```
[student@desktopX ~]$ls > /tmp/saved-output | less
```

The **tee** command is used to work around this. In a pipeline, **tee** will copy its standard input to its standard output and will also redirect its standard output to the files named as arguments to the command. If data is imagined to be like water flowing through a pipeline, **tee** can be visualized as a "T" joint in the pipe which directs output in two directions.

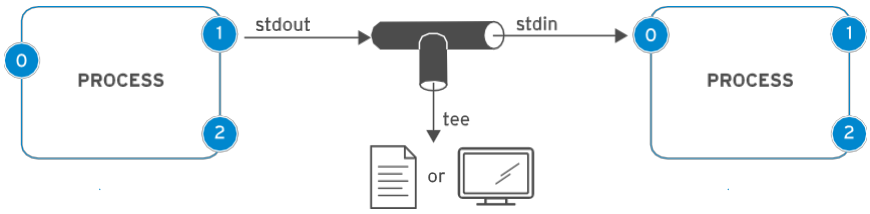


Figure 4.9: Process I/O piping with tee

Pipeline examples using the tee command

This example will redirect the output of the **ls** command to the file and will pass it to **less** to be displayed on the terminal a screen at a time.

```
[student@desktopX ~]$ls -l | tee /tmp/saved-output | less
```

If **tee** is used at the end of a pipeline, then the final output of a command can be saved and output to the terminal at the same time.

```
[student@desktopX ~]$ls -t | head -n 10 | tee /tmp/ten-last-changed-files
```

This more sophisticated example takes advantage of the fact that a special *device file* exists that represents the terminal. The name of the device file for a particular terminal can be determined by running the **tty** command at its shell prompt. Then **tee** can be used to redirect output to that file to display it on the terminal window, while standard output can be passed to some other program through a pipe. In this case, **mail** will e-mail the output to student@desktop1.example.com.

```
[student@desktopX ~]$tty
/dev/pts/0
[student@desktopX ~]$ ls -l | tee /dev/pts/0 | mail student@desktop1.example.com
```

IMPORTANT

Standard error can be redirected through a pipe, but the merging redirection operators (**&>** and **&>>**) can not be used to do this.

The following is the correct way to redirect both standard output and standard error through a pipe:

```
[student@desktopX ~]$ find -name / passwd 2>&1 | less
```

REFERENCES

info bash (*The GNU Bash Reference Manual*)

- Section 3.2.2: Pipelines
- Section 3.6: Redirections

info coreutils 'tee invocation' (*The GNU coreutils Manual*)

- Section 17.1: Redirect output to multiple files or processes

bash(1), **cat**(1), **head**(1), **less**(1), **mail**(1), **tee**(1), **tty**(1), **wc**(1) man pages

SOLUTION

I/O REDIRECTION AND PIPELINES

Match the following items to their counterparts in the table.

RESULT NEEDED	REDIRECTION SYNTAX USED
Display command output to terminal, ignore all errors.	2>/dev/null
Send command output to file; errors to different file.	>file 2>file2
Send output and errors to the same new, empty file.	&>file
Send output and errors to the same file, but preserve existing file content.	>>file 2>&1
Run a command, but throw away all possible terminal displays.	&>/dev/null
Send command output to both the screen and a file at the same time.	| tee file
Run command, save output in a file, discard error messages.	> file 2> /dev/null

EDITING TEXT FILES WITH A GRAPHICAL EDITOR

OBJECTIVES

After completing this section, students should be able to:

- Edit text files with gedit.
- Copy text between graphical windows.

EDITING FILES WITH GEDIT

The **gedit** application is a full-featured text editor for the GNOME desktop environment. Launch **gedit** by selecting Applications → Accessories → gedit from the GNOME menu. Like other graphical applications, **gedit** can be started without navigating the menu. Press **Alt+F2** to open the Enter a Command dialog box. Type **gedit** and press **Enter**.

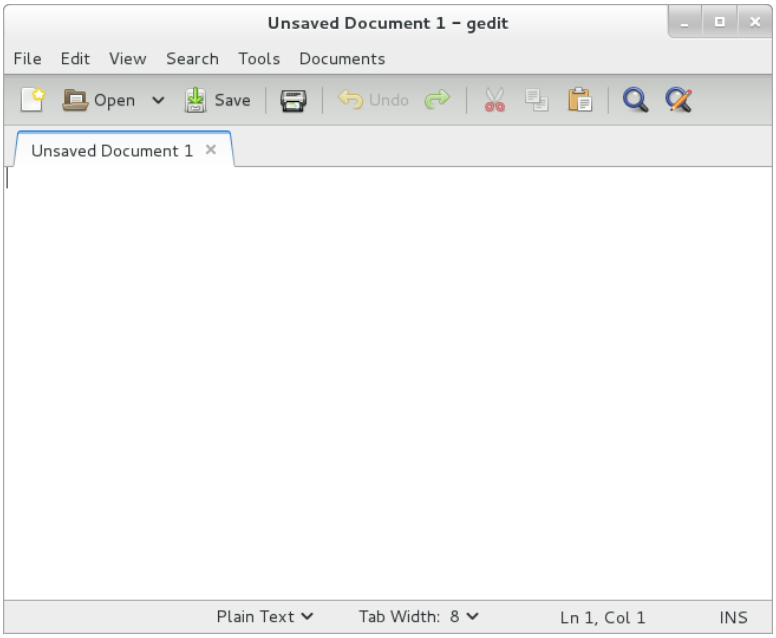


Figure 4.11: gedit text editor

GNOME Help includes a **gedit** help guide, which may be found by selecting Applications → Favorites → Help from the GNOME menu. Then select Go → All Documents to view the list of graphical applications. Scroll down to select the gedit Text Editor hyperlink.

Basic gedit Keystrokes

Perform many file management tasks using **gedit**'s menu:

- To create a new file in gedit, click the blank paper toolbar icon, or select File → New (**Ctrl+n**) from the menu.
- To save a file, click the disk-drive save toolbar icon, or select File → Save (**Ctrl+s**) from the menu.

- To open an existing file, click the Open toolbar icon, or select File → Open (`Ctrl+o`) from the menu. The Open Files dialog window will display from which users can locate and select the file to open.

Multiple files may be opened simultaneously, each with a unique tab under the menu bar. Tabs display a file name after being saved the first time.

COPYING TEXT BETWEEN GRAPHICAL WINDOWS

Text can be copied between documents, text windows, and command windows in the graphical environment. Selected text is duplicated using *copy and paste* or moved using *cut and paste*. Whether cut or copied, the text is held in memory for pasting into another location.

To select text:

- Click and hold the left mouse button before the first character desired.
- Drag the mouse over and down until all required text is in a single highlighted selection, then release the left button. Do not click again with the left button, as that deselects the text.

To paste the selection, multiple methods can accomplish the same result. In the first method:

- Click the right mouse button anywhere on the text area just selected.
- From the resulting context menu, select *either* cut or copy.
- Move the mouse to the window or document where the text is to be placed, click the left mouse button to position where the text should go, and click the right mouse button again, now choosing paste.

Here is a shorter mouse technique to practice:

- First, select the text.
- Hover the mouse over the destination window and click the center mouse button, just once, to paste the text at the cursor.

This last method can only copy, not cut. The original text remains selected and can be deleted. As with other methods, the text remains in memory and can be repeatedly pasted.

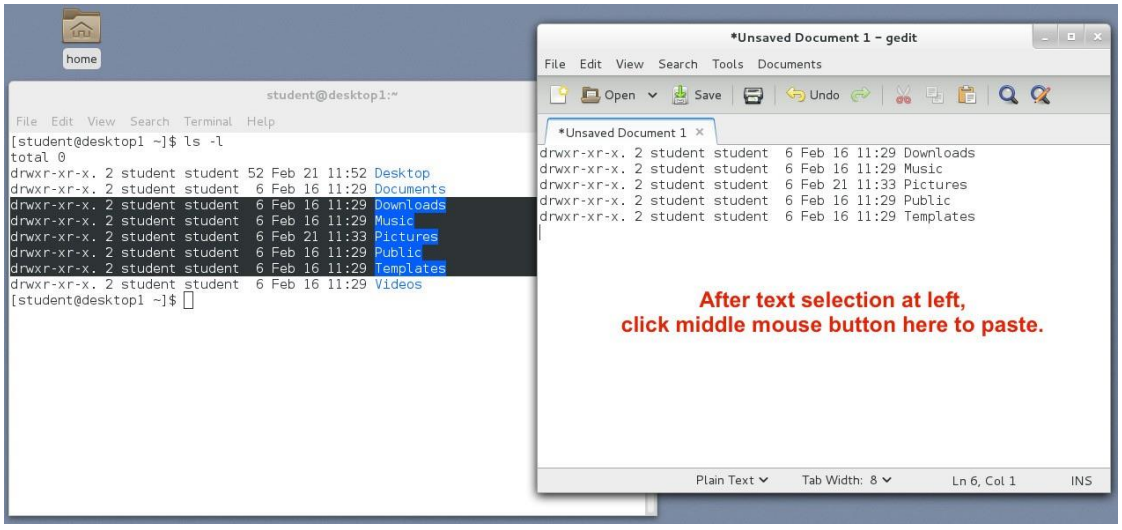


Figure 4.12: Select and paste using middle mouse button

The keyboard shortcut method can also be used in graphical applications:

- First, select the text.
- Use **Ctrl+x** to cut or **Ctrl+c** to copy the text.
- Click the location where the text is to be placed to position the cursor.
- Use **Ctrl+v** to paste.

NOTE

Ctrl+c and **Ctrl+v** will not copy and paste within a terminal window. **Ctrl+c** will actually terminate the current running process within a terminal window. To copy and paste within a terminal window, use **Ctrl+Shift+c** and **Ctrl+Shift+v**.

REFERENCES

gedit(1) man page

gedit Text Editor

- **yelp help:gedit**

gedit Wiki
https://wiki.gnome.org/Apps/Gedit

GUIDED EXERCISE

COPYING TEXT BETWEEN WINDOWS

In this lab, you will edit a file with gedit, selecting text and pasting it into the editor.

OUTCOMES

An edited list of the configuration files found in the user's home directory.

Perform the following steps on serverX unless directed otherwise. Log in as **student** and begin in student's home directory.

1. Redirect a long listing of all home directory files, including hidden, into a file named gedit_lab.txt. Confirm that the file contains the listing.

```
[student@serverX ~]$
cd
[student@serverX ~]$
ls -al > gedit_lab.txt
[student@serverX ~]$
cat gedit_lab.txt
```

2. Open the file with the **gedit** text editor. Include the ending ampersand so that the shell prompt can return while **gedit** is running.

```
[student@serverX ~]$
gedit gedit_lab.txt &
```

3. Insert the date at the top of your file document.

   3.1. In the shell command window, display today's date with day of the week, month, date, and year.

```
[student@serverX ~]$ date +%A", "%B" "%d", "%Y
```

```
Friday, February 21, 2014
```

3.2. Select the text using the mouse.

3.3. Insert the text at the top of the file document. Switch to the **gedit** window. Using arrow keys, place the cursor at the upper-left corner of the document. Press the middle mouse button to paste the text.

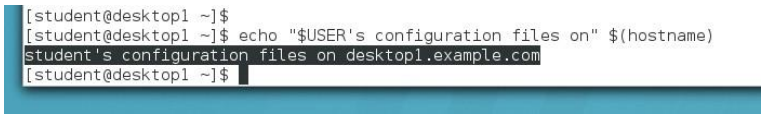
3.4. Press **Enter** one or more times at the end of the inserted text to open blank lines above the file listing.

4. Insert a description for this document, including your username and host name, on line 2.

4.1. In the shell command window, create descriptive text using shell expansion concepts to include the username and host name where the file list was generated.

```
[student@serverX ~]$ echo "$USER's configuration files on" $(hostname)
student's configuration files on serverX.example.com
```

4.2. Select the text using the mouse.



4.3. Insert the text on the second line of the file document. Switch to the **gedit** window. Using the arrow keys, place the cursor at the second line's leftmost character. Press the middle mouse button to paste the text.

4.4. Press **Enter** or **Delete**, as necessary, to maintain blank lines above the file listing.

5. Remove file lines that are not hidden configuration files or directories.

5.1. Remove the “total” line at the beginning of the listing.

5.2. Remove the two lines representing the current directory and the parent directory.

5.3. Remove lines for file names that do *not* start with a dot. Do not edit or remove lines for hidden files or directories that begin with a dot.

SOLUTION

CREATING, VIEWING, AND EDITING TEXT FILES

PERFORMANCE CHECKLIST

In this lab, you will edit a file using Vim's visual mode to simplify repetitive edits.

OUTCOMES

Familiarity with the utilities and techniques required to perform file editing. The final edited file will be a list of selected files and tabular data.

Perform the following steps on serverX unless directed otherwise. Log in as **student** and begin in the student's home directory.

1. Redirect a long listing of all content in student's home directory, including hidden directories and files, into a file named **editing_final_lab.txt**. Your home directory files may not exactly match those shown in the example graphics. This lab edits arbitrary lines and columns. The important outcome is to practice the visual selection process.

```
[student@serverX ~]$
cd
[student@serverX ~]$
ls -al > editing_final_lab.txt
```

2. Edit the file using Vim, to take advantage of *visual mode*.

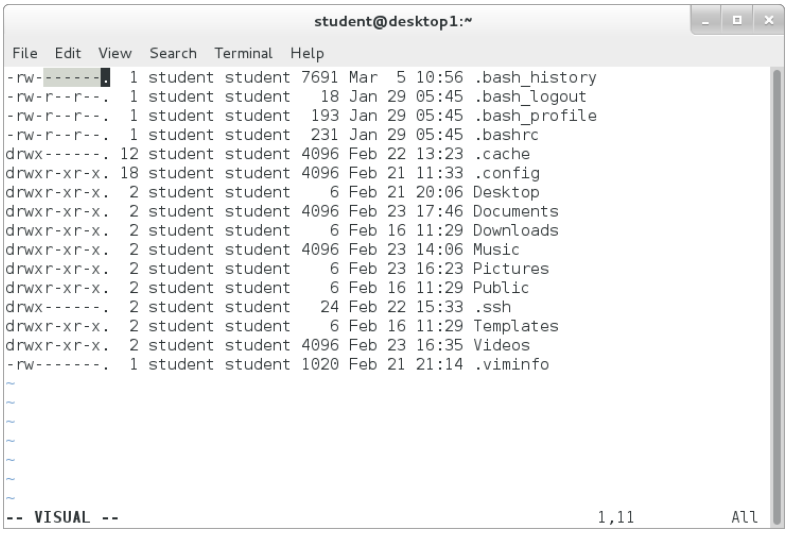
```
[student@serverX ~]$
vim editing_final_lab.txt
```

3. Remove the first three lines, since those lines are not normal file names. Enter line-based visual mode with upper case **V**.

   Use the arrow keys to position the cursor at the first character in the first row. Enter line-based visual mode with **V**. Move down using the down arrow key twice to select the first three rows. Delete the rows with **x**.

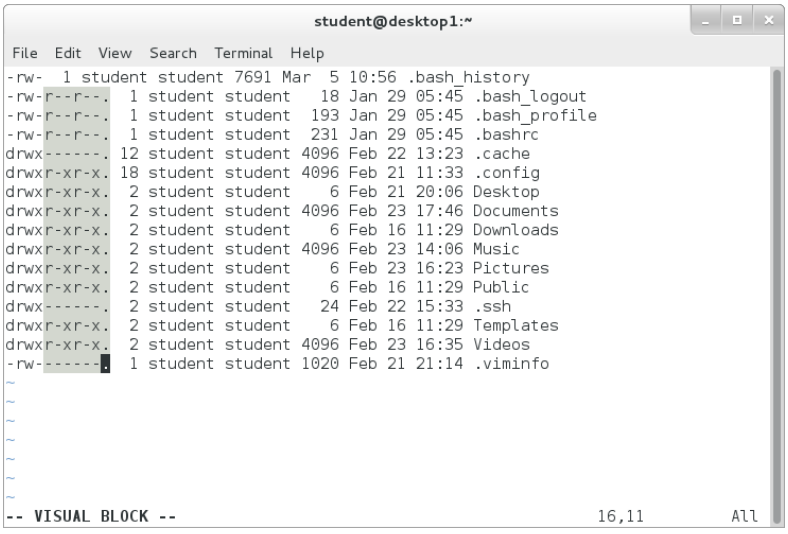
4. Remove the permission columns for group and other on the first line. In this step, enter visual mode with lower case **v**, which allows selecting characters on a single line only.

   Use the arrow keys to position the cursor at the first character. Enter visual mode with **v**. Use the arrow keys to position the cursor at the last character, as shown in the screenshot. Delete the selection with **x**.



5. Remove the permission columns for group and other on the remaining lines. This step will use a more efficient block selection visual mode to avoid having to repeat the single line edit multiple times. This time, enter visual mode with the control sequence **Ctrl+v**, which allows selecting a block of characters on multiple lines.

   Use the arrow keys to position the cursor at the first character. Enter visual mode with the control sequence **Ctrl+v**. Use the arrow keys to position the cursor at the last character of the column on the last line, as shown in the screenshot. Delete the selection with **x**.



LAB-4 MANAGING LOCAL LINUX USERS AND GROUPS

Goal- To manage local Linux users and groups and administer local password policies.

Sections

- Users and Groups (and Practice)
- Gaining Superuser Access (and Practice)
- Managing Local User Accounts (and Practice)
- Managing Local Group Accounts (and Practice)
- Managing User Passwords (and Practice)

LAB

MANAGING LOCAL LINUX USERS AND GROUPS

PERFORMANCE CHECKLIST

In this lab, you will define a default password policy, create a supplementary group of three new users, and modify the password policy of one user.

OUTCOMES

- A new group on serverX called **`consultants`**, including three new user accounts for Sam Spade, Betty Boop, and Dick Tracy.
- All new accounts should require that passwords be changed at first login and every 30 days thereafter.
- The new consultant accounts should expire at the end of the 90-day contract, and Betty Boop must change her password every 15 days.

Reset your serverX system.

1. Ensure that newly created users have passwords which must be changed every 30 days.
2. Create a new group named **`consultants`** with a GID of 40000.
3. Create three new users: **`sspade`**, **`bboop`**, and **`dtracy`**, with a password of **`default`** and add them to the supplementary group **`consultants`**. The primary group should remain as the user private group.
4. Determine the date 90 days in the future and set each of the three new user accounts to expire on that date.
5. Change the password policy for the **`bboop`** account to require a new password every 15 days.
6. Additionally, force all users to change their password on first login.
7. When you finish, run the **`lab localusers grade`** evaluation script to confirm you have done everything correctly.

USERS AND GROUPS

OBJECTIVES

After completing this section, students should be able to explain the role of users and groups on a Linux system and how they are understood by the computer.

WHAT IS A USER?

Every process (running program) on the system runs as a particular user. Every file is owned by a particular user. Access to files and directories are restricted by user. The user associated with a running process determines the files and directories accessible to that process.

The **id** command is used to show information about the current logged-in user. Basic information about another user can also be requested by passing in the username of that user as the first argument to the **id** command.

```
[student@desktopX ~]$ id
uid=1000(student) gid=1000(student) groups=1000(student),10(wheel)
 context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
```

To view the user associated with a file or directory, use the **ls -l** command. The third column shows the username:

```
[student@serverX ~]$ ls -l /tmp
drwx------. 2 gdm     gdm      4096 Jan 24 13:05 orbit-gdm
drwx------. 2 student student  4096 Jan 25 20:40 orbit-student
-rw-r--r--. 1 root    root    23574 Jan 24 13:05 postconf
```

To view process information, use the **ps** command. The default is to show only processes in the current shell. Add the **a** option to view all processes with a terminal. To view the user associated with a process, include the **u** option. The first column shows the username:

```
[student@serverX ~]$ ps au
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root        428  0.0  0.7 152768 14400 tty1     Ss+  Feb03   0:04 /usr/bin/Xorg
root        511  0.0  0.0 110012   812 ttyS0    Ss+  Feb03   0:00 /sbin/agetty
root       1805  0.0  0.1 116040  2580 pts/0    Ss   Feb03   0:00 -bash
root       2109  0.0  0.1 178468  2200 pts/0    S    Feb03   0:00 su - student
student    2110  0.0  0.1 116168  2864 pts/0    S    Feb03   0:00 -bash
student    3690  0.0  0.0 123368  1300 pts/0    R+   11:42   0:00 ps au
```

The output of the previous commands displays users by name, but internally the operating system tracks users by a *UID number*. The mapping of names to numbers is defined in databases of account information. By default, systems use a simple "flat file," the **/etc/passwd** file, to store information about local users. The format of **/etc/passwd** follows (seven colon-separated fields):

```
❶username:❷password:❸UID:❹GID:❺GECOS:❻/home/dir:❼shell
```

1. *username* is a mapping of a UID to a name for the benefit of human users.
2. *password* is where, historically, passwords were kept in an encrypted format. Today, they are stored in a separate file called **`/etc/shadow`**.
3. *UID* is a user ID, a number that identifies the user at the most fundamental level.
4. *GID* is the user's primary group ID number. Groups will be discussed in a moment.
5. *GECOS* field is arbitrary text, which usually includes the user's real name.
6. */home/dir* is the location of the user's personal data and configuration files.
7. *shell* is a program that runs as the user logs in. For a regular user, this is normally the program that provides the user's command line prompt.

WHAT IS A GROUP?

Like users, groups have a name and a number (GID). Local groups are defined in **`/etc/group`**.

Primary groups

- Every user has exactly one *primary group*.
- For local users, the primary group is defined by the GID number of the group listed in the fourth field of **`/etc/passwd`**.
- Normally, the primary group owns new files created by the user.
- Normally, the primary group of a newly created user is a newly created group with the same name as the user. The user is the only member of this *User Private Group* (UPG).

Supplementary groups

- Users may be a member of zero or more *supplementary groups*.
- The users that are supplementary members of local groups are listed in the last field of the group's entry in **`/etc/group`**. For local groups, user membership is determined by a comma-separated list of users found in the last field of the group's entry in **`/etc/group`**:

```
groupname:password:GID:list,of,users,in,this,group
```

- Supplementary group membership is used to help ensure that users have access permissions to files and other resources on the system.

REFERENCES

`id`(1), **`passwd`**(5), and **`group`**(5) man pages

`info libc` (*GNU C Library Reference Manual*)

- Section 29: Users and groups

(Note that the *glibc-devel* package must be installed for this info node to be available.)

SOLUTION

USER AND GROUP CONCEPTS

Match the items below to their counterparts in the table.

DESCRIPTION	KEYWORD
A number that identifies the user at the most fundamental level	UID
The program that provides the user's command line prompt	login shell
Location of local group information	`/etc/group`
Location of the user's personal files	home directory
A number that identifies the group at the most fundamental level	GID
Location of local user account information	`/etc/passwd`
The fourth field of `/etc/passwd`	primary group

GAINING SUPERUSER ACCESS

OBJECTIVES

After completing this section, students should be able to run commands as the superuser to administer a Linux system.

THE `root` USER

Most operating systems have some sort of *superuser*, a user that has all power over the system. This user in Red Hat Enterprise Linux is the **`root`** user. This user has the power to override normal privileges on the file system, and is used to manage and administer the system. In order to perform tasks such as installing or removing software and to manage system files and directories, a user must escalate privileges to the **`root`** user.

Most devices can only be controlled by **`root`**, but there are a few exceptions. For instance, removable devices, such as USB devices, are allowed to be controlled by a normal user. Thus, a non-root user is allowed to add and remove files and otherwise manage a removable device, but only root is allowed to manage "fixed" hard drives by default.

This unlimited privilege, however, comes with responsibility. **`root`** has unlimited power to damage the system: remove files and directories, remove user accounts, add backdoors, etc. If the **`root`** account is compromised, someone else would have administrative control of the system. Throughout this course, administrators will be encouraged to log in as a normal user and escalate privileges to **`root`** only when needed.

The **`root`** account on Linux is roughly equivalent to the local Administrator account on Windows. In Linux, most system administrators log into an unprivileged user account and use various tools to temporarily gain root privileges.

WARNING

One common practice on Windows in the past is for the local Administrator user to log in directly to perform system administrator duties. However, on Linux, it is recommended that system administrators *should not* log in directly as **`root`**. Instead, system administrators should log in as a non-root user, and use other mechanisms (**`su`**, **`sudo`**, or PolicyKit, for example) to temporarily gain superuser privileges.

By logging in as the administrative user, the entire desktop environment unnecessarily runs with administrative privileges. In that situation, any security vulnerability which would normally only compromise the user account has the potential to compromise the entire system.

In recent versions of Microsoft Windows, Administrator disabled by default, and features such as User Account Control (UAC) are used to limit administrative privileges for users until actually needed. In Linux, the PolicyKit system is the nearest equivalent to UAC.

SWITCHING USERS WITH `su`

The **`su`** command allows a user to switch to a different user account. If a username is not specified, the *root* account is implied. When invoked as a regular user, a prompt will display asking for the

password of the account you are switching to; when invoked as *root*, there is no need to enter the account password.

su [-] <username>

```
[student@desktopX ~]$
su -
Password:
redhat
[root@desktopX ~]#
```

The command **su *username*** starts a *non-login shell*, while the command **su - *username*** starts a *login* shell. The main distinction is **su -** sets up the shell environment as if this were a clean login as that user, while **su** just starts a shell as that user with the current environment settings.

In most cases, administrators want to run **su -** to get the user's normal settings. For more information, see the **bash**(1) man page.

NOTE

The **su** command is most frequently used to get a command line interface (shell prompt) which is running as another user, typically **root**. However, with the **-c** option, it can be used like the Windows utility **runas** to run an arbitrary program as another user. See **info su** for details.

RUNNING COMMANDS AS root WITH sudo

Fundamentally, Linux implements a very coarse-grained permissions model: *root* can do everything, other users can do nothing (systems-related). The common solution previously discussed is to allow standard users to temporarily "become *root*" using the **su** command. The disadvantage is that while acting as *root*, all the privileges (and responsibilities) of *root* are granted. Not only can the user restart the web server, but they can also remove the entire **/etc** directory. Additionally, all users requiring superuser privilege in this manner must know the **root** password.

The **sudo** command allows a user to be permitted to run a command as root, or as another user, based on settings in the **/etc/sudoers** file. Unlike other tools such as **su**, **sudo** requires users to enter their own password for authentication, not the password of the account they are trying to access. This allows an administrator to hand out fine-grained permissions to users to delegate system administration tasks, without having to hand out the *root* password.

For example, when **sudo** has been configured to allow the user *student* to run the command **usermod** as *root*, *student* could run the following command to lock a user account:

```
[student@serverX ~]$ sudo usermod -L username
[sudo] password for student: password
```

One additional benefit to using **sudo** is that all commands executed using **sudo** are logged by default to **/var/log/secure**.

```
[student@serverX ~]$ sudo tail /var/log/secure
...
Feb 19 15:23:36 localhost sudo: student : TTY=pts/0 ; PWD=/home/student ;
 USER=root ; COMMAND=/sbin/usermod -L student
```

```
Feb 19 15:23:36 localhost usermod[16325]: lock user 'student' password
Feb 19 15:23:47 localhost sudo: student : TTY=pts/0 ; PWD=/home/student ;
 USER=root ; COMMAND=/bin/tail /var/log/secure
```

In Red Hat Enterprise Linux 7, all members of group **wheel** can use **sudo** to run commands as any user, including **root**. The user will be prompted for their own password. This is a change from Red Hat Enterprise Linux 6 and earlier. Users who were members of group **wheel** did not get this administrative access by default in RHEL 6 and earlier.

To enable similar behavior on earlier versions of Red Hat Enterprise Linux, use **visudo** to edit the configuration file and uncomment the line allowing the group **wheel** to run all commands.

```
[root@desktopX ~]# cat /etc/sudoers
...Output omitted...
## Allows people in group wheel to run all commands
%wheel        ALL=(ALL)       ALL

## Same thing without a password
# %wheel  ALL=(ALL)       NOPASSWD: ALL
...Output omitted...
```

WARNING

RHEL 6 did not grant group **wheel** any special privileges by default. Sites which have been using this group may be surprised when RHEL 7 automatically grants all members of **wheel** full **sudo** privileges. This could lead to unauthorized users getting superuser access to RHEL 7 systems.

Historically, membership in group **wheel** has been used by Unix-like systems to grant or control superuser access.

Most system administration applications with a GUI use PolicyKit to prompt users for authentication and to manage root access. In Red Hat Enterprise Linux 7, PolicyKit may also prompt members of group **wheel** for their own password in order to get **root** privileges when using graphical tools. This is similar to the way in which they can use **sudo** to get those privileges at the shell prompt. PolicyKit grants these privileges based on its own configuration settings, separate from **sudo**. Advanced students may be interested in the **pkexec**(1) and **polkit**(8) man pages for details on how this system works, but it is beyond the scope of this course.

REFERENCES

su(1), **visudo**(8) and **sudo**(8) man pages

info libc (*GNU C Library Reference Manual*)

- Section 29.2: The Persona of a Process

(Note that the *glibc-devel* package must be installed for this info node to be available.)

GUIDED EXERCISE

RUNNING COMMANDS AS ROOT

In this lab, you will practice running commands as **root**.

OUTCOMES

Use the **su** with and without login scripts to switch users. Use **sudo** to run commands with privilege.

Reset your serverX system.

1. Log into the GNOME desktop on serverX as **student** with a password of **student**.

2. Open a window with a Bash prompt.
   Select Applications → Utilities → Terminal.

3. Explore characteristics of the current student login environment.
   3.1. View the user and group information and display the current working directory.

```
[student@serverX ~]$ id
uid=1000(student) gid=1000(student) groups=1000(student),10(wheel)
 context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
[student@serverX ~]$ pwd
/home/student
```

   3.2. View the variables which specify the home directory and the locations searched for executable files.

```
[student@serverX ~]$ echo $HOME
/home/student
[student@serverX ~]$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/home/student/.local/bin:/home/
student/bin
```

4. Switch to root without the dash and explore characteristics of the new environment.
   4.1. Become the **root** user at the shell prompt.

```
[student@serverX ~]$
su
Password:
redhat
```

4.2. View the user and group information and display the current working directory. Note the identity changed, but not the current working directory.

```
[root@serverX student]# id
uid=0(root) gid=0(root) groups=0(root)
 context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
[root@serverX student]# pwd
/home/student
```

4.3. View the variables which specify the home directory and the locations searched for executable files. Look for references to the student and root accounts.

```
[root@serverX student]# echo $HOME
/root
[root@serverX student]# echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/home/student/.local/bin:/home/
student/bin
```

4.4. Exit the shell to return to the **student** user.

```
[root@serverX student]# exit
exit
```

5. Switch to root with the dash and explore characteristics of the new environment.

5.1. Become the **root** user at the shell prompt. Be sure all the login scripts are also executed.

```
[student@serverX ~]$
su -
Password:
redhat
```

5.2. View the user and group information and display the current working directory.

```
[root@serverX ~]# id
uid=0(root) gid=0(root) groups=0(root)
 context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
[root@serverX ~]# pwd
/root
```

5.3. View the variables which specify the home directory and the locations searched for executable files. Look for references to the student and root accounts.

```
[root@serverX ~]# echo $HOME
/root
[root@serverX ~]# echo $PATH
```

```
/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/root/bin
```

5.4. Exit the shell to return to the **student** user.

```
[root@serverX ~]# exit
logout
```

6. Run several commands as student which require root access.

6.1. View the last 5 lines of the **/var/log/messages**.

```
[student@serverX ~]$ tail -5 /var/log/messages
tail: cannot open ‘/var/log/messages’ for reading: Permission denied
[student@serverX ~]$ sudo tail -5 /var/log/messages
Feb  3 15:07:22 localhost su: (to root) root on pts/0
Feb  3 15:10:01 localhost systemd: Starting Session 31 of user root.
Feb  3 15:10:01 localhost systemd: Started Session 31 of user root.
Feb  3 15:12:05 localhost su: (to root) root on pts/0
Feb  3 15:14:47 localhost su: (to student) root on pts/0
```

6.2. Make a backup of a configuration file in the **/etc** directory.

```
[student@serverX ~]$ cp /etc/motd /etc/motdOLD
cp: cannot create regular file ‘/etc/motdOLD’: Permission denied
[student@serverX ~]$ sudo cp /etc/motd /etc/motdOLD
```

6.3. Remove the **/etc/motdOLD** file that was just created.

```
[student@serverX ~]$ rm /etc/motdOLD
rm: remove write-protected regular empty file ‘/etc/motdOLD’? y
rm: cannot remove ‘/etc/motdOLD’: Permission denied
[student@serverX ~]$ sudo rm /etc/motdOLD
```

6.4. Edit a configuration file in the **/etc** directory.

```
[student@serverX ~]$ echo "Welcome to class" >> /etc/motd
-bash: /etc/motd: Permission denied
[student@serverX ~]$ sudo vim /etc/motd
```

MANAGING LOCAL USER ACCOUNTS

OBJECTIVES

After completing this section, students should be able to create, modify, lock, and delete locally defined user accounts.

MANAGING LOCAL USERS

A number of command-line tools can be used to manage local user accounts.

`useradd` creates users

- **`useradd`** ***`username`*** sets reasonable defaults for all fields in **`/etc/passwd`** when run without options. The **`useradd`** command does not set any valid password by default, and the user cannot log in until a password is set.
- **`useradd --help`** will display the basic options that can be used to override the defaults. In most cases, the same options can be used with the **`usermod`** command to modify an existing user.
- Some defaults, such as the range of valid UID numbers and default password aging rules, are read from the **`/etc/login.defs`** file. Values in this file are only used when creating new users. A change to this file will not have an effect on any existing users.

`usermod` modifies existing users

- **`usermod --help`** will display the basic options that can be used to modify an account. Some common options include:

USERMOD OPTIONS:	
`-c, --comment COMMENT`	Add a value, such as a full name, to the GECOS field.
`-g, --gid GROUP`	Specify the primary group for the user account.
`-G, --groups GROUPS`	Specify a list of supplementary groups for the user account.
`-a, --append`	Used with the **`-G`** option to append the user to the supplemental groups mentioned without removing the user from other groups.
`-d, --home HOME_DIR`	Specify a new home directory for the user account.
`-m, --move-home`	Move a user home directory to a new location. Must be used with the **`-d`** option.
`-s, --shell SHELL`	Specify a new login shell for the user account.
`-L, --lock`	Lock a user account.
`-U, --unlock`	Unlock a user account.

`userdel` deletes users

- **`userdel`** ***`username`*** removes the user from **`/etc/passwd`**, but leaves the home directory intact by default.
- **`userdel -r`** ***`username`*** removes the user and the user's home directory.

WARNING

When a user is removed with **`userdel`** without the **`-r`** option specified, the system will have files that are owned by an unassigned user ID number. This can also happen when files created by a deleted user exist outside that user's home directory. This situation can lead to information leakage and other security issues.

In Red Hat Enterprise Linux 7 the **`useradd`** command assigns new users the first free UID number available in the range starting from UID 1000 or above. (unless one is explicitly specified with the **`-u`** ***`UID`*** option). This is how information leakage can occur: If the first free UID number had been previously assigned to a user account which has since been removed from the system, the old user's UID number will get reassigned to the new user, giving the new user ownership of the old user's remaining files. The following scenario demonstrates this situation:

```
[root@serverX ~]# useradd prince[root@serverX ~]# ls -l /home
drwx------. 3 prince  prince     74 Feb  4 15:22 prince
[root@serverX ~]# userdel prince
[root@serverX ~]# ls -l /home
drwx------. 3    1000    1000    74 Feb  4 15:22 prince
[root@serverX ~]# useradd bob
[root@serverX ~]# ls -l /home
drwx------. 3 bob      bob        74 Feb  4 15:23 bob
drwx------. 3 bob      bob        74 Feb  4 15:22 prince
```

Notice that **`bob`** now owns all files that **`prince`** once owned. Depending on the situation, one solution to this problem is to remove all "unowned" files from the system when the user that created them is deleted. Another solution is to manually assign the "unowned" files to a different user. The root user can find "unowned" files and directories by running: **`find / -nouser -o -nogroup 2> /dev/null`**.

`passwd` sets passwords

- **`passwd`** ***`username`*** can be used to either set the user's initial password or change that user's password.
- The *root* user can set a password to any value. A message will be displayed if the password does not meet the minimum recommended criteria, but is followed by a prompt to retype the new password and all tokens are updated successfully.

```
[root@serverX ~]# passwd student
Changing password for user student.
New password: redhat123
BAD PASSWORD: The password fails the dictionary check - it is based on a
 dictionary word
Retype new password: redhat123
passwd: all authentication tokens updated successfully.
```

- A regular user must choose a password which is at least 8 characters in length and is not based on a dictionary word, the username, or the previous password.

UID ranges

Specific UID numbers and ranges of numbers are used for specific purposes by Red Hat Enterprise Linux.

- *UID 0* is always assigned to the superuser account, **root**.
- *UID 1-200* is a range of "system users" assigned statically to system processes by Red Hat.
- *UID 201-999* is a range of "system users" used by system processes that do not own files on the file system. They are typically assigned dynamically from the available pool when the software that needs them is installed. Programs run as these "unprivileged" system users in order to limit their access to just the resources they need to function.
- *UID 1000+* is the range available for assignment to regular users.

NOTE

Prior to Red Hat Enterprise Linux 7, the convention was that UID 1-499 was used for system users and UID 500+ for regular users. Default ranges used by **useradd** and **groupadd** can be changed in the **/etc/login.defs** file.

REFERENCES

useradd(8), **usermod**(8), **userdel**(8) man pages

GUIDED EXERCISE

CREATING USERS USING COMMAND-LINE TOOLS

In this lab, you will create a number of users on your serverX system, setting and recording an initial password for each user.

OUTCOMES

A system with additional user accounts.

Reset your serverX system.

1. Log into the GNOME desktop on serverX as **student** with a password of **student**.

2. Open a window with a Bash prompt.
   Select Applications → Utilities → Terminal.

3. Become the **root** user at the shell prompt.

```
[student@serverX ~]$
su -
Password:
redhat
```

4. Add the user *juliet*.

```
[root@serverX ~]#
useradd juliet
```

5. Confirm that *juliet* has been added by examining the **/etc/passwd** file.

```
[root@serverX ~]# tail -2 /etc/passwd
tcpdump:x:72:72::/:/sbin/nologin
juliet:x:1001:1001::/home/juliet:/bin/bash
```

6. Use the **passwd** command to initialize *juliet*'s password.

```
[root@serverX ~]# passwd juliet
Changing password for user juliet.
New password: juliet
```

```
BAD PASSWORD: The password is shorter than 8 characters
Retype new password: juliet
passwd: all authentication tokens updated successfully.
```

7. Continue adding the remaining users in the steps below and set initial passwords.

7.1. romeo

```
[root@serverX ~]# useradd romeo[root@serverX ~]# passwd romeo
Changing password for user romeo.
New password: romeo
BAD PASSWORD: The password is shorter than 8 characters
Retype new password: romeo
passwd: all authentication tokens updated successfully.
```

7.2. hamlet

```
[root@serverX ~]#
useradd hamlet
[root@serverX ~]#
passwd hamlet
```

7.3. reba

```
[root@serverX ~]#
useradd reba
[root@serverX ~]#
passwd reba
```

7.4. dolly

```
[root@serverX ~]#
useradd dolly
[root@serverX ~]#
passwd dolly
```

7.5. elvis

```
[root@serverX ~]#
useradd elvis
[root@serverX ~]#
passwd elvis
```

MANAGING LOCAL GROUP ACCOUNTS

OBJECTIVES

After completing this section, students should be able to create, modify, and delete locally defined group accounts.

MANAGING SUPPLEMENTARY GROUPS

A group must exist before a user can be added to that group. Several command-line tools are used to manage local group accounts.

`groupadd` creates groups

- **`groupadd`** ***`groupname`*** without options uses the next available GID from the range specified in the **`/etc/login.defs`** file.
- The **`-g`** ***`GID`*** option is used to specify a specific GID.

```
[student@serverX ~]$ sudo groupadd -g 5000 ateam
```

NOTE

Given the automatic creation of user private groups (GID 1000+), it is generally recommended to set aside a range of GID numbers to be used for supplementary groups. A higher range will avoid a collision with a system group (GID 0-999).

- The **`-r`** option will create a system group using a GID from the range of valid system GID numbers listed in the **`/etc/login.defs`** file.

```
[student@serverX ~]$ sudo groupadd -r appusers
```

`groupmod` modifies existing groups

- The **`groupmod`** command is used to change a group name to a GID mapping. The **`-n`** option is used to specify a new name.

```
[student@serverX ~]$ sudo groupmod -n javaapp appusers
```

- The **`-g`** option is used to specify a new GID.

```
[student@serverX ~]$ sudo groupmod -g 6000 ateam
```

`groupdel` deletes a group

- The **`groupdel`** command will remove a group.

```
[student@serverX ~]$ sudo groupdel javaapp
```

- A group may not be removed if it is the primary group of any existing user. As with **userdel**, check all file systems to ensure that no files remain owned by the group.

`usermod` alters group membership

- The membership of a group is controlled with user management. Change a user's primary group with **usermod -g *groupname***.

```
[student@serverX ~]$ sudo usermod -g student student
```

- Add a user to a supplementary group with **usermod -aG *groupname username***.

```
[student@serverX ~]$ sudo usermod -aG wheel elvis
```

IMPORTANT

The use of the **-a** option makes **usermod** function in "append" mode. Without it, the user would be removed from *all other* supplementary groups.

REFERENCES

group(5), **groupadd**(8), **groupdel**(8), and **usermod**(8) man pages

GUIDED EXERCISE

MANAGING GROUPS USING COMMAND-LINE TOOLS

In this lab, you will add users to newly created supplementary groups.

OUTCOMES

The **shakespeare** group consists of **juliet**, **romeo**, and **hamlet**. The **artists** group contains **reba**, **dolly**, and **elvis**.

Perform the following steps on serverX unless directed otherwise.

1. Become the **root** user at the shell prompt.

```
[student@serverX ~]$
su -
Password:
redhat
```

2. Create a supplementary group called **shakespeare** with a group ID of **30000**.

```
[root@serverX ~]#
groupadd -g 30000 shakespeare
```

3. Create a supplementary group called **artists**.

```
[root@serverX ~]#
groupadd artists
```

4. Confirm that *shakespeare* and *artists* have been added by examining the **/etc/group** file.

```
[root@serverX ~]# tail -5 /etc/group
reba:x:1004:
dolly:x:1005:
elvis:x:1006:
shakespeare:x:30000:
artists:x:30001:
```

5. Add the *juliet* user to the *shakespeare* group as a supplementary group.

```
[root@serverX ~]#
usermod -G shakespeare juliet
```

6. Confirm that *juliet* has been added using the **id** command.

```
[root@serverX ~]# id juliet
uid=1001(juliet) gid=1001(juliet) groups=1001(juliet),30000(shakespeare)
```

7. Continue adding the remaining users to groups as follows:

7.1. Add *romeo* and *hamlet* to the *shakespeare* group.

```
[root@serverX ~]#
usermod -G shakespeare romeo
[root@serverX ~]#
usermod -G shakespeare hamlet
```

7.2. Add *reba*, *dolly*, and *elvis* to the *artists* group.

```
[root@serverX ~]#
usermod -G artists reba
[root@serverX ~]#
usermod -G artists dolly
[root@serverX ~]#
usermod -G artists elvis
```

7.3. Verify the supplemental group memberships by examining the **/etc/group** file.

```
[root@serverX ~]# tail -5 /etc/group
reba:x:1004:
dolly:x:1005:
elvis:x:1006:
shakespeare:x:30000:juliet,romeo,hamlet
artists:x:30001:reba,dolly,elvis
```

MANAGING USER PASSWORDS

OBJECTIVES

After completing this section, students should be able to lock accounts manually or by setting a password-aging policy in the shadow password file.

SHADOW PASSWORDS AND PASSWORD POLICY

In the distant past, encrypted passwords were stored in the world-readable /etc/passwd file. This was thought to be reasonably secure until dictionary attacks on encrypted passwords became common. At that point, the encrypted passwords, or "password hashes," were moved to the more secure /etc/shadow file. This new file also allowed password aging and expiration features to be implemented.

There are three pieces of information stored in a modern password hash:

```
$1$gCjLa2/Z$6Pu0EK0AzfCjxjv2hoLOB/
```

1. **1**: The hashing algorithm. The number 1 indicates an MD5 hash. The number 6 appears when a SHA-512 hash is used.

2. **gCjLa2/Z**: The *salt* used to encrypt the hash. This is originally chosen at random. The salt and the unencrypted password are combined and encrypted to create the encrypted password hash. The use of a salt prevents two users with the same password from having identical entries in the **/etc/shadow** file.

3. **6Pu0EK0AzfCjxjv2hoLOB/**: The encrypted hash.

When a user tries to log in, the system looks up the entry for the user in **/etc/shadow**, combines the salt for the user with the unencrypted password that was typed in, and encrypts them using the hashing algorithm specified. If the result matches the encrypted hash, the user typed in the right password. If the result doesn't match the encrypted hash, the user typed in the wrong password and the login attempt fails. This method allows the system to determine if the user typed in the correct password without storing that password in a form usable for logging in.

NOTE

Red Hat Enterprise Linux 6 and 7 support two new strong password hashing algorithms, SHA-256 (algorithm **5**) and SHA-512 (algorithm **6**). Both the salt string and the encrypted hash are longer for these algorithms. The default algorithm used for password hashes can be changed by the **root** user by running the command **authconfig --passalgo** with one of the arguments **md5**, **sha256**, or **sha512**, as appropriate.

Red Hat Enterprise Linux 7 defaults to using SHA-512 encryption.

/etc/shadow format

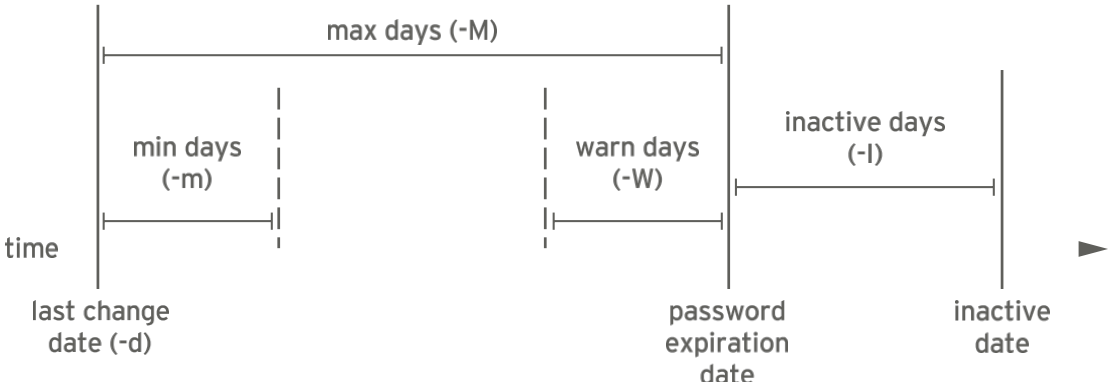
The format of **/etc/shadow** follows (nine colon-separated fields):

```
1name:2password:3lastchange:4minage:5maxage:6warning:7inactive:8expire:9blank
```

1. The login *name*. This must be a valid account name on the system.
2. The encrypted *password*. A password field which starts with a exclamation mark means that the password is locked.
3. The date of the last *password change*, represented as the number of days since 1970.01.01.
4. The *minimum* number of days before a password may be changed, where 0 means "no minimum age requirement."
5. The *maximum* number of days before a password must be changed.
6. The *warning* period that a password is about to expire. Represented in days, where 0 means "no warning given."
7. The number of days an account remains active after a password has expired. A user may still log into the system and change the password during this period. After the specified number of days, the account is locked, becoming *inactive*.
8. The account *expiration* date, represented as the number of days since 1970.01.01.
9. This *blank* field is reserved for future use.

PASSWORD AGING

The following diagram relates the relevant password-aging parameters, which can be adjusted using **chage** to implement a password-aging policy.




```
[root@serverX ~]#
chage -m 0 -M 90 -W 7 -I 14 username
```

chage -d 0 *username* will force a password update on next login.

chage -l *username* will list a username's current settings.

chage -E *YYYY-MM-DD username* will expire an account on a specific day.

NOTE

The **date** command can be used to calculate a date in the future.

```
[student@serverX ~]$ date -d "+45 days"
Sat Mar 22 11:47:06 EDT 2014
```

RESTRICTING ACCESS

With the **chage** command, an account expiration can be set. Once that date is reached, the user cannot log into the system interactively. The **usermod** command can "lock" an account with the **-L** option.

```
[student@serverX ~]$ sudo usermod -L elvis[student@serverX ~]$ su -
 elvisPassword: elvis
su: Authentication failure
```

When a user has left the company, the administrator may lock and expire an account with a single **usermod** command. The date must be given as the number of days since 1970.01.01.

```
[student@serverX ~]$
sudo usermod -L -e 1 elvis
```

Locking the account prevents the user from authenticating with a password to the system. It is the recommended method of preventing access to an account by an employee who has left the company. If the employee returns, the account can later be unlocked with **usermod -U *USERNAME***. If the account was also expired, be sure to also change the expiration date.

The nologin shell

Sometimes a user needs an account with a password to authenticate to a system, but does not need an interactive shell on the system. For example, a mail server may require an account to store mail and a password for the user to authenticate with a mail client used to retrieve mail. That user does not need to log directly into the system.

A common solution to this situation is to set the user's login shell to **/sbin/nologin**. If the user attempts to log into the system directly, the **nologin** "shell" will simply close the connection.

```
[root@serverX ~]# usermod -s /sbin/nologin student[root@serverX ~]# su - student
Last login: Tue Feb  4 18:40:30 EST 2014 on pts/0
This account is currently not available.
```

NOTE

Use of the **nologin** shell prevents interactive use of the system, but does not prevent all access. A user may still be able to authenticate and upload or retrieve files through applications such as web applications, file transfer programs, or mail readers.

REFERENCES

chage(1), **usermod**(8), **shadow**(5), **crypt**(3) man pages

GUIDED EXERCISE

MANAGING USER PASSWORD AGING

In this lab, you will set unique password policies for users.

OUTCOMES

The password for *romeo* must be changed when the user first logs into the system, every 90 days thereafter, and the account expires in 180 days.

Perform the following steps on serverX unless directed otherwise.

1. Explore locking and unlocking accounts.

   1.1. Lock the **romeo** account.

```
[student@serverX ~]$
sudo usermod -L romeo
```

   1.2. Attempt to log in as **romeo**.

```
[student@serverX ~]$ su - romeoPassword: romeo
su: Authentication failure
```

   1.3. Unlock the **romeo** account.

```
[student@serverX ~]$
sudo usermod -U romeo
```

2. Change the password policy for **romeo** to require a new password every 90 days.

```
[student@serverX ~]$ sudo chage -M 90 romeo[student@serverX ~]$ sudo chage -l
 romeo
Last password change                                    : Feb 03, 2014
Password expires                                        : May 04, 2014
Password inactive                                       : never
Account expires                                         : never
Minimum number of days between password change          : 0
Maximum number of days between password change          : 90
Number of days of warning before password expires       : 7
```

3. Additionally, force a password change on the first login for the **romeo** account.

```
[student@serverX ~]$
```

```
sudo chage -d 0 romeo
```

4. Log in as **romeo** and change the password to **forsooth123**.

```
[student@serverX ~]$ su - romeoPassword: romeo
You are required to change your password immediately (root enforced)
Changing password for romeo.
(current) UNIX password: romeo
New password: forsooth123
Retype new password: forsooth123
[romeo@serverX ~]$ exit
```

5. Expire accounts in the future.

5.1. Determine a date 180 days in the future.

```
[student@serverX ~]$
date -d "+180 days"
Sat Aug  2 17:05:20 EDT 2014
```

5.2. Set accounts to expire on that date.

```
[student@serverX ~]$ sudo chage -E 2014-08-02 romeo[student@serverX ~]$ sudo chage
 -l romeo
Last password change                                    : Feb 03, 2014
Password expires                                        : May 04, 2014
Password inactive                                       : never
Account expires                                         : Aug 02, 2014
Minimum number of days between password change          : 0
Maximum number of days between password change          : 90
Number of days of warning before password expires       : 7
```

LAB-5 CONTROLLING ACCESS TO FILES WITH LINUX FILE SYSTEM PERMISSIONS

Goal- To set Linux file system permissions on files and interpret the security effects of different permission settings

Sections

- Linux File System Permissions (and Practice)
- Managing File System Permissions from the Command Line (and Practice)
- Managing Default Permissions and File Access (and Practice)

LAB

CONTROLLING ACCESS TO FILES WITH LINUX FILE SYSTEM PERMISSIONS

PERFORMANCE CHECKLIST

In this lab, you will configure a system with directories for user collaboration.

OUTCOMES

- A directory on serverX called **/home/stooges** where these three users can work collaboratively on files.
- Only the user and group access, create, and delete files in **/home/stooges**. Files created in this directory should automatically be assigned a group ownership of **stooges**.
- New files created by users will not be accessible outside of the group.

Reset your serverX system. Log into and set up your server system.

```
[student@serverX ~]$
lab permissions setup
```

Your serverX machine has three accounts, **curly**, **larry**, and **moe**, who are members of a group called **stooges**. The password for each account is **password**.

1. Open a terminal window and become root on serverX.
2. Create the **/home/stooges** directory.
3. Change group permissions on the /home/stooges directory so it belongs to the stooges group.
4. Set permissions on the **/home/stooges** directory so it is a set GID bit directory (2), the owner (7) and group (7) have full read/write/execute permissions, and other users have no permission (0) to the directory.
5. Check that the permissions were set properly.
6. Modify the global login scripts so that normal users have a umask setting which prevents others from viewing or modifying new files and directories.
7. When you finish, open a terminal window on serverX and run **lab permissions grade** to confirm you have done everything correctly.

LINUX FILE SYSTEM PERMISSIONS

OBJECTIVES

After completing this section, students should be able to explain how the Linux file permissions model works.

LINUX FILE SYSTEM PERMISSIONS

Access to files by users are controlled by *file permissions*. The Linux file permissions system is simple but flexible, which makes it easy to understand and apply, yet able to handle most normal permission cases easily.

Files have just three categories of user to which permissions apply. The file is owned by a *user*, normally the one who created the file. The file is also owned by a single *group*, usually the primary group of the user who created the file, but this can be changed. Different permissions can be set for the owning user, the owning group, and for all *other* users on the system that are not the user or a member of the owning group.

The most specific permissions apply. So, *user* permissions override *group* permissions, which override *other* permissions.

In the graphic that follows, joshua is a member of the groups joshua and web, while allison is a member of allison, wheel, and web. When joshua and allison have the need to collaborate, the files should be associated with the group web and the group permissions should allow the desired access.

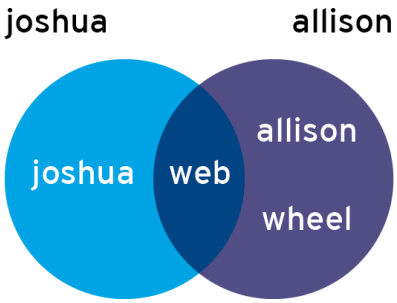


Figure 6.1: Group membership illustration

There are also just three categories of permissions which apply: *read*, *write*, and *execute*. These permissions affect access to files and directories as follows:

Effects of permissions on files and directories

PERMISSIO	EFFECT ON FILES	EFFECT ON DIRECTORIES
`r` (read)	Contents of the file can be read.	Contents of the directory (file names) can be listed.
`w` (write)	Contents of the file can be changed.	Any file in the directory may be created or deleted.

PERMISSIO	EFFECT ON FILES	EFFECT ON DIRECTORIES
x (exec)	Files can be executed as commands.	Contents of the directory can be accessed (dependent on the permissions of the files in the directory).

Note that users normally have both **read** and **exec** on read-only directories, so that they can list the directory and access its contents. If a user only has **read** access on a directory, the names of the files in it can be listed, but no other information, including permissions or time stamps, are available, nor can they be accessed. If a user only has **exec** access on a directory, they cannot list the names of the files in the directory, but if they already know the name of a file which they have permission to read, then they can access the contents of that file by explicitly specifying the file name.

A file may be removed by anyone who has write permission to the directory in which the file resides, regardless of the ownership or permissions on the file itself. (This can be overridden with a special permission, the *sticky bit*, which will be discussed at the end of the unit.)

VIEWING FILE/DIRECTORY PERMISSIONS AND OWNERSHIP

The **-l** option of the **ls** command will expand the file listing to include both the permissions of a file and the ownership:

```
[student@desktopX ~]$ ls -l test
-rw-rw-r--. 1 student student 0 Feb  8 17:36 test
```

The command **ls -l directoryname** will show the expanded listing of all of the files that reside inside the directory. To prevent the descent into the directory and see the expanded listing of the directory itself, add the **-d** option to ls:

```
[student@desktopX ~]$ ls -ld /home
drwxr-xr-x. 5 root root 4096 Jan 31 22:00 /home
```

NOTE

Unlike NTFS permissions, Linux permissions only apply to the directory or file that they are set on. Permissions on a directory are not inherited automatically by the subdirectories and files within it. (The permissions on a directory *may* effectively block access to its contents, however.) All permissions in Linux are set directly on each file or directory.

The read permission on a directory in Linux is roughly equivalent to List folder contents in Windows.

The write permission on a directory in Linux is equivalent to Modify in Windows; it implies the ability to delete files and subdirectories. In Linux, if write and the sticky bit are both set on a directory, then only the user that owns a file or subdirectory in the directory may delete it, which is close to the behavior of the Windows Write permission.

Root has the equivalent of the Windows Full Control permission on all files in Linux. However, root may still have access restricted by the system's SELinux policy and the security context of the process and files in question. SELinux will be discussed in a later course.

EXAMPLES: LINUX USER, GROUP, OTHER CONCEPTS

```
Users and their groups:

    lucy      lucy,ricardo
    ricky     ricky,ricardo
    ethel     ethel,mertz
    fred      fred,mertz

File attributes (permissions, user & group ownership, name):

    drwxrwxr-x    ricky    ricardo    dir (which contains the following files)
      -rw-rw-r--    lucy     lucy       lfile1
      -rw-r--rw-    lucy     ricardo    lfile2
      -rw-rw-r--    ricky    ricardo    rfile1
      -rw-r-----    ricky    ricardo    rfile2
```

ALLOWED/DENIED BEHAVIOR	CONTROLLING PERMISSIONS
`lucy` is the only person who can change the contents of `lfile1`.	**`lucy`** has write permissions on the file **`lfile1`** as the owner. No one is listed as a member of the group **`lucy`**. The permissions for *other* do not include write permissions.
`ricky` can view the contents of `lfile2`, but cannot modify the contents of `lfile2`.	**`ricky`** is a member of the group **`ricardo`**, and that group has read-only permissions on **`lfile2`**. Even though *other* has write permissions, group permissions take precedence.
`ricky` can delete `lfile1` and `lfile2`.	**`ricky`** has write permissions on the directory containing both files, and as such, he can delete any file in that directory.
`ethel` can change the contents of `lfile2`.	Since **`ethel`** is not **`lucy`**, and is not a member of the **`ricardo`** group, *other* permissions apply to her, and those include write permission.
`lucy` can change the contents of `rfile1`.	**`lucy`** is a member of the **`ricardo`** group, and that group has both read and write permissions on **`rfile1`**.
`ricky` can view and modify the contents of `rfile2`. **`lucy` can view but not modify the contents of `rfile2`.** **`ethel` and `fred` do not have any access to the contents of `rfile2`.**	**`ricky`** owns the file and has both read and write access to **`rfile2`**. **`lucy`** is a member of the **`ricardo`** group, and that group has read-only access to **`rfile2`**. *other* permissions apply to **`ethel`** and **`fred`**, and those permissions do not include read or write permission.

SOLUTION

INTERPRETING FILE AND DIRECTORY PERMISSIONS

Using the directory listing presented below, match the items that follow to their counterparts in the table.

```
Users and their groups:

    wilma     wilma,flintstone
    fred      fred,flintstone
    betty     betty,rubble
    barney    barney,rubble

File attributes (permissions, user & group ownership, name):

    drwxrwxr-x    fred     flintstone  dir (which contains the following files)
      -rw-rw-r--    wilma    wilma       lfile1
      -rw-r--rw-    wilma    flintstone  lfile2
      -rw-rw-r--    fred     flintstone  rfile1
      -rw-r-----    fred     flintstone  rfile2
```

DESCRIPTION	FILE NAME
Is owned by **fred** and readable by all users	rfile1
Contents may be modified by the user **betty**	lfile2
Can be deleted by the user **fred**	all
Cannot be read by the user **barney**	rfile2
Has a group ownership of **wilma**	lfile1
Can be deleted by the user **barney**	none

MANAGING FILE SYSTEM PERMISSIONS FROM THE COMMAND LINE

OBJECTIVES

After completing this section, students should be able to change the permissions and ownership of files using command-line tools.

CHANGING FILE/DIRECTORY PERMISSIONS

The command used to change permissions from the command line is **chmod**, short for "change mode" (permissions are also called the *mode* of a file). The **chmod** command takes a permission instruction followed by a list of files or directories to change. The permission instruction can be issued either symbolically (the symbolic method) or numerically (the numeric method).

Symbolic method keywords:

```
chmod WhoWhatWhich file|directory
```

- *Who* is u, g, o, a *(for user, group, other, all)*
- *What* is +, -, = *(for add, remove, set exactly)*
- *Which* is r, w, x *(for read, write, execute)*

The *symbolic* method of changing file permissions uses letters to represent the different groups of permissions: **u** for user, **g** for group, **o** for other, and **a** for all.

With the symbolic method, it is not necessary to set a complete new group of permissions. Instead, it is possible to change one or more of the existing permissions. In order to accomplish this, use three symbols: **+** to add permissions to a set, **–** to remove permissions from a set, and **=** to replace the entire set for a group of permissions.

The permissions themselves are represented by a single letter: **r** for read, **w** for write, and **x** for execute. When using **chmod** to change permissions with the symbolic method, using a capital **X** as the permission flag will add execute permission only if the file is a directory or already has execute set for user, group, or other.

Numeric method:

```
chmod ### file|directory
```

- Each digit represents an access level: user, group, other.
- # is sum of r=4, w=2, and x=1.

Using the *numeric* method, permissions are represented by a three-digit (or four, when setting advanced permissions) *octal* number. A single octal digit can represent the numbers 0–7, exactly the number of possibilities for a three-bit number.

To convert between symbolic and numeric representation of permissions, we need to know how the mapping is done. In the three-digit octal (numeric) representation, each digit stands for one group of permissions, from left to right: user, group, and other. In each of these groups, start with **0**. If the read permission is present, add **4**. Add **2** if write is present, and **1** for execute.

Numeric permissions are often used by advanced administrators since they are shorter to type and pronounce, while still giving full control over all permissions.

Examine the permissions **-rwxr-x---**. For the user, **rwx** is calculated as **4+2+1=7**. For the group, **r-x** is calculated as **4+0+1=5**, and for other users, **---** is represented with **0**. Putting these three together, the numeric representation of those permissions is **750**.

This calculation can also be performed in the opposite direction. Look at the permissions **640**. For the user permissions, **6** represents read (4) and write (2), which displays as **rw-**. For the group part, **4** only includes read (4) and displays as **r--**. The **0** for other provides no permissions (**---**) and the final set of symbolic permissions for this file is **-rw-r-----**.

Examples

- Remove read and write permission for group and other on **file1**:

```
[student@desktopX ~]$
chmod go-rw file1
```

- Add execute permission for everyone on **file2**:

```
[student@desktopX ~]$
chmod a+x file2
```

- Set read, write, and execute permission for user, read, and execute for group, and no permission for other on **sampledir**:

```
[student@desktopX ~]$
chmod 750 sampledir
```

NOTE

The **chmod** command supports the **-R** option to recursively set permissions on the files in an entire directory tree. When using the **-R** option, it can be useful to set permissions symbolically using the **X** flag. This will allow the execute (search) permission to be set on directories so that their contents can be accessed, without changing permissions on most files. But be cautious. If a file has any execute permission set, **X** will set the specified execute permission on that file as well. For example, the following command will recursively set read and write access on **demodir** and all its children for their group owner, but will only apply group execute permissions to directories and files which already have execute set for user, group, and/or other.

```
[student@desktopX ~]#
chmod -R g+rwX demodir
```

CHANGING FILE/DIRECTORY USER OR GROUP OWNERSHIP

A newly created file is owned by the user who creates the file. By default, the new file has a group ownership which is the primary group of the user creating the file. Since Red Hat Enterprise Linux uses user private groups, this group is often a group with only that user as a member. To grant access based on group membership, the owner or the group of a file may need to be changed.

File ownership can be changed with the **chown** command (change owner). For example, to grant ownership of the file **foofile** to **student**, the following command could be used:

```
[root@desktopX ~]#
chown student foofile
```

chown can be used with the **-R** option to recursively change the ownership of an entire directory tree. The following command would grant ownership of **foodir** and all files and subdirectories within it to **student**:

```
[root@desktopX ~]#
chown -R student foodir
```

The **chown** command can also be used to change group ownership of a file by preceding the group name with a colon (**:**). For example, the following command will change the group **foodir** to **admins**:

```
[root@desktopX ~]#
chown :admins foodir
```

The **chown** command can also be used to change both owner and group at the same time by using the syntax ***owner*:*group***. For example, to change the ownership of **foodir** to **visitor** and the group to **guests**, use:

```
[root@desktopX ~]#
chown visitor:guests foodir
```

Only **root** can change the ownership of a file. Group ownership, however, can be set by **root** or the file's owner. **root** can grant ownership to any group, while non-**root** users can grant ownership only to groups they belong to.

NOTE

Instead of using **chown**, some users change the group ownership by using the **chgrp** command; this command works exactly the same as changing ownership with **chown**, including the use of **-R** to affect entire directory trees.

REFERENCES

ls(1), **chmod**(1), **chown**(1), and **chgrp**(1) man pages

MANAGING DEFAULT PERMISSIONS AND FILE ACCESS

OBJECTIVES

After completing this section, students should be able to configure a directory in which newly created files are automatically writable by members of the group which owns the directory, using special permissions and default umask settings.

SPECIAL PERMISSIONS

The setuid (or setgid) permission on an executable file means that the command will run as the user (or group) of the file, not as the user that ran the command. One example is the **passwd** command:

```
[student@desktopX ~]$ ls -l /usr/bin/passwd
-rwsr-xr-x. 1 root root 35504 Jul 16  2010 /usr/bin/passwd
```

In a long listing, you can spot the **setuid** permissions by a lowercase **s** where you would normally expect the **x** (owner execute permissions) to be. If the owner does not have execute permissions, this will be replaced by an uppercase **S**.

The sticky bit for a directory sets a special restriction on deletion of files: Only the owner of the file (and **root**) can delete files within the directory. An example is **/tmp**:

```
[student@desktopX ~]$ ls -ld /tmp
drwxrwxrwt. 39 root root 4096 Feb  8 20:52 /tmp
```

In a long listing, you can spot the **sticky** permissions by a lowercase **t** where you would normally expect the **x** (other execute permissions) to be. If the other does not have execute permissions, this will be replaced by an uppercase **T**.

Lastly, setgid on a directory means that files created in the directory will inherit the group affiliation from the directory, rather than inheriting it from the creating user. This is commonly used on group collaborative directories to automatically change a file from the default private group to the shared group.

In a long listing, you can spot the **setgid** permissions by a lowercase **s** where you would normally expect the **x** (group execute permissions) to be. If the group does not have execute permissions, this will be replaced by an uppercase **S**.

Effects of special permissions on files and directories

SPECIAL PERMISSION	EFFECT ON FILES	EFFECT ON DIRECTORIES
u+s (suid)	File executes as the user that owns the file, not the user that ran the file.	No effect.

SPECIAL PERMISSION	EFFECT ON FILES	EFFECT ON DIRECTORIES
g+s (sgid)	File executes as the group that owns the file.	Files newly created in the directory have their group owner set to match the group owner of the directory.
o+t (sticky)	No effect.	Users with write on the directory can only remove files that they own; they cannot remove or force saves to files owned by other users.

Setting special permissions

- Symbolically: setuid = **u+s**; setgid = **g+s**; sticky = **o+t**
- Numerically (fourth preceding digit): setuid = 4; setgid = 2; sticky = 1

Examples

- Add the setgid bit on **directory**:

```
[root@desktopX ~]#
chmod g+s directory
```

- Set the setgid bit, and read/write/execute for user and group on **directory**:

```
[root@desktopX ~]#
chmod 2770 directory
```

DEFAULT FILE PERMISSIONS

The default permissions for files are set by the processes that create them. For example, text editors create files so they are readable and writeable, but not executable, by everyone. The same goes for shell redirection. Additionally, binary executables are created executable by the compilers that create them. The **mkdir** command creates new directories with all permissions set—read, write, and execute.

Experience shows that these permissions are not typically set when new files and directories are created. This is because some of the permissions are cleared by the umask of the shell process. The **umask** command without arguments will display the current value of the shell's umask:

```
[student@desktopX ~]$ umask
0002
```

Every process on the system has a umask, which is an octal bitmask that is used to clear the permissions of new files and directories that are created by the process. If a bit is set in the umask, then the corresponding permission is cleared in new files. For example, the previous umask,

0002, clears the write bit for other users. The leading zeros indicate the special, user, and group permissions are not cleared. A umask of 077 clears all the group and other permissions of newly created files.

Use the **umask** command with a single numeric argument to change the umask of the current shell. The numeric argument should be an octal value corresponding to the new umask value. If it is less than 3 digits, leading zeros are assumed.

The system default umask values for Bash shell users are defined in the **/etc/profile** and **/etc/bashrc** files. Users can override the system defaults in their **.bash_profile** and **.bashrc** files.

In this example, please follow along with the next steps while your instructor demonstrates the effects of **umask** on new files and directories.

1. Create a new file and directory to see how the default umask affects permissions.

```
[student@desktopX ~]$ touch newfile1[student@desktopX ~]$ ls -l newfile1
-rw-rw-r--. 1 student student 0 May  9 01:54 newfile1
[student@desktopX ~]$ mkdir newdir1
[student@desktopX ~]$ ls -ld newdir1
drwxrwxr-x. 2 student student 0 May  9 01:54 newdir1
```

2. Set the umask value to 0. This setting will not mask any of the permissions of new files. Create a new file and directory to see how this new umask affects permissions.

```
[student@desktopX ~]$ umask 0[student@desktopX ~]$ touch newfile2[student@desktopX
 ~]$ ls -l newfile2
-rw-rw-rw-. 1 student student 0 May  9 01:54 newfile2
[student@desktopX ~]$ mkdir newdir2
[student@desktopX ~]$ ls -ld newdir2
drwxrwxrwx. 2 student student 0 May  9 01:54 newdir2
```

3. Set the umask value to 007. This setting will mask all of the “other” permissions of new files.

```
[student@desktopX ~]$ umask 007[student@desktopX ~]$ touch
 newfile3[student@desktopX ~]$ ls -l newfile3
-rw-rw----. 1 student student 0 May  9 01:55 newfile3
[student@desktopX ~]$ mkdir newdir3
[student@desktopX ~]$ ls -ld newdir3
drwxrwx---. 2 student student 0 May  9 01:54 newdir3
```

4. Set the umask value to 027. This setting will mask write access for group members and all of the “other” permissions of new files.

```
[student@desktopX ~]$ umask 027[student@desktopX ~]$ touch
 newfile4[student@desktopX ~]$ ls -l newfile4
-rw-r-----. 1 student student 0 May  9 01:55 newfile4
[student@desktopX ~]$ mkdir newdir4
[student@desktopX ~]$ ls -ld newdir4
drwxr-x---. 2 student student 0 May  9 01:54 newdir4
```

5. Log in as **root** to change the default umask for unprivileged users to prohibit all access for users not in their group.

Modify **/etc/bashrc** and **/etc/profile** to change the default umask for Bash shell users. Since the default umask for unprivileged users is 0002, look for the **umask** command in these files that sets the umask to that value. Change them to set the umask to 007.

```
[root@desktopX ~]# less /etc/bashrc
    # You could check uidgid reservation validity in
    # /usr/share/doc/setup-*/uidgid file
    if [ $UID -gt 199 ] && [ "`id -gn`" = "`id -un`" ]; then
       umask 002
    else
       umask 022
    fi

    # Only display echos from profile.d scripts if we are no login shell
[root@desktopX ~]# vim /etc/bashrc
[root@desktopX ~]# less /etc/bashrc
    # You could check uidgid reservation validity in
    # /usr/share/doc/setup-*/uidgid file
    if [ $UID -gt 199 ] && [ "`id -gn`" = "`id -un`" ]; then
       umask 007
    else
       umask 022
    fi

    # Only display echos from profile.d scripts if we are no login shell
[root@desktopX ~]# less /etc/profile
    # You could check uidgid reservation validity in
    # /usr/share/doc/setup-*/uidgid file
    if [ $UID -gt 199 ] && [ "`id -gn`" = "`id -un`" ]; then
        umask 002
    else
        umask 022
    fi

    for i in /etc/profile.d/*.sh ; do
[root@desktopX ~]# vim /etc/profile
[root@desktopX ~]# less /etc/profile
    # You could check uidgid reservation validity in
    # /usr/share/doc/setup-*/uidgid file
    if [ $UID -gt 199 ] && [ "`id -gn`" = "`id -un`" ]; then
        umask 007
    else
        umask 022
    fi

    for i in /etc/profile.d/*.sh ; do
```

6. Log back in as **student** and confirm that the umask changes you made are persistent.

```
[student@desktopX ~]$ umask
0007
```

LAB-6 MONITORING AND MANAGING LINUX PROCESSES

Goal- To evaluate and control processes running on a Red Hat Enterprise Linux system

Sections

- Processes (and Practice)
- Controlling Jobs (and Practice)
- Killing Processes (and Practice)
- Monitoring Process Activity (and Practice)

LAB

MONITORING AND MANAGING LINUX PROCESSES

PERFORMANCE CHECKLIST

In this lab, students will locate and manage processes that are using the most resources on a system.

OUTCOMES

Experience using **top** as a process management tool.

Run **lab processes setup** as **root** on serverX to prepare for this exercise.

```
[root@serverX ~]#
lab processes setup
```

Perform the following tasks as **student** on the serverX machine.

1. In a terminal window, run the **top** utility. Size the window to be as tall as possible.
2. Observe the **top** display. The default display sorts by CPU utilization, highest first. What are the processes using the most CPU time?
3. Change the display to sort by the amount of memory in use by each process.
4. What are the processes with the largest memory allocations?
5. Turn off the use of bold in the display. Save this configuration for reuse when **top** is restarted.
6. Exit **top**, then restart it again. Confirm that the new display uses the saved configuration; i.e., the display starts sorted by memory utilization and bold is turned off.
7. Modify the display to again sort by CPU utilization. Turn on the use of bold. Observe that only *Running* or *Runnable* (state **R**) process entries are bold. Save this configuration.
8. Open another terminal window if necessary. As **root**, suspend the **hippo** process. In **top**, observe that the process state is now **T**.
9. The **hippo** process quickly disappears from the display, since it is no longer actively using CPU resources. List the process information from the command line to confirm the process state.
10. Resume execution of the **hippo** processes.
11. When finished observing the display, terminate the extra processes elephant and hippo using the command line. Confirm that the processes no longer display in **top**.
12. Check that the cleanup is successful by running the grading script. If necessary, find and terminate processes listed by the grading script, and repeat grading.
13. Exit the **top** display. Close extra terminal windows.

PROCESS STATES

In a multitasking operating system, each CPU (or CPU core) can be working on one process at a single point in time. As a process runs, its immediate requirements for CPU time and resource allocation change. Processes are assigned a *state*, which changes as circumstances require.

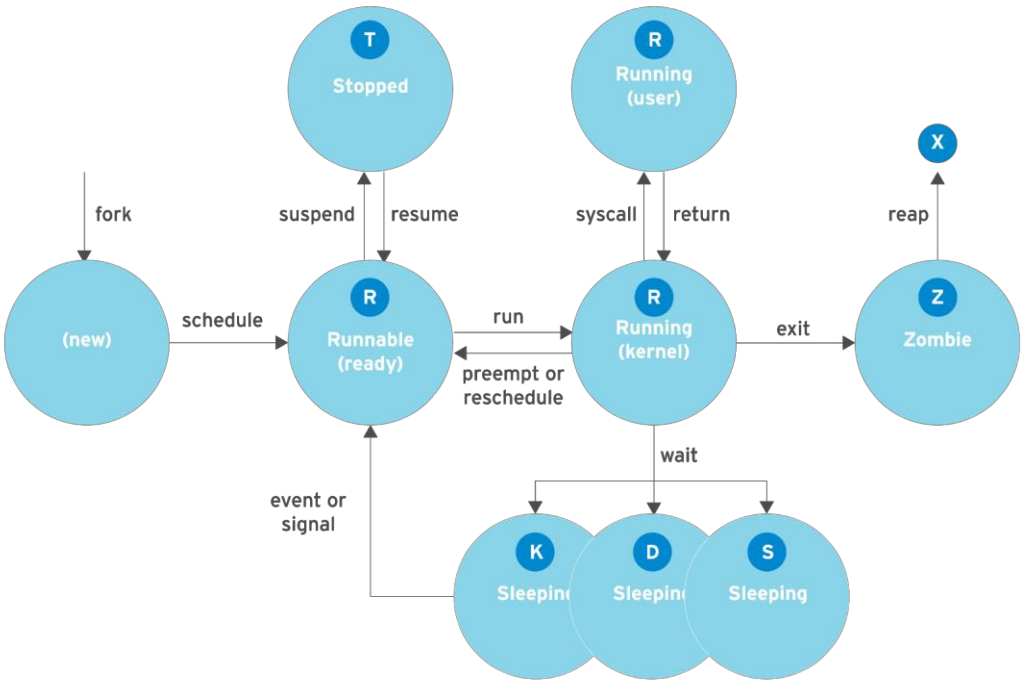


Figure 7.2: Linux process states

The Linux process states are illustrated in the previous diagram and described in the following table.

Linux process states

NAME	FLAG	KERNEL-DEFINED STATE NAME AND DESCRIPTION
Running	`R`	TASK_RUNNING: The process is either executing on a CPU or waiting to run. Process can be executing user routines or kernel routines (system calls), or be queued and ready when in the *Running* (or *Runnable*) state.
Sleeping	`S`	TASK_INTERRUPTIBLE: The process is waiting for some condition: a hardware request, system resource access, or signal. When an event or signal satisfies the condition, the process returns to *Running*.
	`D`	TASK_UNINTERRUPTIBLE: This process is also *Sleeping*, but unlike `S` state, will not respond to delivered signals. Used only under specific conditions in which process interruption may cause an unpredictable device state.

NAME	FLAG	KERNEL-DEFINED STATE NAME AND DESCRIPTION
	K	TASK_KILLABLE: Identical to the uninterruptible D state, but modified to allow the waiting task to respond to a signal to be killed (exited completely). Utilities frequently display *Killable* processes as D state.
Stopped	T	TASK_STOPPED: The process has been *Stopped* (suspended), usually by being signaled by a user or another process. The process can be continued (resumed) by another signal to return to *Running*.
	T	TASK_TRACED: A process that is being debugged is also temporarily *Stopped* and shares the same T state flag.
Zombie	Z	EXIT_ZOMBIE: A child process signals its parent as it exits. All resources except for the process identity (PID) are released.
	X	EXIT_DEAD: When the parent cleans up (*reaps*) the remaining child process structure, the process is now released completely. This state will never be observed in process-listing utilities.

LISTING PROCESSES

The **ps** command is used for listing current processes. The command can provide detailed process information, including:

- the user identification (UID) which determines process privileges,
- the unique process identification (PID),
- the CPU and real time already expended,
- how much memory the process has allocated in various locations,
- the location of process **STDOUT**, known as the *controlling terminal*, and
- the current process state.

A common display listing (options **aux**) displays all processes, with columns in which users will be interested, and includes processes without a controlling terminal. A long listing (options **lax**) provides more technical detail, but may display faster by avoiding the username lookup. The similar UNIX syntax uses the options **-ef** to display all processes.

```
[student@serverX ~]$ ps aux
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root          1  0.1  0.1  51648  7504 ?        Ss   17:45   0:03 /usr/lib/systemd/
syst
```

By default, **ps** with no options selects all processes with the same *effective user ID* (EUID) as the current user and associated with the same terminal where **ps** was invoked.

- Processes in brackets (usually at the top) are scheduled kernel threads.
- Zombies show up in a ps listing as *exiting* or *defunct*.
- **ps** displays once. Use **top**(1) for a repetitive update process display.
- **ps** can display in tree format to view parent/child relationships.
- The default output is unsorted. Display order matches that of the system process table, which reuses table rows as processes die and new ones are created. Output may appear chronological, but is not guaranteed unless explicit **-O** or **--sort** options are used.

REFERENCES

info libc signal (*GNU C Library Reference Manual*)

- Section 24: Signal Handling

info libc processes (*GNU C Library Reference Manual*)

- Section 26: Processes

ps(1) and **signal**(7) man pages

SOLUTION

PROCESSES

Match the following items to their counterparts in the table.

DESCRIPTION	STATE
Process has been stopped (suspended).	T
Process has released all its resources except its PID.	Z
Process is running or waiting to run on a CPU.	R
Process is sleeping until some condition is met.	S
Process is waiting for I/O or some condition to be met and must not respond to signals.	D

CONTROLLING JOBS

OBJECTIVES

After completing this section, students should be able to:

- Explain the terms foreground, background, and controlling terminal.
- Use job control to manage multiple command-line tasks.

JOBS AND SESSIONS

Job control is a feature of the shell which allows a single shell instance to run and manage multiple commands.

A *job* is associated with each pipeline entered at a shell prompt. All processes in that pipeline are part of the job and are members of the same *process group*. (If only one command is entered at a shell prompt, that can be considered to be a minimal "pipeline" of one command. That command would be the only member of that job.)

Only one job can read input and keyboard-generated signals from a particular terminal window at a time. Processes that are part of that job are *foreground* processes of that *controlling terminal*.

A *background* process of that controlling terminal is a member of any other job associated with that terminal. Background processes of a terminal can not read input or receive keyboard-generated interrupts from the terminal, but may be able to write to the terminal. A job in the background may be stopped (suspended) or it may be running. If a running background job tries to read from the terminal, it will be automatically suspended.

Each terminal is its own *session*, and can have a foreground process and independent background processes. A job is part of exactly one session, the one belonging to its controlling terminal.

The **ps** command will show the device name of the controlling terminal of a process in the **TTY** column. Some processes, such as *system daemons*, are started by the system and not from a shell prompt. These processes do not have a controlling terminal, are not members of a job, and can not be brought to the foreground. The **ps** command will display a question mark (**?**) in the **TTY** column for these processes.

RUNNING JOBS IN THE BACKGROUND

Any command or pipeline can be started in the background by appending an ampersand (**&**) to the end of the command line. The **bash** shell displays a *job number* (unique to the session) and the PID of the new child process. The shell does not wait for the child process and redisplays the shell prompt.

```
[student@serverX ~]$ sleep 10000 &
[1] 5947
[student@serverX ~]$
```

NOTE

When backgrounding a pipeline with an ampersand, the PID of the last command in the pipeline will be the one that is output. All processes in the pipeline are still members of that job.

```
[student@serverX ~]$  example_command | sort | mail -s "Sort output" &
[1] 5998
```

The **bash** shell tracks jobs, per session, in a table displayed with the **jobs** command.

```
[student@serverX ~]$ jobs
[1]+  Running                 sleep 10000 &
[student@serverX ~]$
```

A background job can be brought to the foreground by using the **fg** command with its job ID (**%***job number*).

```
[student@serverX ~]$ fg %1
sleep 10000
_
```

In the preceding example, the **sleep** command is now running in the foreground on the controlling terminal. The shell itself is again asleep, waiting for this child process to exit.

To send a foreground process to the background, first press the keyboard-generated *suspend* request (**Ctrl+z**) on the terminal.

```
sleep 10000
^Z
[1]+  Stopped                 sleep 10000
[student@serverX ~]$
```

The job is immediately placed in the background and is suspended.

The **ps j** command will display information relating to jobs. The PGID is the PID of the *process group leader*, normally the first process in the job's pipeline. The SID is the PID of the *session leader*, which for a job is normally the interactive shell that is running on its controlling terminal. Since the example **sleep** command is currently suspended, its process state is **T**.

```
[student@serverX ~]$ ps j
 PPID   PID  PGID   SID TTY      TPGID STAT   UID   TIME COMMAND
 2764  2768  2768  2768 pts/0     6377 Ss    1000   0:00 /bin/bash
 2768  5947  5947  2768 pts/0     6377 T     1000   0:00 sleep 10000
 2768  6377  6377  2768 pts/0     6377 R+    1000   0:00 ps j
[student@serverX ~]$
```

To start the suspended process running in the background, use the **bg** command with the same job ID.

```
[student@serverX ~]$ bg %1
[1]+ sleep 10000 &
```

```
[student@serverX ~]$
```

The shell will warn a user who attempts to exit a terminal window (session) with suspended jobs. If the user tries exiting again immediately, the suspended jobs are killed.

REFERENCES

Additional information may be available in the chapter on viewing system processes in the *Red Hat Enterprise Linux System Administrator's Guide* for Red Hat Enterprise Linux 7, which can be found at
https://access.redhat.com/documentation/

bash info page (*The GNU Bash Reference Manual*)

- Section 7: Job Control

libc info page (*GNU C Library Reference Manual*)

- Section 24: Signal Handling
- Section 26: Processes

bash(1), **builtins**(1), **ps**(1), **sleep**(1) man pages

KILLING PROCESSES

OBJECTIVES

After completing this section, students should be able to:

- Use commands to kill and communicate with processes.
- Define the characteristics of a daemon process.
- End user sessions and processes.

PROCESS CONTROL USING SIGNALS

A signal is a software interrupt delivered to a process. Signals report events to an executing program. Events that generate a signal can be an *error*, *external event* (e.g., I/O request or expired timer), or by *explicit request* (e.g., use of a signal-sending command or by keyboard sequence).

The following table lists the fundamental signals used by system administrators for routine process management. Refer to signals by either their short (**`HUP`**) or proper (**`SIGHUP`**) name.

Fundamental process management signals

SIGNAL NUMBER	SHORT NAME	DEFINITION	PURPOSE
1	**`HUP`**	Hangup	Used to report termination of the controlling process of a terminal. Also used to request process reinitialization (configuration reload) without termination.
2	**`INT`**	Keyboard interrupt	Causes program termination. Can be blocked or handled. Sent by pressing **`INTR`** key combination (**`Ctrl+c`**).
3	**`QUIT`**	Keyboard quit	Similar to **`SIGINT`**, but also produces a process dump at termination. Sent by pressing **`QUIT`** key combination (**`Ctrl+\`**).
9	**`KILL`**	Kill, unblockable	Causes abrupt program termination. Cannot be blocked, ignored, or handled; always fatal.
15 ***default***	**`TERM`**	Terminate	Causes program termination. Unlike **`SIGKILL`**, can be blocked, ignored, or handled. The polite way to ask a program to terminate; allows self-cleanup.
18	**`CONT`**	Continue	Sent to a process to resume if stopped. Cannot be blocked. Even if handled, always resumes the process.
19	**`STOP`**	Stop, unblockable	Suspends the process. Cannot be blocked or handled.

SIGNAL NUMBER	SHORT NAME	DEFINITION	PURPOSE
20	**TSTP**	Keyboard stop	Unlike **SIGSTOP**, can be blocked, ignored, or handled. Sent by pressing **SUSP** key combination (**Ctrl+z**).

NOTE
Signal numbers vary on different Linux hardware platforms, but signal names and meanings are standardized. For command use, it is advised to use signal names instead of numbers. The numbers discussed in this section are for Intel x86 systems.

Each signal has a *default action*, usually one of the following:

Term — Cause a program to terminate (exit) at once.
Core — Cause a program to save a memory image (core dump), then terminate.
Stop — Cause a program to stop executing (suspend) and wait to continue (resume).
Programs can be prepared for expected event signals by implementing handler routines to ignore, replace, or extend a signal's default action.

Commands for sending signals by explicit request

Users signal their current foreground process by pressing a keyboard control sequence to suspend (**Ctrl+z**), kill (**Ctrl+c**), or core dump (**Ctrl+**) the process. To signal a background process or processes in a different session requires a signal-sending command.

Signals can be specified either by name (e.g., **-HUP** or **-SIGHUP**) or by number (e.g., **-1**). Users may kill their own processes, but root privilege is required to kill processes owned by others.

- The **kill** command sends a signal to a process by ID. Despite its name, the kill command can be used for sending any signal, not just those for terminating programs.

```
[student@serverX ~]$ kill PID[student@serverX ~]$ kill -signal PID[student@serverX
 ~]$ kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL       5) SIGTRAP
 6) SIGABRT      7) SIGBUS       8) SIGFPE       9) SIGKILL     10) SIGUSR1
11) SIGSEGV     12) SIGUSR2     13) SIGPIPE     14) SIGALRM     15) SIGTERM
16) SIGSTKFLT   17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
-- output truncated --
```

- Use **killall** to send a signal to one or more processes matching selection criteria, such as a command name, processes owned by a specific user, or all system-wide processes.

```
[student@serverX ~]$
killall command_pattern
[student@serverX ~]$
killall -signal command_pattern
[root@serverX ~]#
killall -signal -u username command_pattern
```

- The **pkill** command, like **killall**, can signal multiple processes. **pkill** uses advanced selection criteria, which can include combinations of:

Command — Processes with a pattern-matched command name.
UID — Processes owned by a Linux user account, effective or real.
GID — Processes owned by a Linux group account, effective or real.
Parent — Child processes of a specific parent process.
Terminal — Processes running on a specific controlling terminal.

```
[student@serverX ~]$
pkill command_pattern
[student@serverX ~]$
pkill -signal command_pattern
[root@serverX ~]#
pkill -G GID command_pattern
[root@serverX ~]#
pkill -P PPID command_pattern
[root@serverX ~]#
pkill -t terminal_name -U UID command_pattern
```

LOGGING USERS OUT ADMINISTRATIVELY

The **w** command views users currently logged into the system and their cumulative activities. Use the **TTY** and **FROM** columns to determine the user's location.

All users have a controlling terminal, listed as **pts/*N*** while working in a graphical environment window (*pseudo-terminal*) or **tty*N*** on a system console, alternate console, or other directly connected terminal device. Remote users display their connecting system name in the **FROM** column when using the **-f** option.

```
[student@serverX ~]$ w -f
 12:43:06 up 27 min,  5 users,  load average: 0.03, 0.17, 0.66
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
student  :0       :0               12:20   ?xdm?   1:10   0.16s gdm-session-wor
student  pts/0    :0               12:20    2.00s  0.08s  0.01s w -f
root     tty2                       12:26   14:58   0.04s  0.04s -bash
bob      tty3                       12:28   14:42   0.02s  0.02s -bash
student  pts/1    desktop2.example.12:41     1:07   0.03s  0.03s -bash
[student@serverX ~]$
```

Discover how long a user has been on the system by viewing the session login time. For each session, CPU resources consumed by current jobs, including background tasks and child processes, are in the **JCPU** column. Current foreground process CPU consumption is in the **PCPU** column.

Users may be forced off a system for security violations, resource overallocation, or other administrative need. Users are expected to quit unnecessary applications, close unused command shells, and exit login sessions when requested.

When situations occur in which users cannot be contacted or have unresponsive sessions, runaway resource consumption, or improper system access, their sessions may need to be administratively terminated using signals.

IMPORTANT

Although **SIGTERM** is the default signal, **SIGKILL** is a commonly misused administrator favorite. Since the **SIGKILL** signal cannot be handled or ignored, it is always fatal. However, it forces termination without allowing the killed process to run self-cleanup routines. It is recommended to send **SIGTERM** first, then retry with **SIGKILL** only if a process fails to respond.

Processes and sessions can be individually or collectively signaled. To terminate all processes for one user, use the **pkill** command. Because the initial process in a login session (*session leader*) is designed to handle session termination requests and ignore unintended keyboard signals, killing all of a user's processes and login shells requires using the **SIGKILL** signal.

```
[root@serverX ~]# pgrep -l -u bob
6964 bash
6998 sleep
6999 sleep
7000 sleep
[root@serverX ~]# pkill -SIGKILL -u bob
[root@serverX ~]# pgrep -l -u bob
[root@serverX ~]#
```

When processes requiring attention are in the same login session, it may not be necessary to kill all of a user's processes. Determine the controlling terminal for the session using the **w** command, then kill only processes which reference the same terminal ID. Unless **SIGKILL** is specified, the session leader (here, the **bash** login shell) successfully handles and survives the termination request, but all other session processes are terminated.

```
[root@serverX ~]# pgrep -l -u bob
7391 bash
7426 sleep
7427 sleep
7428 sleep
[root@serverX ~]# w -h -u bob
bob      tty3      18:37    5:04   0.03s  0.03s -bash
[root@serverX ~]# pkill -t tty3
[root@serverX ~]# pgrep -l -u bob
7391 bash
[root@serverX ~]# pkill -SIGKILL -t tty3
[root@serverX ~]# pgrep -l -u bob
[root@serverX ~]#
```

The same selective process termination can be applied using parent and child process relationships. Use the **pstree** command to view a process tree for the system or a single user. Use the parent process's PID to kill all children they have created. This time, the parent **bash** login shell survives because the signal is directed only at its child processes.

```
[root@serverX ~]# pstree -p bob
bash(8391)─┬─sleep(8425)
           ├─sleep(8426)
           └─sleep(8427)
[root@serverX ~]# pkill -P 8391
[root@serverX ~]# pgrep -l -u bob
```

```
bash(8391)
[root@serverX ~]# pkill -SIGKILL -P 8391
[root@serverX ~]# pgrep -l -u bob
bash(8391)
[root@serverX ~]#
```

REFERENCES

`info libc signal` (*GNU C Library Reference Manual*)

- Section 24: Signal Handling

`info libc processes` (*GNU C Library Reference Manual*)

- Section 26: Processes

`kill`(1), **`killall`**(1), **`pgrep`**(1), **`pkill`**(1), **`pstree`**(1), **`signal`**(7), and **`w`**(1) man pages

GUIDED EXERCISE

KILLING PROCESSES

In this lab, students will use keyboard sequences and signals to manage and stop processes.

OUTCOMES

Experience with observing the results of starting and stopping multiple shell processes.

Log in as student to serverX. Start in your home directory.

1. Open two terminal windows, side by side, to be referred to as *left* and *right*.

2. In the left window, start three processes that append text to an output file at one-second intervals. To properly background each process, the complete command set must be contained in parentheses and ended with an ampersand.

```
[student@serverX ~]$
(while true; do echo -n "game " >> ~/outfile; sleep 1; done) &
[student@serverX ~]$
(while true; do echo -n "set " >> ~/outfile; sleep 1; done) &
[student@serverX ~]$
(while true; do echo -n "match " >> ~/outfile; sleep 1; done) &
```

3. In the right window, use **tail** to confirm that all three processes are appending to the file. In the left window, view **jobs** to see all three processes "Running".

```
[student@serverX ~]$ tail -f ~/outfile[student@serverX ~]$ jobs
[1]   Running                 ( while true; do
    echo -n "game " >> ~/outfile; sleep 1;
done ) &
[2]-  Running                 ( while true; do
    echo -n "set " >> ~/outfile; sleep 1;
done ) &
[3]+  Running                 ( while true; do
    echo -n "match " >> ~/outfile; sleep 1;
done ) &
```

4. Suspend the "game" process using signals. Confirm that the "game" process is "Stopped". In the right window, confirm that "game" output is no longer active.

```
[student@serverX ~]$
kill -SIGSTOP %number
[student@serverX ~]$
jobs
```

5. Terminate the "set" process using signals. Confirm that the "set" process has disappeared. In the right window, confirm that "set" output is no longer active.

```
[student@serverX ~]$
kill -SIGTERM %number
[student@serverX ~]$
jobs
```

6. Resume the "game" process using signals. Confirm that the "game" process is "Running". In the right window, confirm that "game" output is again active.

```
[student@serverX ~]$
kill -SIGCONT %number
[student@serverX ~]$
jobs
```

7. Terminate the remaining two jobs. Confirm that no jobs remain and that output has stopped. From the left window, terminate the right window's `tail` command.

   Close extra terminal windows.

```
[student@serverX ~]$
kill -SIGTERM %number
[student@serverX ~]$
kill -SIGTERM %number
[student@serverX ~]$
jobs
[student@serverX ~]$

  pkill -SIGTERM tail

[student@serverX ~]$
```

MONITORING PROCESS ACTIVITY

OBJECTIVES

After completing this section, students should be able to:

- Interpret uptime and load averages.
- Monitor real-time processes.

LOAD AVERAGE

The Linux kernel calculates a *load average* metric as an *exponential moving average* of the *load number*, a cumulative CPU count of active system resource requests.

- *Active requests* are counted from per-CPU queues for running threads and threads waiting for I/O, as the kernel tracks process resource activity and corresponding process state changes.
- *Load number* is a calculation routine run every five seconds by default, which accumulates and averages the active requests into a single number for all CPUs.
- *Exponential moving average* is a mathematical formula to smooth out trending data highs and lows, increase current activity significance, and decrease aging data quality.
- *Load average* is the load number calculation routine result. Collectively, it refers to the three displayed values of system activity data averaged for the last 1, 5, and 15 minutes.

Understanding the Linux load average calculation

The load average represents the perceived system load over a time period. Linux implements the load average calculation as a representation of expected service wait times, not only for CPU but also for disk and network I/O.

- Linux counts not only processes, but threads individually, as separate tasks. CPU request queues for running threads (*nr_running*) and threads waiting for I/O resources (*nr_iowait*) reasonably correspond to process states **R** (*Running*) and **D** (*Uninterruptable Sleeping*). Waiting for I/O includes tasks sleeping for expected disk and network responses.
- The load number is a global counter calculation, which is sum-totaled for all CPUs. Since tasks returning from sleep may reschedule to different CPUs, accurate per-CPU counts are difficult, but an accurate cumulative count is assured. Displayed load averages represent all CPUs.
- Linux counts each physical CPU core and microprocessor hyperthread as separate execution units, logically represented and referred to as individual CPUs. Each CPU has independent request queues. View **/proc/cpuinfo** for the kernel representation of system CPUs.

```
[student@serverX ~]$ grep "model name" /proc/cpuinfo
model name : Intel(R) Core(TM) i5 CPU        M 520  @ 2.40GHz
model name : Intel(R) Core(TM) i5 CPU        M 520  @ 2.40GHz
model name : Intel(R) Core(TM) i5 CPU        M 520  @ 2.40GHz
model name : Intel(R) Core(TM) i5 CPU        M 520  @ 2.40GHz
[student@serverX ~]$ grep "model name" /proc/cpuinfo | wc -l
4
```

- Some UNIX systems only considered CPU utilization or run queue length to indicate system load. Since a system with idle CPUs can experience extensive waiting due to busy disk or network resources, I/O consideration is included in the Linux load average. When experiencing high load averages with minimal CPU activity, examine the disk and network activity.

Interpreting displayed load average values

The three values represent the weighted values over the last 1, 5, and 15 minutes. A quick glance can indicate whether system load appears to be increasing or decreasing. Calculate the approximate *per-CPU* load value to determine whether the system is experiencing significant waiting.

- **`top`**, **`uptime`**, **`w`**, and **`gnome-system-monitor`** display load average values.

```
[student@serverX ~]$ uptime
 15:29:03 up 14 min,  2 users,  load average: 2.92, 4.48, 5.20
```

- Divide the displayed load average values by the number of logical CPUs in the system. A value below 1 indicates satisfactory resource utilization and minimal wait times. A value above 1 indicates resource saturation and some amount of service waiting times.

```
# From /proc/cpuinfo, system has four logical CPUs, so divide by 4:
#                                load average: 2.92, 4.48, 5.20
#           divide by number of logical CPUs:    4     4     4
#                                             ----  ----  ----
#                       per-CPU load average: 0.73  1.12  1.30
#
# This system's load average appears to be decreasing.
# With a load average of 2.92 on four CPUs, all CPUs were in use ~73% of the time.
# During the last 5 minutes, the system was overloaded by ~12%.
# During the last 15 minutes, the system was overloaded by ~30%.
```

- An idle CPU queue has a load number of 0. Each ready and waiting thread adds a count of 1. With a total queue count of 1, the resource (CPU, disk, or network) is in use, but no requests spend time waiting. Additional requests increment the count, but since many requests can be processed within the time period, resource *utilization* increases, but not *wait times*.
- Processes sleeping for I/O due to a busy disk or network resource are included in the count and increase the load average. While not an indication of CPU utilization, the queue count still indicates that users and programs are waiting for resource services.
- Until resource saturation, a load average will remain below 1, since tasks will seldom be found waiting in queue. Load average only increases when resource saturation causes requests to remain queued and counted by the load calculation routine. When resource utilization approaches 100%, each additional request starts experiencing service wait time.

REAL-TIME PROCESS MONITORING

The **`top`** program is a dynamic view of the system's processes, displaying a summary header followed by a process or thread list similar to **`ps`** information. Unlike the static **`ps`** output, **`top`** continuously refreshes at a configurable interval, and provides capabilities for column reordering, sorting, and highlighting. User configurations can be saved and made persistent.

Default output columns are recognizable from other resource tools:

- The process ID (**`PID`**).
- User name (**`USER`**) is the process owner.

- Virtual memory (**VIRT**) is all memory the process is using, including the resident set, shared libraries, and any mapped or swapped memory pages. (Labeled **VSZ** in the **ps** command.)
- Resident memory (**RES**) is the physical memory used by the process, including any resident shared objects. (Labeled **RSS** in the **ps** command.)
- Process state (**S**) displays as:
  - **D** = Uninterruptable Sleeping
  - **R** = Running or Runnable
  - **S** = Sleeping
  - **T** = Stopped or Traced
  - **Z** = Zombie
- CPU time (**TIME**) is the total processing time since the process started. May be toggled to include cumulative time of all previous children.
- The process command name (**COMMAND**).

Fundamental keystrokes in top

KEY	PURPOSE
? *or* **h**	Help for interactive keystrokes.
l, **t**, **m**	Toggles for load, threads, and memory header lines.
1	Toggle showing individual CPUs or a summary for all CPUs in header.
s (1)	Change the refresh (screen) rate, in decimal seconds (e.g., 0.5, 1, 5).
b	Toggle reverse highlighting for *Running* processes; default is bold only.
B	Enables use of bold in display, in the header, and for *Running* processes.
H	Toggle threads; show process summary or individual threads.
u, **U**	Filter for any user name (effective, real).
M	Sorts process listing by memory usage, in descending order.
P	Sorts process listing by processor utilization, in descending order.
k (1)	Kill a process. When prompted, enter **PID**, then **signal**.
r (1)	Renice a process. When prompted, enter **PID**, then **nice_value**.
W	Write (save) the current display configuration for use at the next **top** restart.
q	Quit.
Note:	(1) Not available if top started in secure mode. See **top**(1).

Lab 7: CPU Scheduling

7.a Write a program to implement FCFS CPU Scheduling algorithm

```
import java.util.Scanner;
public class FCFS {
 public static void main(String[] args) {
 int i,n;
    float avgWt,totalWt=0;
    Scanner scan = new Scanner(System.in);
    System.out.print("Enter how many jobs ?\t");
    n = scan.nextInt();
    int  bt[]=  new  int[n];
    int  wt[]=  new  int[n];
    for(i=0;i<n;i++)
    {
     System.out.print("Enter burst time for job "+(i+1)+" :\t");
     bt[i] = scan.nextInt();
    }
    scan.close();
    System.out.print("\n\nWaiting time for Job 1 : 0 units\t");
    wt[0]=0;
    for(i=1;i<n;i++)
    {
       wt[i]=bt[i-1]+wt[i-1];
       System.out.print("\nWaiting time for Job"+(i+1)+" : "+wt[i]+" units \t");
       totalWt = totalWt + wt[i];
    }
    System.out.print("\n\nThe total waiting time : "+totalWt);
    avgWt= totalWt/n;
    System.out.println("\n\nAverage waiting time : "+avgWt);
 }
}
```

Output:

```
Enter how many jobs ?          4
Enter burst time for job 1 :     8
Enter burst time for job 2 :     2
Enter burst time for job 3 :     6
Enter burst time for job 4 :     4
Waiting time for Job 1 : 0 units
Waiting time for Job2 : 8 units
Waiting time for Job3 : 10 units
Waiting time for Job4 : 16 units
The total waiting time : 34.0
Average waiting time : 8.5
```

7.b Write a program to implement SJF CPU Scheduling algorithm

```
import java.util.Scanner;
  public class SJF
  {
      public static void main(String[] args) {
      int i, j, temp, temp2, n;
    float avgWt, totalWt = 0;
    int a[] = new int[10];
    Scanner scan = new Scanner(System.in);

    System.out.print("Enter how many jobs ?\t");
    n = scan.nextInt();
    int bt[] = new int[100];
    int wt[] = new int[100];
    wt[1] = 0;
    for (i = 1; i <= n; i++) {
     System.out.print("Enter burst time for job " + i);
     bt[i] = scan.nextInt();
     a[i] = i; // stores job has how much burst time in array i
    }
    scan.close();
    for (i = 1; i <= n; i++)
     // ascending order of burst times and a[i]

     for (j = i; j <= n; j++)

      if (bt[i] > bt[j]) {
       temp = bt[i];
       bt[i] = bt[j];
       bt[j] = temp;
       temp2 = a[i];
      a[i] = a[j];
       a[j] = temp2;
      }
    System.out.print("\nWaiting time for Job " + a[i] + " : 0 units \t");
    for (i = 2; i <= n; i++) {
     wt[i] = bt[i - 1] + wt[i - 1];
     System.out.print("\nWaiting time for Job" + a[i] + ":\t" + wt[i] + " units \t");
     totalWt = totalWt + wt[i];
    }

   System.out.print("\n\nThe total waiting time : " + totalWt);
    avgWt = totalWt / n;
    System.out.println("\n\nAverage waiting time : " + avgWt);
   }
  }
```

Output:

```
Enter how many jobs ?       4
Enter burst time for job 1 8
Enter burst time for job 2 2
Enter burst time for job 3 6
Enter burst time for job 4 4

Waiting time for Job 2 : 0 units Waiting time for Job4: 2 units Waiting time for Job3: 6 units Waiting time for Job1: 12 units

The total waiting time : 20.0

Average waiting time : 5.0
```

Lab 7.c : Write a program to implement Round Robin CPU Scheduling algorithm

```
import java.util.Scanner;
public class RoundRobin {
 public static void main(String[] args) {
 int i, j, n, r, q, e = 0;
  int bt_c[] = new int[10];
  int bt[] = new int[10];
  int m[] = new int[50];
  float f, avg = 0;
  Scanner scan = new Scanner(System.in);
  System.out.print("\nEnter how many jobs ?\t");
  n = scan.nextInt();
  for (i = 1; i <= n; i++) {
   System.out.print("Enter burst time for job " + i + " :\t");
   bt[i] = scan.nextInt();
  bt_c[i] = bt[i]; // stores job has how much burst time in array i
  }
  System.out.print("\nEnter Quantum (time slice value) :\t");
  q = scan.nextInt();
  int max = 0;
  max = bt[1];
  for (j = 1; j <= n; j++)
  if (max  <=  bt[j])
  max = bt[j];

  if ((max % q) == 0)
  r = (max / q);
  else
   r = (max / q) + 1;
  for (i = 1; i <= r; i++) {
   System.out.print("\n\nRound" + i);
  for (j = 1; j <= n; j++) {
    if (bt[j] > 0) {
     bt[j] = bt[j] - q;

     if (bt[j] <= 0) {
      bt[j] = 0;
      System.out.print("\njob " + j + " is completed");
     } else
      System.out.print("\njob" + j + " remaining time is " + bt[j]);
    }
   }
  }
```

```
 for (i = 1; i <= n; i++) {
 e = 0;
  for (j = 1; j <= r; j++) {
  if (bt_c[i] != 0) {
    if (bt_c[i] >= q) {
     m[i + e] = q;
     bt_c[i] -= q;
    }
    else {
     m[i + e] = bt_c[i];
     bt_c[i] = 0;
    }
   }
   else
    m[i + e] = 0;
    e = e + n;
  }
 }
 for (i = 2; i <= n; i++)
 for (j = 1; j <= i - 1; j++)
  avg = avg + m[j];
 for (i = n + 1; i <= r * n; i++) {
 if (m[i] != 0) {
   for (j = i - (n - 1); j <= i - 1; j++)
    avg = m[j] + avg;
  }
 }
 f = avg / n;
 System.out.print("\n\n\nTOTAL WATING TIME: \t" + avg);
 System.out.print("\n\nAVERAGE WAITING TIME: \t\n" + f);
 scan.close();
 }
}
```

Output:

```
Enter how many jobs ?          4
Enter burst time for job 1 :     8
Enter burst time for job 2 :     2
Enter burst time for job 3 :     6
Enter burst time for job 4 :     4
Enter Quantum (time slice value) : 4
Round1
job1 remaining time is 4
job 2 is completed
job3 remaining time is 2
job 4 is completed Round2
job 1 is completed
job 3 is completed
TOTAL WATING TIME:             38.0
AVERAGE WAITING TIME: 9.5
```

Lab 8: To Demonstrate Mutual Exclusion in Deadlock

Although Java doesn't have a Mutex class, you can mimic a Mutex with the use of a Semaphore of 1. The following example executes two threads with and without locking. Without locking, the programspits out a somewhat random order of output characters ($ or #). With locking, the program spits out nice, orderly character sets of either ##### or $$$$$, but never a mix of # & $

```
import java.util.concurrent.Semaphore;
import java.util.concurrent.ThreadLocalRandom;
public class MutexTest {
  static Semaphore semaphore = new Semaphore(1);
  static class MyThread extends Thread {
    boolean lock;
    char c = ' ';

    MyThread(boolean lock, char c) {
      this.lock = lock;
      this.c = c;
    }

    public void run() {
      try {
        // Generate a random number between 0 & 50
        // The random nbr is used to simulate the "unplanned"
        // execution of the concurrent code
        int randomNbr = ThreadLocalRandom.current().nextInt(0, 50 + 1);

        for (int j=0; j<10; ++j) {
          if(lock) semaphore.acquire();
          try {
            for (int i=0; i<5; ++i) {
              System.out.print(c);
              Thread.sleep(randomNbr);
            }
          } finally {
            if(lock) semaphore.release();
          }
          System.out.print('|');
        }
      } catch (InterruptedException e) {
        e.printStackTrace();
      }
    }
  }
```

```
  public static void main(String[] args) throws Exception {
    System.out.println("Without Locking:");
    MyThread th1 = new MyThread(false, '$');
    th1.start();
    MyThread th2 = new MyThread(false, '#');
    th2.start();

    th1.join();
    th2.join();

    System.out.println('\n');

    System.out.println("With Locking:");
    MyThread th3 = new MyThread(true, '$');
    th3.start();
    MyThread th4 = new MyThread(true, '#');
    th4.start();

    th3.join();
    th4.join();

    System.out.println('\n');
  }
}
```

Output:

```
Without Locking:
$##$#$#$#$|#|$#$#$#$#$|#|$#$#$#$#$|#|$#$#$##$|#$#|$#$#$#$|#$#|$#$#$
#$|#$#|$#$#$#$|#$##|$#$#$|#$#$#|$#$#$|#$#$#|$#$#$|$$|

With Locking:
$$$$$|$$$$$|$$$$$|$$$$$|$$$$$|$$$$$|$$$$$|$$$$$|$$$$$|$$$$$|#####|#
####|#####|#####|#####|#####|#####|#####|#####|#####|
```

Lab 9. Page Replacement Algorithm

9.a Write a program to implement FIFO page replacement algorithm

```
import java.util.HashSet;
import java.util.LinkedList;
import java.util.Queue; public
class FIFO
{
 // Method to find page faults using FIFO
    static int pageFaults(int pages[], int n, int capacity)
    {
        // To represent set of current pages. We use
        // an unordered_set so that we quickly check
        // if a page is present in set or not
        HashSet<Integer> s = new HashSet<>(capacity);

        // To store the pages in FIFO manner
        Queue<Integer> indexes = new LinkedList<>() ;

        // Start from initial page
        int page_faults = 0;
        for (int i=0; i<n; i++)
        {
            // Check if the set can hold more pages
            if (s.size() < capacity)
            {
                // Insert it into set if not present
                // already which represents page fault
                if (!s.contains(pages[i]))
                {
                    s.add(pages[i]);

                    // increment page fault
                    page_faults++;

                    // Push the current page into the queue
                    indexes.add(pages[i]);
                }
            }

            // If the set is full then need to perform FIFO
            // i.e. remove the first page of the queue from
            // set and queue both and insert the current page
```

```
        else
            {
         // Check if current page is not already
                // present in the set
            if(!s.contains(pages[i]))
                {
                    //Pop the first page from the queueint
                    val = indexes.peek();
                    indexes.poll();

                    // Remove the indexes page
                    s.remove(val);

                    // insert the current page
                    s.add(pages[i]);

                    // push the current page into
                    // the queue
                    indexes.add(pages[i]);

                    // Increment page faults
                    page_faults++;
                }
            }
        }

        return page_faults;
    }

    // Driver method
    public static void main(String args[])
    {
          int pages[] = {7, 0, 1, 2, 0, 3, 0, 4,
                         2, 3, 0, 3, 2};

        int capacity = 3;
        System.out.println(pageFaults(pages, pages.length, capacity));
    }
}
```

Output:

10

9.b Write a program to implement ORA page replacement algorithm

```
import java.io.BufferedReader;
import  java.io.IOException;
import java.io.InputStreamReader;
public class OptimalReplacement {

    public static void main(String[] args) throws IOException
    {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        int frames, pointer = 0, hit = 0, fault = 0,ref_len;
        boolean isFull = false;
        int buffer[];
        int reference[];
        int mem_layout[][];

        System.out.println("Please enter the number of Frames: ");
        frames = Integer.parseInt(br.readLine());

        System.out.println("Please enter the length of the Reference string: ");
        ref_len = Integer.parseInt(br.readLine());

        reference = new int[ref_len];
        mem_layout = new int[ref_len][frames];
        buffer = new int[frames];
        for(int j = 0; j < frames; j++)
                buffer[j] = -1;

        System.out.println("Please enter the reference string: ");
        for(int i = 0; i < ref_len; i++)
        {
            reference[i] = Integer.parseInt(br.readLine());
        }
        System.out.println();
        for(int i = 0; i < ref_len; i++)
        {
         int search = -1;
         for(int j = 0; j < frames; j++)
         {
          if(buffer[j] == reference[i])
          {
           search = j;
           hit++;
           break;
          }
         }
```

```
    if(search == -1)
{
 if(isFull)
 {
  int index[] = new int[frames];
  boolean index_flag[] = new boolean[frames];
  for(int j = i + 1; j < ref_len; j++)
  {
   for(int k = 0; k < frames; k++)
   {
    if((reference[j] == buffer[k]) && (index_flag[k] == false))
    {
     index[k] = j;
     index_flag[k] = true;
     break;
    }
   }
  }
  int max = index[0];
  pointer = 0;
  if(max == 0)
   max = 200;
  for(int j = 0; j < frames; j++)
  {
   if(index[j] == 0)
    index[j] = 200;
    if(index[j] > max)
   {
    max = index[j];
    pointer = j;
   }
  }
 }
 buffer[pointer] = reference[i];
 fault++;
 if(!isFull)
 {
  pointer++;
     if(pointer == frames)
     {
      pointer = 0;
      isFull = true;
     }
 }
}
   for(int j = 0; j < frames; j++)
       mem_layout[i][j] = buffer[j];
}
```

```
        for(int i = 0; i < frames; i++)
        {
            for(int j = 0; j < ref_len; j++)
            System.out.printf("%3d ",mem_layout[j][i]);
            System.out.println();
        }
        System.out.println("The number of Hits: " + hit);
        System.out.println("Hit Ratio: " + (float)((float)hit/ref_len));
        System.out.println("The number of Faults: " + fault);
    }
```

Output:

```
Please enter the number of Frames:
3
Please enter the length of the Reference string:
10
Please enter the reference string:
7
0
1
2
0
3
0
4
2
3

  7   7   7   2   2   2   2   2   2   2
 -1   0   0   0   0   0   0   4   4   4
 -1  -1   1   1   1   3   3   3   3   3
The number of Hits: 4
Hit Ratio: 0.4
The number of Faults: 6
```