RH124

Install RHEL 9 Step by Step



Red Hat | Enterprise Linux

MOOCA

Install RHEL 9 Step by Step

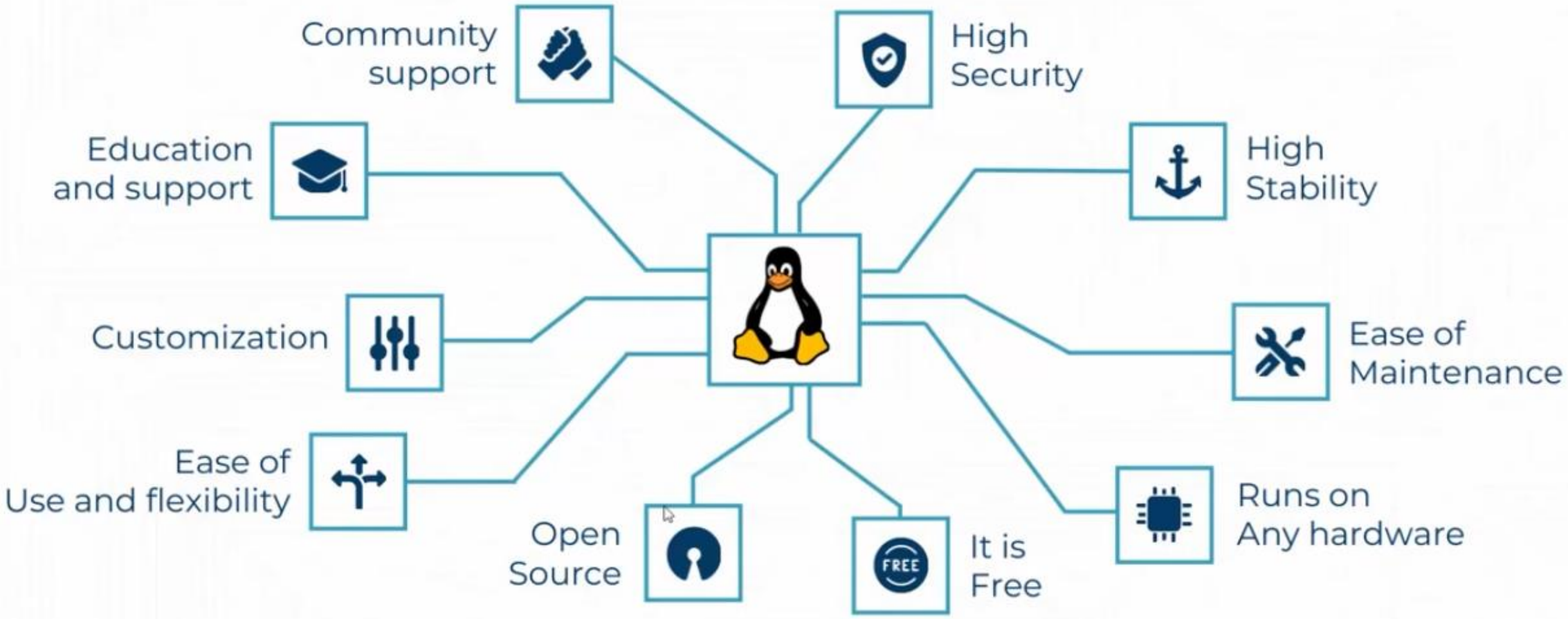
Goal

- Install Red Hat Enterprise Linux 9 virtual machine on VMWare Workstation.

Objectives

- After completing this section, you should be able to install Red Hat Enterprise Linux 9 virtual machine.

Why should you learn about LINUX?



LINUX Distributions



UNIX Operating System Names

A few popular names:

- HP-UX
- IBM AIX
- Sun Solaris
- Mac OS X

LINUX Operating System Names

A few popular names:

- Red Hat Enterprise Linux
- Fedora Linux
- Debian Linux
- Suse Enterprise Linux
- Ubuntu Linux
- CentOS Linux

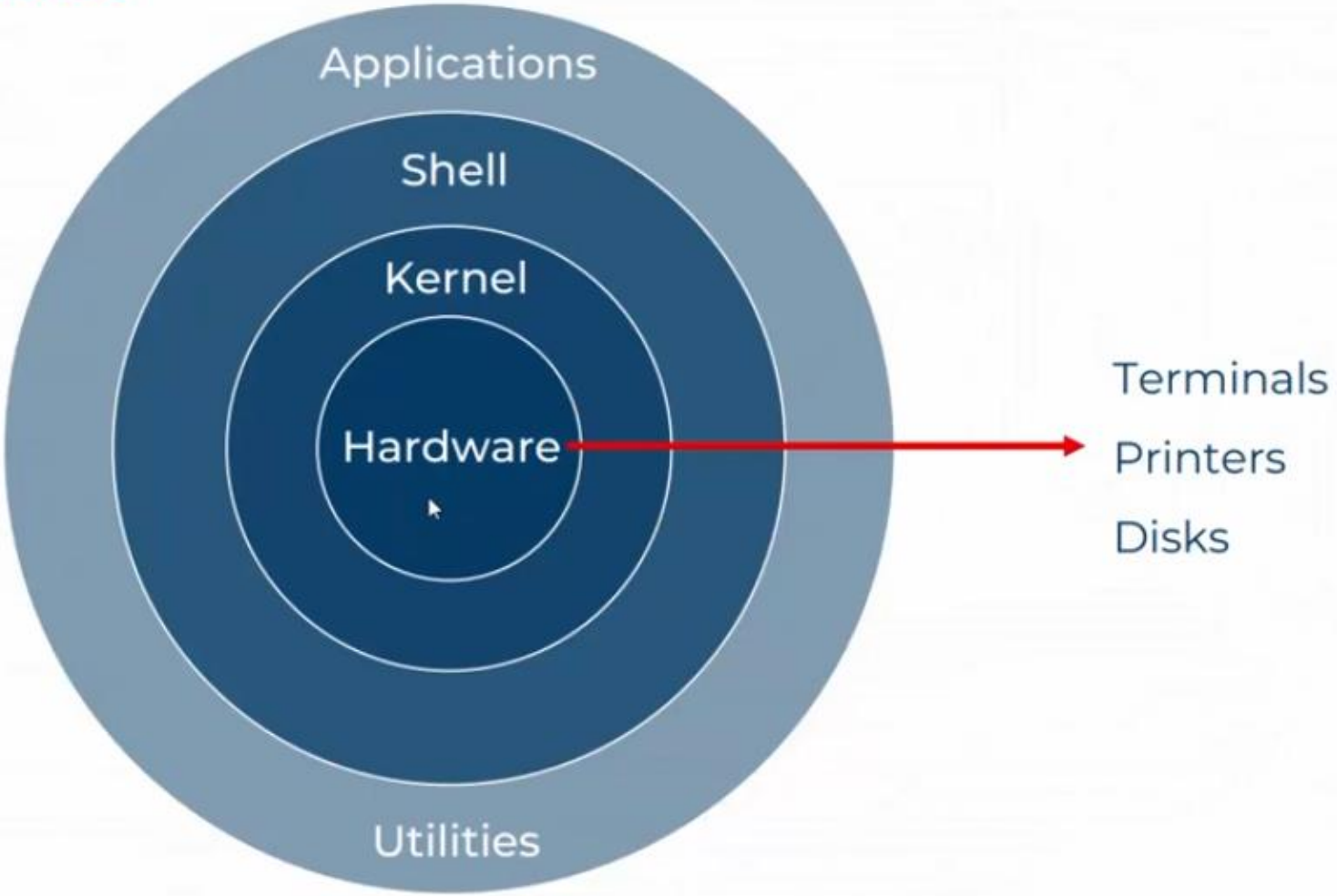
UNIX

- Unix have different pricing depending upon the type of vendor.
- Unix versions are developed by commercial vendors
- The UNIX can be used in servers, workstations, and PCs.
- The source code is not available to anyone.
- Unix users require longer wait time, to get the proper bug fixing patch

LINUX

- Linux is freely distributed, downloaded.
- Linux is Open Source, and thousands of programmer collaborate online and contribute to its development
- Everyone from home users to developers and Servers
- The source is available to the general public
- Threat detection and solution is very fast because Linux is mainly community driven.

LINUX Architecture



WHO IS RED HAT?

- Red Hat is the world's leading provider of open-source software solutions, using a community powered approach to reliable and high-performance cloud, Linux, middleware, storage, and virtualization technologies.

- Red Hat's mission is to be the catalyst in communities of customers, contributors, and partners creating better technology the open source way.



Prerequisites To Install Red Hat Enterprise Linux On VMWare Workstation

RHEL 9 ISO image:

- To download RHEL 9 for free, use following portal

  Red Hat Enterprise Linux product

VMWare Workstation:

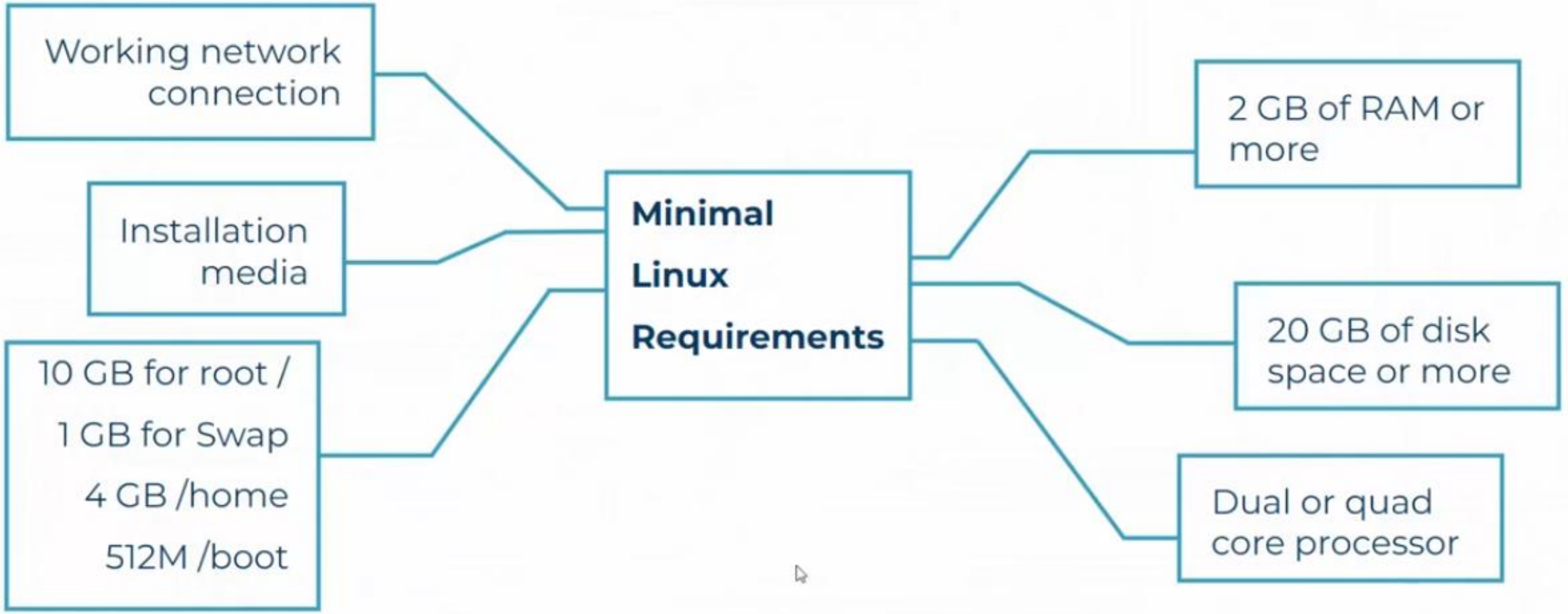
- To download VMWare Workstation, use following portal

  VMWare Workstation

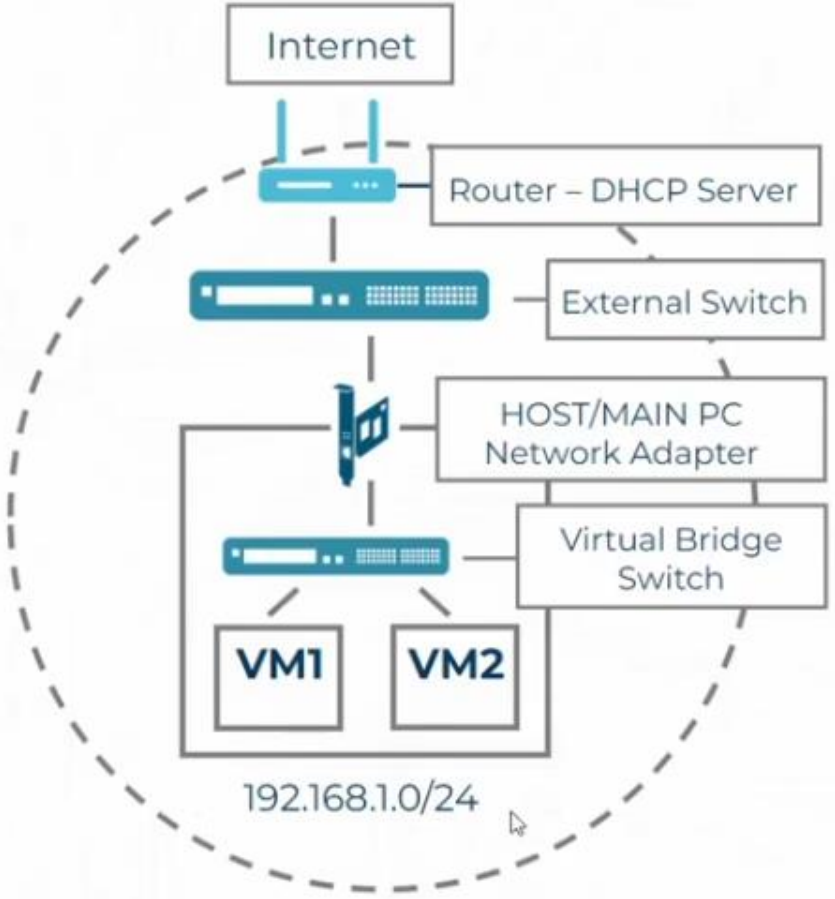
Ensure that hardware virtualization support is turned on in the BIOS settings



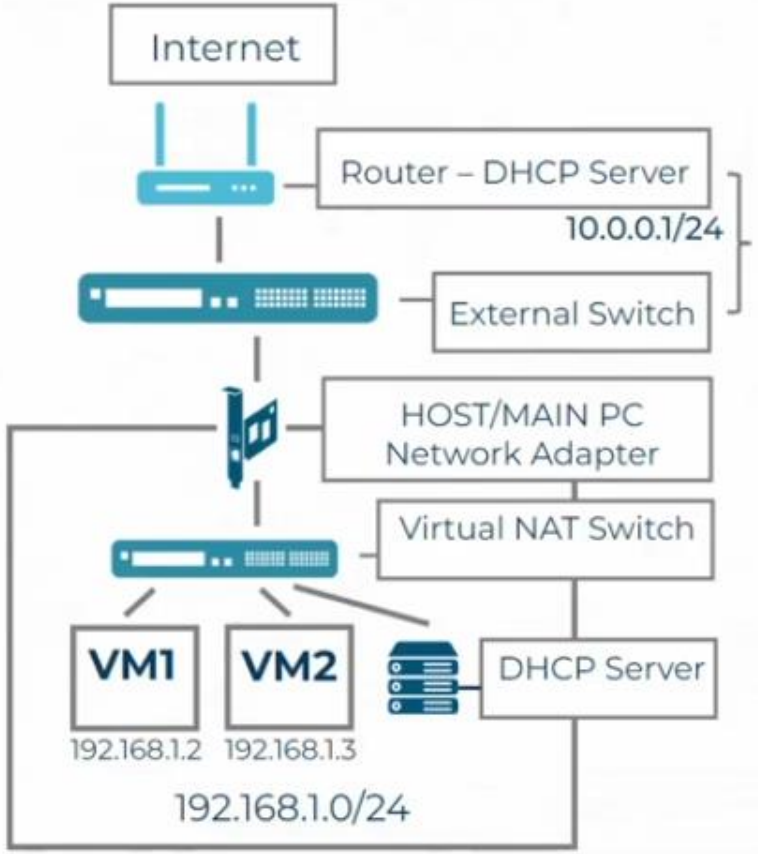
Minimum system requirements for RHEL 9



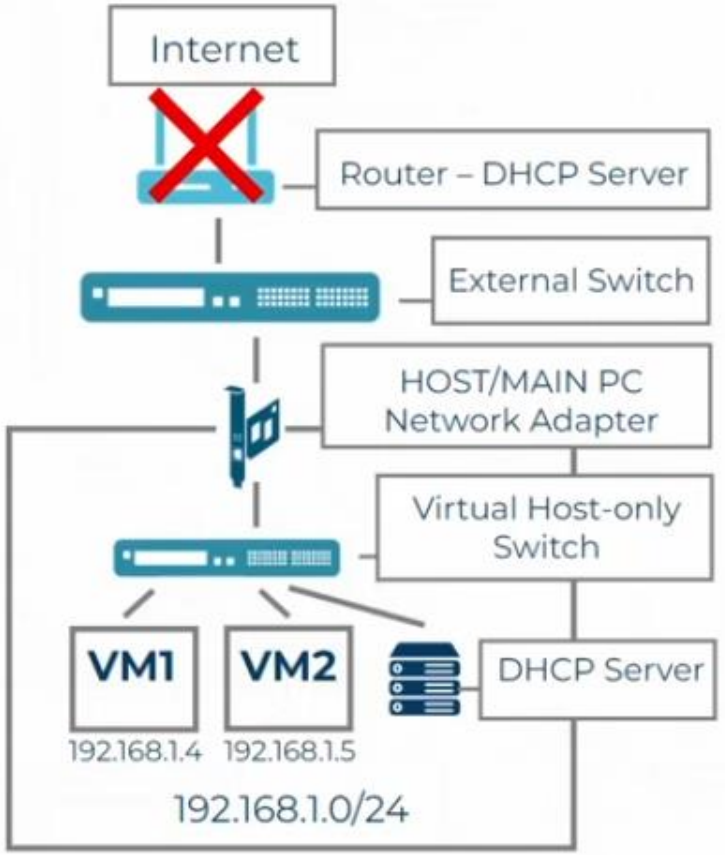
Difference between Bridge vs NAT vs Host-Only (VMware Virtual Network).



Bridge - mode



NAT - mode



Host – Only - mode



RH124

Accessing the Command Line

Red Hat | Enterprise Linux

MOOCA



Accessing the Command Line

Goal

- Log in to a Linux system and run simple commands using the shell.

Objectives

- Understand what is bash shell.
- Learn to write right command syntax.
- Access command line interface using GUI and SSH
- Run simple commands using the shell.
- Learn about terminal shortcuts.

MOOCA

Introduction to the bash shell

- A command line is a text-based interface which can be used to input instructions to a computer system.
- The default shell for users in Red Hat Enterprise Linux is the GNU Bourne-Again Shell (**bash**).
- 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, as shown below.

```
[karim@RHEL9 ~]$
```

- The **$** character is replaced by a **#** character if the shell is running as the superuser, root.

```
[root@RHEL9 ~]#
```

LINUX command syntax

1 command

2 option(s)

3 argument(s)



Right

- $ ls -l /dev
- $ ls -a /dev
- $ mail -s test root
- $ who -u
- $ ls -l -d
- $ ls -ld

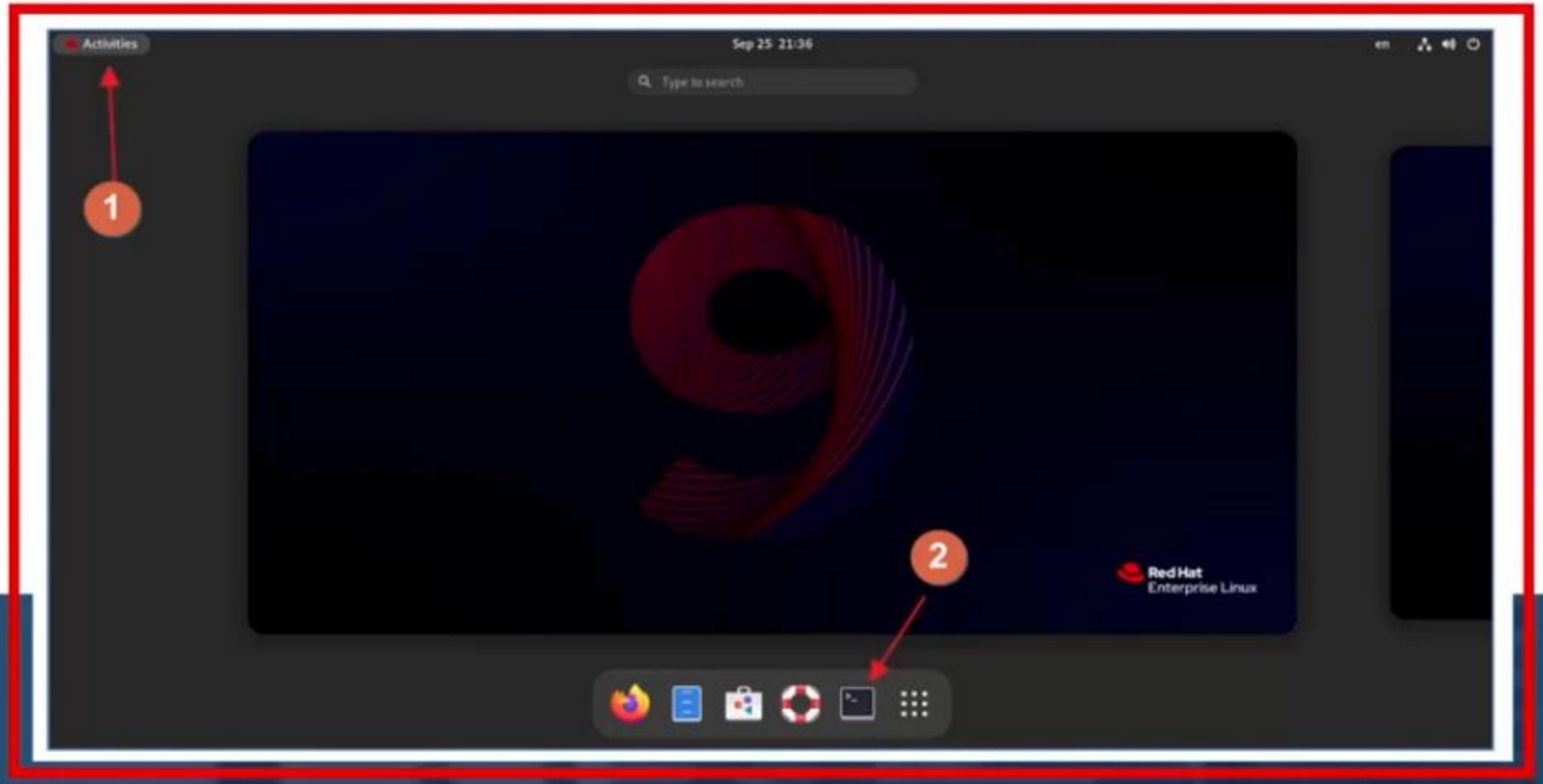
Wrong

- $ ls - l /dev
- $ ls-a /dev
- $ mail test root -s
- $ -u who
- $ ls -l-d
- $ ls -l d



Accessing the command line

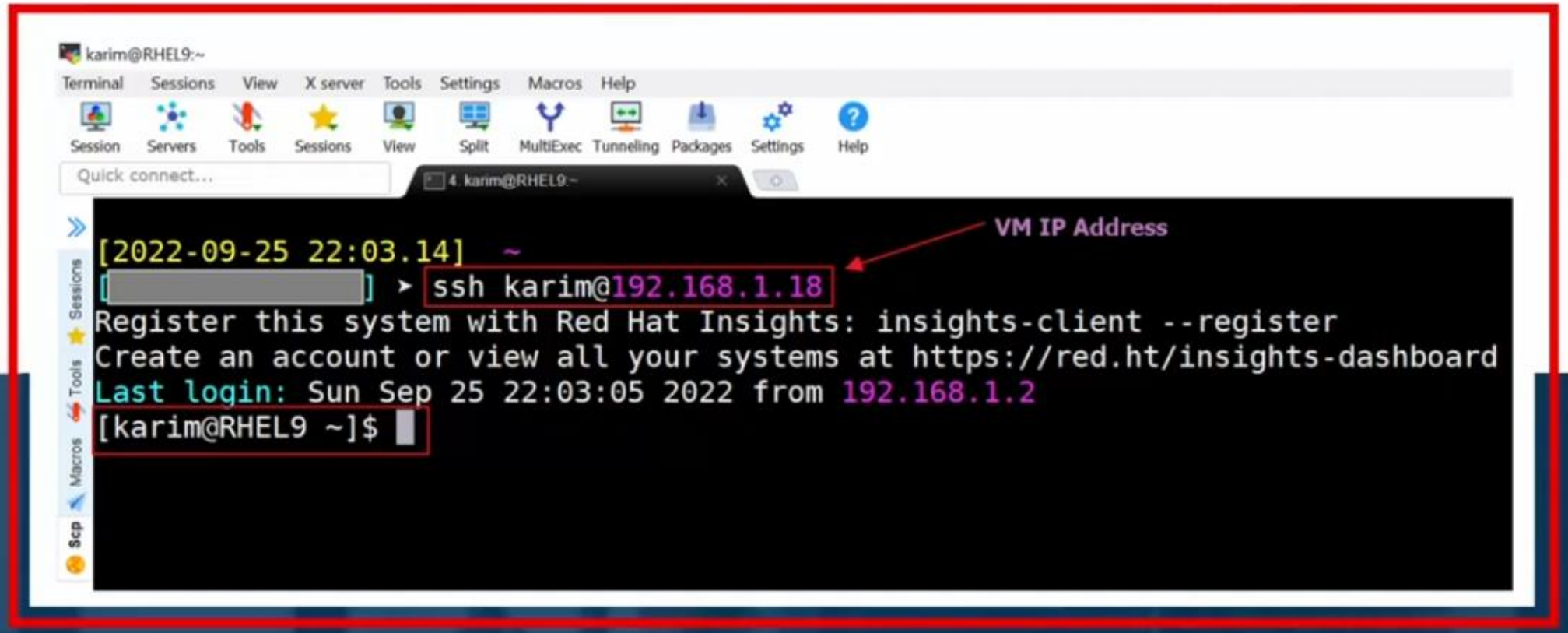
1. From **graphical user interface (GUI)**



Accessing the command line

2. Remote access over Secure Shell (SSH)

You can use any terminal client like MobaXterm to access the command line remotely via **SSH** command



Examples of simple commands

date command

- Displays the current date and time.
- (+) sign specifies a format string for the date command.

```
[karim@RHEL9 ~]$ date
Mon Sep 26 12:17:58 AM EET 2022
[karim@RHEL9 ~]$ date +%R
00:18
[karim@RHEL9 ~]$ date +%x
09/26/2022
[karim@RHEL9 ~]$
```

- It can also be used by the superuser to set the system clock.

Examples of simple commands

passwd command

- Changes a user's own password.

```
[karim@RHEL9 ~]$ passwd
Changing password for user karim.
Current password: old_password
New password: new_password
Retype new password: new_password
passwd: all authentication tokens updated successfully.
```

- The superuser can use the passwd command to change other users' passwords.

Examples of simple commands

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.

```
[karim@RHEL9 ~]$ file /etc/passwd
/etc/passwd: ASCII text
[karim@RHEL9 ~]$ file /home
/home: directory
```

Examples of simple commands

cat command

- View the contents of files.

```
[karim@RHEL9 ~]$ cat /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
```

- Concatenate the contents from multiple files.
- Redirect contents of the file to a terminal or files.

```
[karim@RHEL9 ~]$ cat file1 file2
Hello Mahara-Tech
Introduction to Linux commands.
```

- Display the contents of multiple files.

Examples of simple commands

less command

- Displays **one page** of a file at a time and lets you scroll at your leisure.

```
[karim@RHEL9 ~]$ less /etc/passwd
```

Examples of simple commands

head and **tail** commands

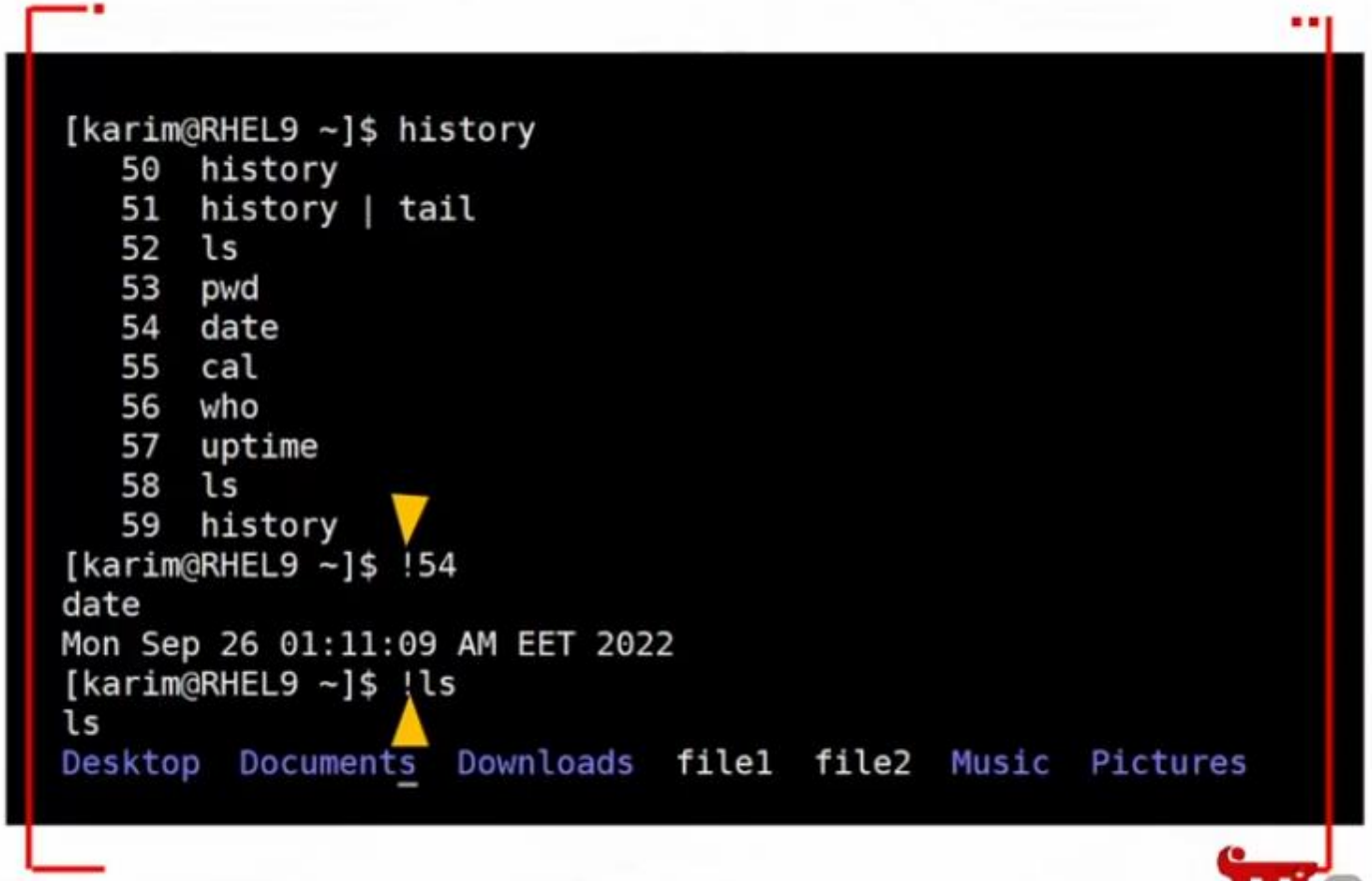
- Display the beginning and end of a file, respectively (10 lines of the file)
- **-n** option allows a different number of lines to be specified

```
[karim@RHEL9 ~]$ head -n 2 /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
[karim@RHEL9 ~]$ tail -n 3 /etc/passwd
tcpdump:x:72:72::/:/sbin/nologin
systemd-oom:x:978:978:systemd Userspace OOM Killer:/:/usr/sbin/nologin
karim:x:1000:1000:Karim:/home/karim:/bin/bash
```

Examples of simple commands

- Displays a list of previously executed commands prefixed with a command number.
- (!) is a metacharacter that is used to expand previous commands without having to retype them.

history command



Terminal shortcuts

TAB completion

- Allows a user to quickly complete commands or file names after 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.

```
[user@host ~]$ pas(1)Tab+Tab
passwd       paste        pasuspender
[user@host ~]$ pass(2)Tab
[user@host ~]$ passwd
Changing password for user user.
Current password:
```

```
[user@host ~]$ ls /etc/pas(1)Tab
[user@host ~]$ ls /etc/passwd(2)Tab
passwd   passwd-
```

Terminal shortcuts

- (\ ) Continuing a long command on another line

```
[karim@RHEL9 ~]$ head -n 3 \
> /usr/share/dict/words \
> /usr/share/dict/linux.words
==> /usr/share/dict/words <==
1080
10-point
10th

==> /usr/share/dict/linux.words <==
1080
10-point
10th
```

Terminal shortcuts

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+LeftArrow	Jump to the beginning of the previous word on the command line.
Ctrl+RightArrow	Jump to the end of the next word on the command line.
Ctrl+R	Search the history list of commands for a pattern.

SUMMARY

In this chapter, you learned:

- The Bash shell is a command interpreter that prompts interactive users to specify Linux commands.
- The file command scans the beginning of a file's contents and displays what type it is.
- The head and tail commands display the beginning and end of a file, respectively.
- You can use Tab completion to complete file names when typing them as arguments to commands.



Managing Files From the Command Line

Goal

- Copy, move, create, delete, and organize files while working from the Bash shell.

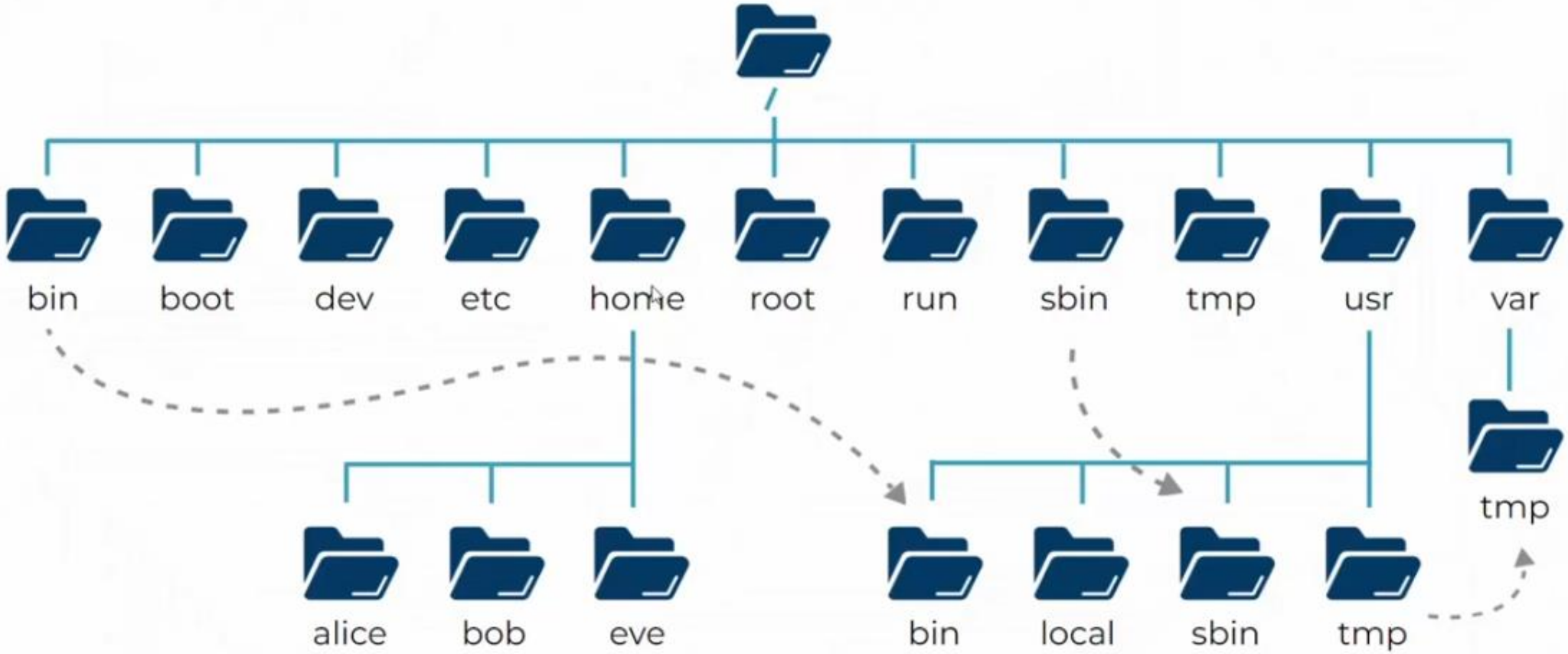
Objectives

- Understand file system hierarchy.
- Specify what type of files are available in Linux.
- Access directories using absolute and relative path.
- List the contents of directories.
- Create, copy, move, and remove files and directories.
- Create hard links and soft links.
- Efficiently run commands affecting many files by using pattern matching features of the Bash shell.

Mooca

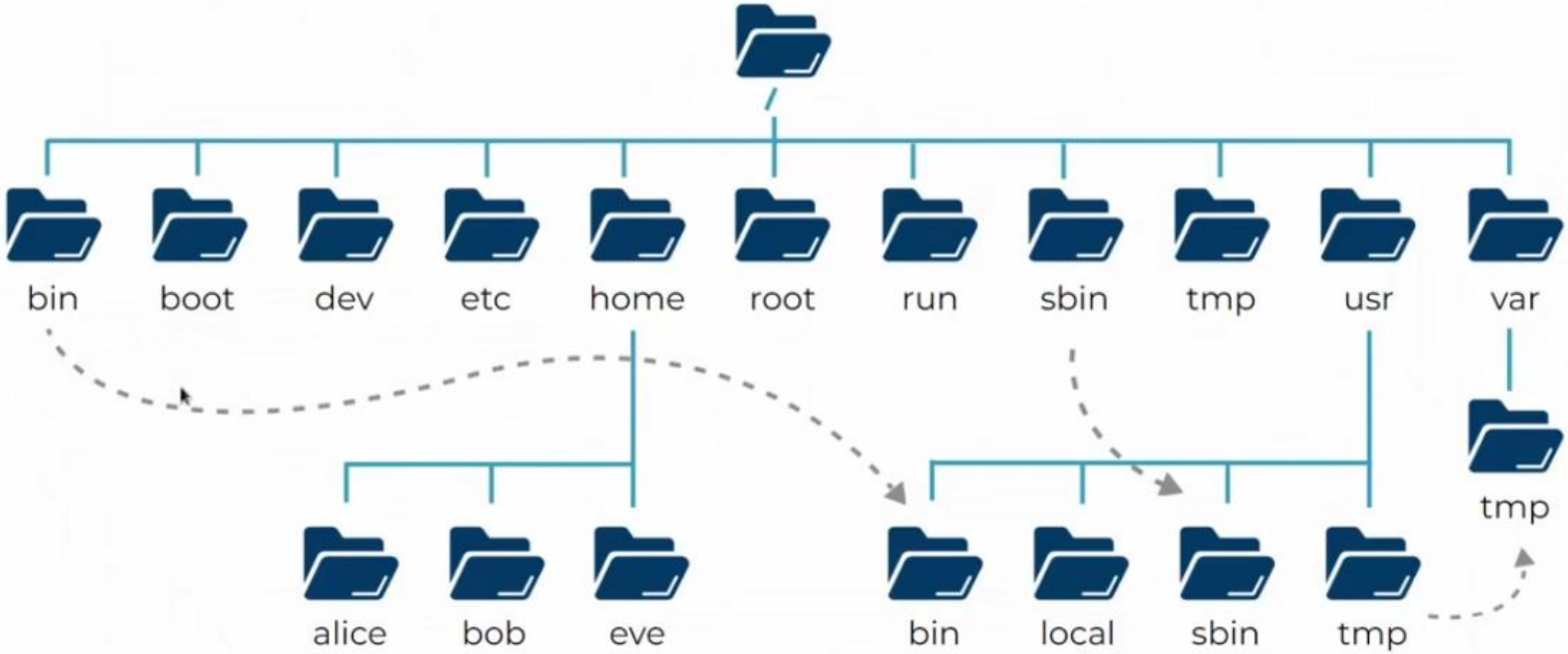
THE FILE-SYSTEM HIERARCHY

- Significant file-system directories in Red Hat Enterprise Linux 9



THE FILE-SYSTEM HIERARCHY

- Significant file-system directories in Red Hat Enterprise Linux 9



Important Red Hat Enterprise Linux Directories

LOCATION	PURPOSE
/usr	Installed software, shared libraries, include files, and 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, such as 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 earlier versions of Red Hat Enterprise Linux.

Important Red Hat Enterprise Linux Directories

/home	*Home directories* are 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* that are used by the system to access hardware.

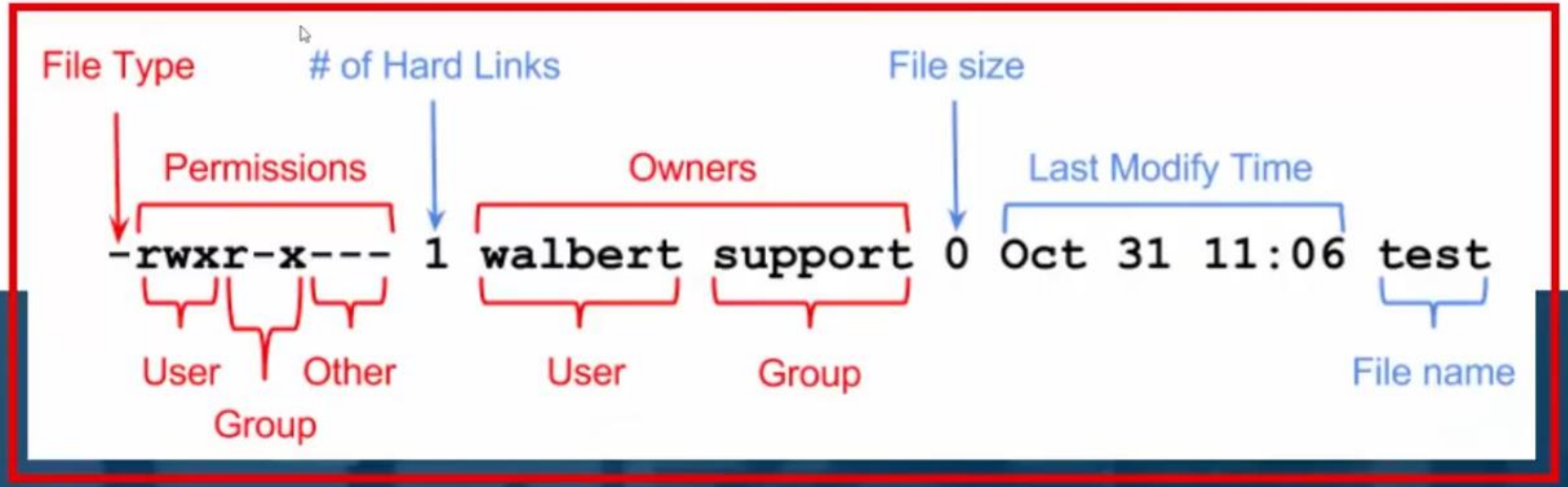
MOOCA

Type of Files in Linux

Symbol	Meaning
-	Regular file
d	Directory
l	Link
c	Special File
s	Socket
p	Named Pipe
b	Block Device

Identify File Types in Linux

```
$ ls -l <file name>
```



Linux File names

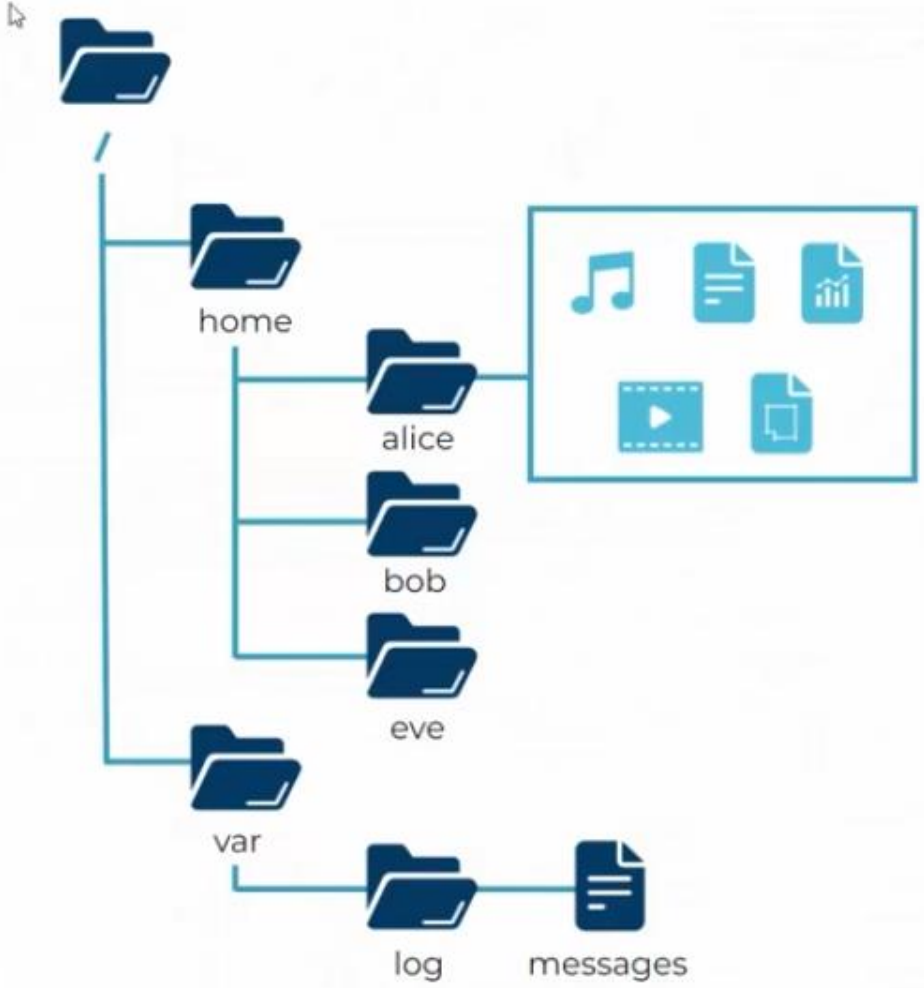
- **Should**
  - Be descriptive
  - Only alphanumeric characters

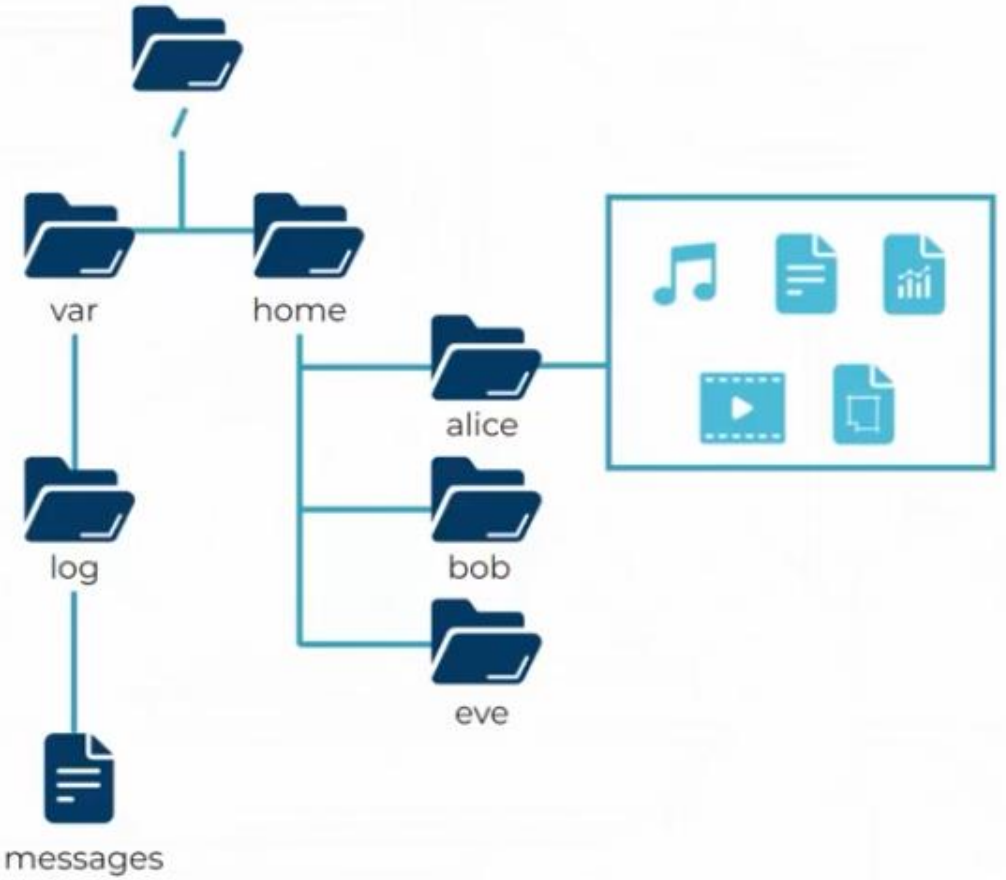
    UPERCASE, lowercase, number, @, _
- **Should Not**
  - Include embedded blanks
  - Contain shell metacharacters

    * ? > < / ; & ! [ ] | \ ‘ “ ( ) { }

- Are case sensitive.
- Filenames starting with a “.“are hidden.
- Maximum number of character for a filename is 255.

Specifying Files By Name





Navigation Paths

pwd command

- Displays the full path name of the current working directory for that shell.

cd command

- Change your shell's current working directory. If you do not specify any arguments to the command, it will change to your home directory.

```
[user@host ~]$ pwd
/home/user
[user@host ~]$ cd Videos
[user@host Videos]$ pwd
/home/user/Videos
[user@host Videos]$ cd /home/user/Documents
[user@host Documents]$ pwd
/home/user/Documents
[user@host Documents]$ cd
[user@host ~]$ pwd
/home/user
```

List Files

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

```
[root@RHEL9 ~]# ls
[root@RHEL9 ~]# ls -l ~
[root@RHEL9 ~]# ls -a
[root@RHEL9 ~]# ls -la
[root@RHEL9 ~]# ls -lh
[root@RHEL9 ~]# ls -R
[root@RHEL9 ~]# ls -t
[root@RHEL9 ~]# ls -r
[root@RHEL9 ~]# dir
[root@RHEL9 ~]# dir --color
```

(long list)

(all files and directories including hidden ones)

(human readable)

(Recursive)

(access time)

(reverse order)

Command – Line File Management

- To manage files, you need to be able to create, remove, copy, and move them. You also need to organize them logically into directories, which you also need to be able to create, remove, copy, and move.

ACTIVITY	COMMAND SYNTAX
Create a directory	`mkdir directory`
Copy a file	`cp file new-file`
Copy a directory and its contents	`cp -r directory new-directory`
Move or rename a file or directory	`mv file new-file`
Remove a file	`rm file`
Remove a directory containing files	`rm -r directory`
Remove an empty directory	`rmdir directory`

Creating Files And Directories

Creating Files

```
[root@RHEL9 ~]# touch file1 FILE1
[root@RHEL9 ~]# touch /root/Desktop/file
[root@RHEL9 ~]# touch {file1,file2,file3,file4}
```

(case sensitive)

If the file exist, it will reset the timestamp of the file.

Creating Files And Directories

Creating Directories

```
[root@RHEL9 ~]# mkdir dir1 dir2 dir3
[root@RHEL9 ~]# mkdir -p dir4/dir5/dir6
[root@RHEL9 ~]# mkdir 'Mahara Tech'
[root@RHEL9 ~]# mkdir " Mahara Tech "
[root@RHEL9 ~]# mkdir Mahara\ Tech
```

(If dir5 not exist it will be created)

Copy And Move Files

Copy files and directories

```
[root@RHEL9 ~]# cp file1 file2
[root@RHEL9 ~]# cp file1 /root/Desktop/
[root@RHEL9 ~]# cp file1 file2 file3 /root/Desktop/
[root@RHEL9 ~]# cp -r /etc dir1
[root@RHEL9 ~]# cp -r /etc/* dir1
[root@RHEL9 Desktop]# cp ~/file1
```

(creates file2)

(last argument must be a directory)

(copy non-empty directory)

(copy the contents of the directory)

Move files and directories

```
[root@RHEL9 ~]# mv file1 new_file1
[root@RHEL9 ~]# mv file1 file2 file3 /root/Documents/
[root@RHEL9 ~]# mv dir1 dir2 dir3 dir4
```

(to rename the file1)

(last argument must be a directory)

(last argument must be a directory)

Copy And Move Files

Copy files and directories

```
[root@RHEL9 ~]# cp file1 file2
[root@RHEL9 ~]# cp file1 /root/Desktop/
[root@RHEL9 ~]# cp file1 file2 file3 /root/Desktop/
[root@RHEL9 ~]# cp -r /etc dir1
[root@RHEL9 ~]# cp -r /etc/* dir1
[root@RHEL9 Desktop]# cp ~/file1
```

(creates file2)

(last argument must be a directory)

(copy non-empty directory)

(copy the contents of the directory)

Move files and directories

```
[root@RHEL9 ~]# mv file1 new_file1
[root@RHEL9 ~]# mv file1 file2 file3 /root/Documents/
[root@RHEL9 ~]# mv dir1 dir2 dir3 dir4
```

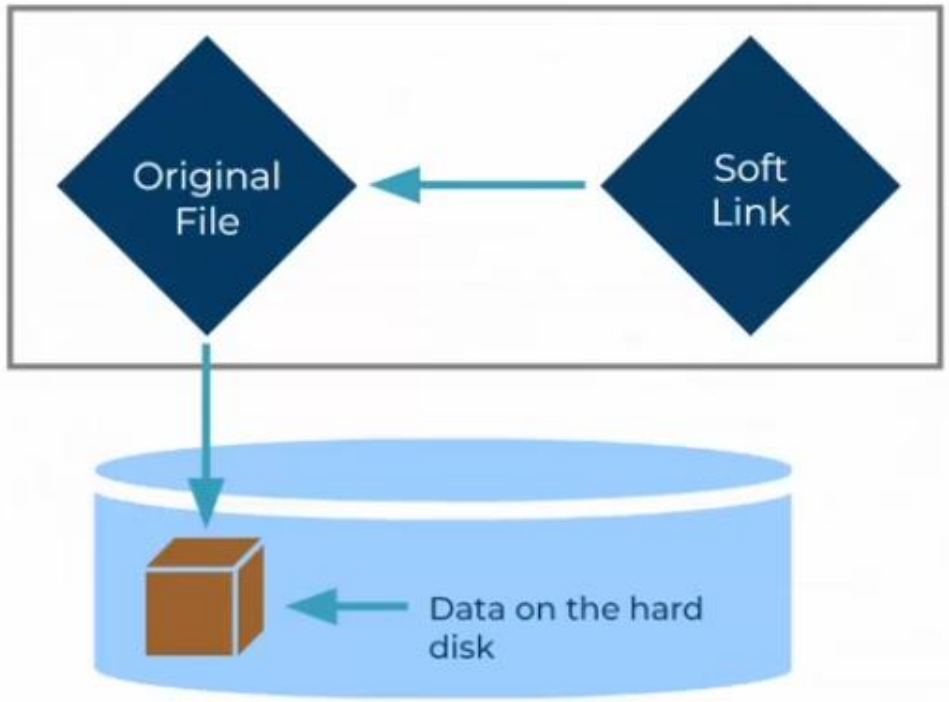
(to rename the file1)

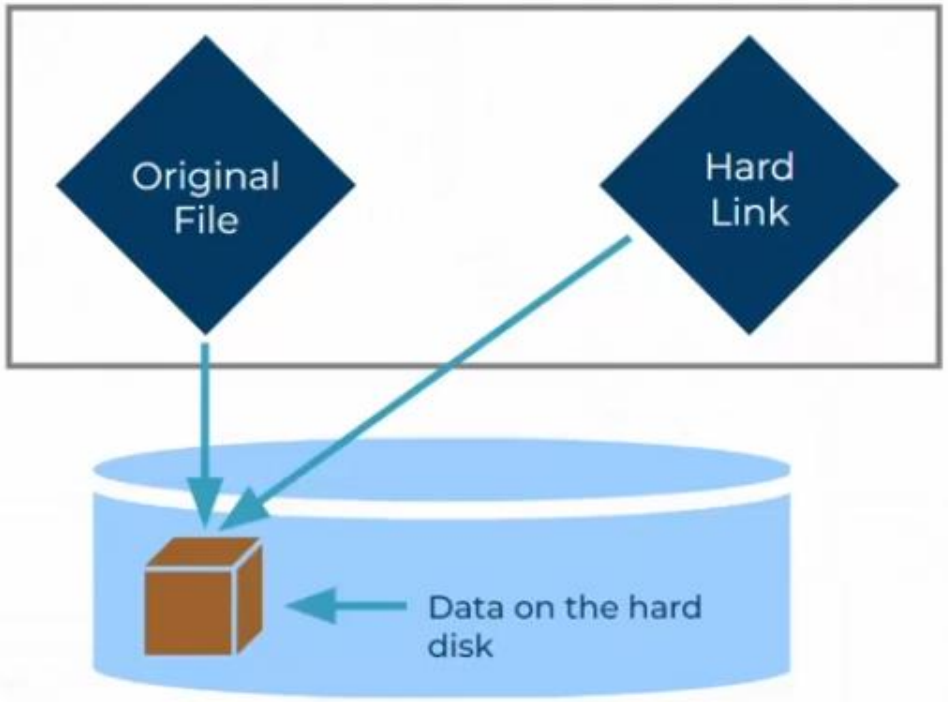
(last argument must be a directory)

(last argument must be a directory)

Hard links and soft links

A link in Linux is a pointer to a file. Like pointers in any programming languages, links in Linux are pointers pointing to a file or a directory. Creating links is a kind of shortcuts to access a file. Links allow more than one file name to refer to the same file, elsewhere.

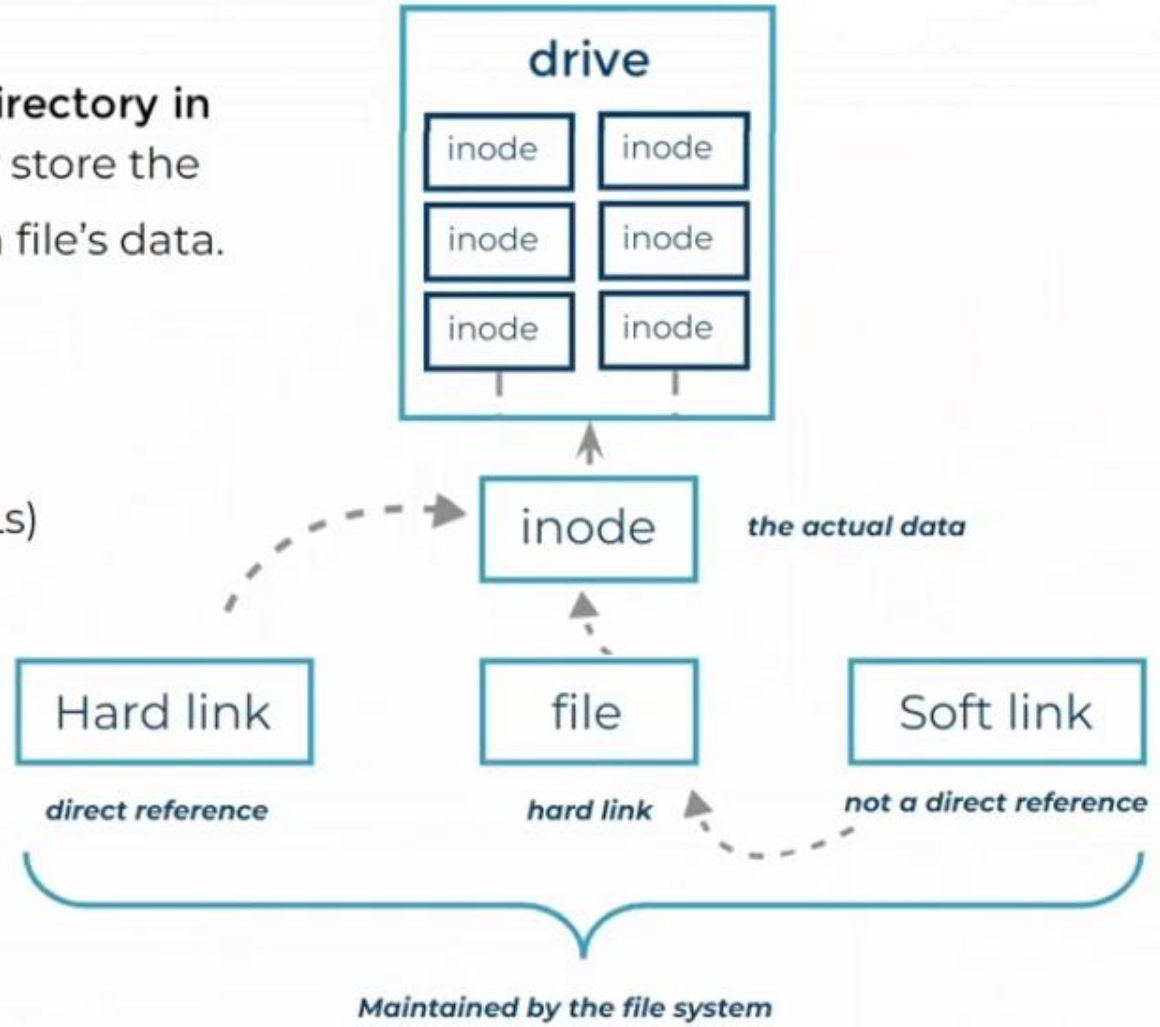




What is inode on Linux Systems

Linux allocate an index node (inode) for every file and directory in filesystem. Inodes do not store actual data. Instead, they store the metadata where you can find the storage blocks of each file's data.

- The following metadata exists in an inode
  - File type
  - Permissions
  - Hard links count
  - Owner ID
  - Group ID
  - Size of file
  - Time stamp (access time)
  - Time stamp (modification time)
  - Soft/Hard Links
  - Access Control List (ACLs)



Create Hard Links

- You can use the **ln** command to create a new hard link (another name) that points to an existing file.
- The following example creates a hard link named **newfile-hlink.txt** for the existing file **newfile.txt.**

```
[karim@RHEL9 ~]$ ls -li newfile.txt
163 -rw-r--r--. 1 karim karim 0 Sep 27 16:43 newfile.txt
[karim@RHEL9 ~]$
[karim@RHEL9 ~]$ ln newfile.txt newfile-hlink.txt
[karim@RHEL9 ~]$
[karim@RHEL9 ~]$ ls -li newfile.txt newfile-hlink.txt
163 -rw-r--r--. 2 karim karim 0 Sep 27 16:43 newfile-hlink.txt
163 -rw-r--r--. 2 karim karim 0 Sep 27 16:43 newfile.txt
```

creates a hard link

same inode

Create Hard Links

- Even if the original file gets deleted, the contents of the file are still available as long as at least one hard link exists. Data is only deleted from storage when the last hard link is deleted.

```
[karim@RHEL9 ~]$ rm -f newfile.txt
[karim@RHEL9 ~]$ ls -li newfile-hlink.txt
163 -rw-r--r--. 1 karim karim 0 Sep 27 16:43 newfile-hlink.txt
```

Creating Soft Links

- The **ln -s** command creates a soft link, which is also called a "symbolic link." A soft link is not a regular file, but a special type of file that points to an existing file or directory.
- The following example, the ln -s command is used to create a new soft link for the existing file **newfile-hlink.txt** that will be named **newfile-symlink.txt.**

```
[karim@RHEL9 ~]$ ln -s newfile-hlink.txt newfile-symlink.txt
[karim@RHEL9 ~]$
[karim@RHEL9 ~]$ ls -li newfile-hlink.txt newfile-symlink.txt
163 -rw-r--r--. 1 karim karim  0 Sep 27 16:43 newfile-hlink.txt
165 lrwxrwxrwx. 1 karim karim 17 Sep 27 17:37 newfile-symlink.txt -> newfile-hlink.txt
```

Hard Links Vs. Soft Links

Soft Link	Hard Link
Soft link is an alias to the original file similar to the shortcut feature in Windows OS.	Hard link is the exact replica of the original file it is pointing to.
It just contain the location to the original file but not the actual data.	It contains the actual content of the file.
Soft links have different Inode values pointing to the original value.	Hard links share the same Inode value pointing to the same file location.
Links can be created across filesystem.	Links cannot be created outside the filesystem.
The link becomes inaccessible when the original file is removed.	Changes in the hard linked file will reflect in the other files.
Soft links can link both to a file or a directory.	Hard links can only link to a file, not a directory.

Pattern Matching

PATTERN	MATCHES
*	Any string of zero or more characters.
?	Any single character.
[*abc...*]	Any one character in the enclosed class (between the square brackets).
[!*abc...*]	Any one character *not* in the enclosed class.
[^*abc...*]	Any one character *not* in the enclosed class.
[[:alpha:]]	Any alphabetic character.
[[:lower:]]	Any lowercase character.
[[:upper:]]	Any uppercase character.
[[:alnum:]]	Any alphabetic character or digit.
[[:punct:]]	Any printable character not a space or alphanumeric.
[[:digit:]]	Any single digit from 0 to 9.
[[:space:]]	Any single white space character. This may include tabs, newlines, carriage returns, form feeds, or spaces.

Basic Regular Expressions

Symbol	Descriptions
.	replaces any character
^	matches start of string
$	matches end of string
*	matches up zero or more times the preceding character
\	Represent special characters
()	Groups regular expressions
?	Matches up exactly one character

Grep command

- -i case insensitive
- -v reverse
- -r recursive
- -n number the lines
- -A3 Display 3 lines after the regular expression match.
- -B3 Display 3 lines before the regular expression match.
- -e Used for multiple search patterns

```
[karim@lab01 ~]$ grep -A3 ^cat$ /usr/share/dict/words
cat
cat.
cata-
catabaptist
[karim@lab01 ~]$ grep -B3 ^cat$ /usr/share/dict/words
Casziel
CAT
Cat
cat
[karim@lab01 ~]$
```

Basic Regular Expressions

Symbol	Descriptions
.	replaces any character
^	matches start of string
$	matches end of string
*	matches up zero or more times the preceding character
\	Represent special characters
()	Groups regular expressions
?	Matches up exactly one character

Grep and Regular Expressions

- used to parse out and or query files for data.

```
[karim@RHEL9 ~]$ grep cat /usr/share/dict/words
[karim@RHEL9 ~]$ grep ^cat /usr/share/dict/words
[karim@RHEL9 ~]$ grep cat$ /usr/share/dict/words
[karim@RHEL9 ~]$ grep ^cat$ /usr/share/dict/words
[karim@RHEL9 ~]$ grep c.t /usr/share/dict/words
[karim@RHEL9 ~]$ grep ^c.t$ /usr/share/dict/words
[karim@RHEL9 ~]$ grep c[aou]t /usr/share/dict/words
[karim@RHEL9 ~]$ grep ^c[aou]t$ /usr/share/dict/words
[karim@RHEL9 ~]$ grep -e cat -e tele /usr/share/dict/words
[karim@RHEL9 ~]$ grep -l ericsson /etc/passwd
[karim@RHEL9 ~]$ ps aux | grep ^root
```

Cut command

- Used for cutting out the sections from each line of files and writing the result to standard output.

```
[karim@RHEL9 ~]$ cut -f 1 -d : /etc/passwd
[karim@RHEL9 ~]$ cut -c 1-7 /etc/passwd
[karim@RHEL9 ~]$ cut -c 25- /etc/passwd
```

tr command

- Utility for translating or deleting characters. It supports a range of transformations including uppercase to lowercase

```
[karim@RHEL9 ~]$ echo hello world | tr a-z A-Z
[karim@RHEL9 ~]$ echo hello world | tr [a-z] [A-Z]
[karim@RHEL9 ~]$ echo hello world | tr [:lower:] [:upper:]
[karim@RHEL9 ~]$ echo "Welcome To Linux" | tr [:space:] '\t'
[karim@RHEL9 ~]$ echo "Welcome To Linux" | tr -d 'W'
[karim@RHEL9 ~]$ echo "my ID is 73535" | tr -d [:digit:]
```

SUMMARY

In this chapter, you learned:

- Files on a Linux system are organized into a single inverted tree of directories, known as a filesystem hierarchy.
- Absolute paths start with a / and specify the location of a file in the file-system hierarchy.
- Relative paths do not start with a / and specify the location of a file relative to the current working directory.
- Five key commands are used to manage files: **mkdir, rmdir, cp, mv**, and **rm**.
- Hard links and soft links are different ways to have multiple file names point to the same data.
- The Bash shell provides pattern matching, expansion, and substitution features to help you efficiently run commands.

RH124

Getting Help in Red Hat Enterprise Linux



Red Hat | Enterprise Linux

mooca



Getting Help in Red Hat Enterprise Linux

Goal

- Resolve problems by using local help systems.

Objectives

- Find information in local Linux system manual pages.
- Use --help option and Red Hat online documentation.

Reading manual pages

- One source of documentation that is generally available on the local system are system manual pages or man pages These pages are shipped as part of the software packages for which they provide documentation, and can be accessed from the command line by using the man command.

Common Sections of the Linux Manual

SECTION	CONTENT TYPE
1	User commands *(both executable and shell programs)*
2	System calls *(kernel routines invoked from user space)*
3	Library functions *(provided by program libraries)*
4	Special files *(such as device files)*
5	File formats *(for many configuration files and structures)*
6	Games *(historical section for amusing programs)*
7	Conventions, standards, and miscellaneous *(protocols, file systems)*
8	System administration and privileged commands *(maintenance tasks)*
9	Linux kernel API *(internal kernel calls)*

Reading manual pages

HEADING	DESCRIPTION
NAME	Subject name. Usually a command or file name. Very brief description.
SYNOPSIS	Summary of the command syntax.
DESCRIPTION	In-depth description to provide a basic understanding of the topic.
OPTIONS	Explanation of the command execution options.
EXAMPLES	Examples of how to use the command, function, or file.
FILES	A list of files and directories related to the man page.
SEE ALSO	Related information, normally other man page topics.
BUGS	Known bugs in the software.
AUTHOR	Information about who has contributed to the development of the topic.

Reading manual pages

- One source of documentation that is generally available on the local system are system manual pages or man pages These pages are shipped as part of the software packages for which they provide documentation, and can be accessed from the command line by using the man command.

Common Sections of the Linux Manual

SECTION	CONTENT TYPE
1	User commands *(both executable and shell programs)*
2	System calls *(kernel routines invoked from user space)*
3	Library functions *(provided by program libraries)*
4	Special files *(such as device files)*
5	File formats *(for many configuration files and structures)*
6	Games *(historical section for amusing programs)*
7	Conventions, standards, and miscellaneous *(protocols, file systems)*
8	System administration and privileged commands *(maintenance tasks)*
9	Linux kernel API *(internal kernel calls)*

Reading manual pages getting

```
[karim@RHEL9 ~]$ man mkdir
[karim@RHEL9 ~]$ man -K passwd
[karim@RHEL9 ~]$ man 1 passwd
[karim@RHEL9 ~]$ man 5 passwd
```

- All man pages are located in **/usr/share/man**. Locate the binary, source, and manual pages located in the /usr/share/man directory by using the **whereis** command.

Reading manual pages getting

```
[karim@RHEL9 ~]$ whereis passwd
passwd: /usr/bin/passwd /etc/passwd /usr/share/man/man5/passwd.5.gz
/usr/share/man/man1/passwd.1ossl.gz /usr/share/man/man1/passwd.1.gz
```

- The --help Option
- Built in the command itself (if supported)

```
[karim@RHEL9 ~]$ who --help
```

- Getting Help From Red Hat

  https://docs.redhat.com
  https://access.redhat.com/documentation/en/

- The Linux documentation project

  http://tldp.org/

- Reading Documentation in:

/usr/share/doc

```
[root@RHEL9 ~]# firefox /usr/share/doc/
```



SUMMARY

In this chapter, you learned:

- Man pages are viewed with the **man** command and provide information on components of a Linux system, such as files, commands, and functions.



Creating, Viewing, and Editing Text Files

Goal

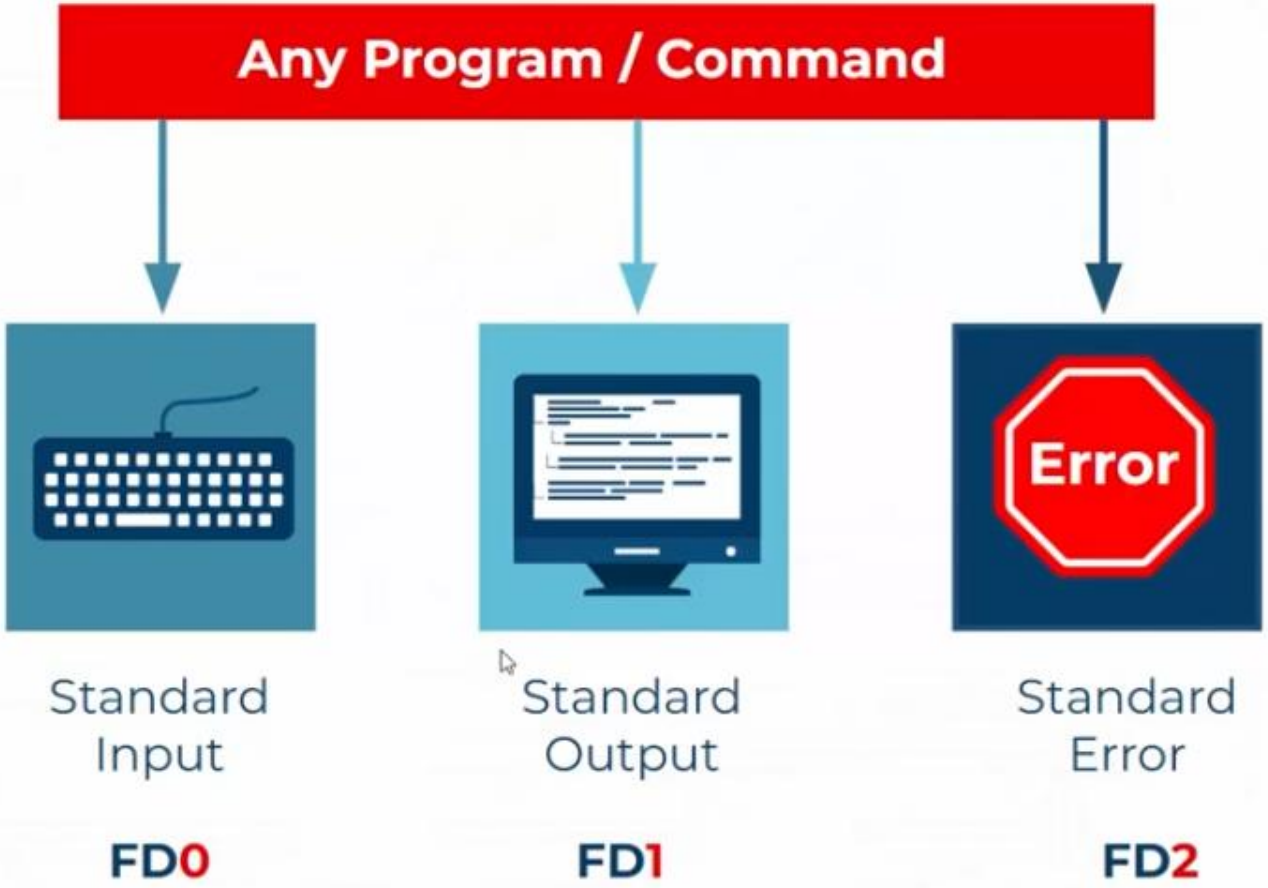
- Create, view, and edit text files from command output or in a text editor.

Objectives

- Save command output or errors to a file with shell redirection, and process command output through multiple command-line programs with pipes.
- Create and edit text files using the vim editor.
- Use shell variables to help run commands, and edit Bash startup scripts to set shell and environment variables to modify the behavior of the shell and programs run from the shell.

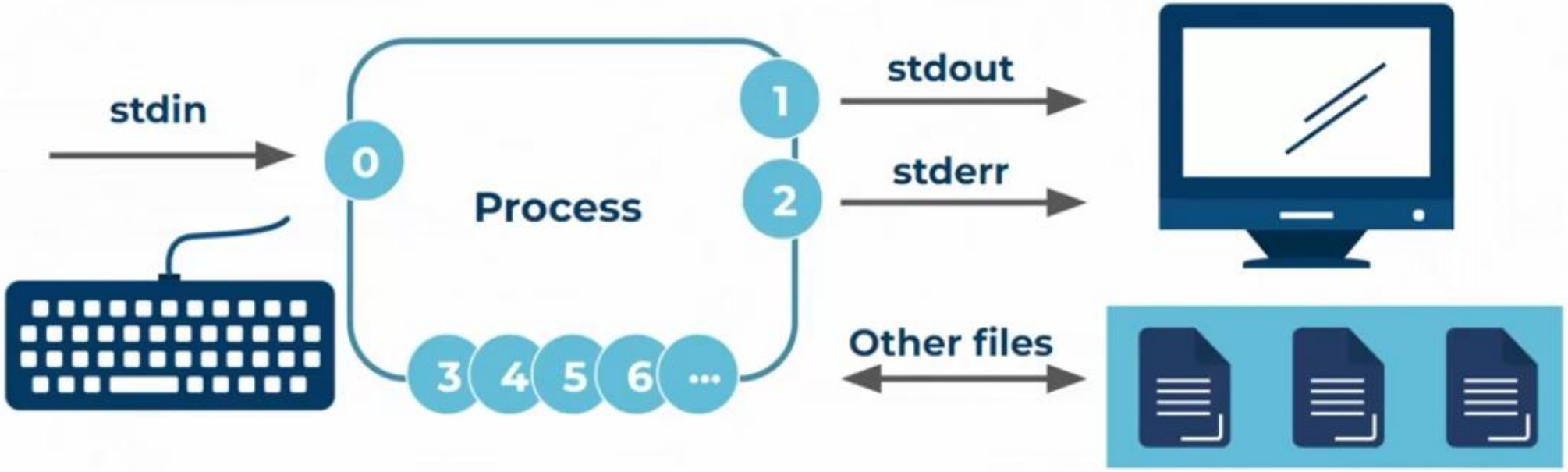
Mooca

Use Input-Output Redirection



What is Redirection?

- A feature in Linux such that when executing a command, you can change the standard input/output devices.



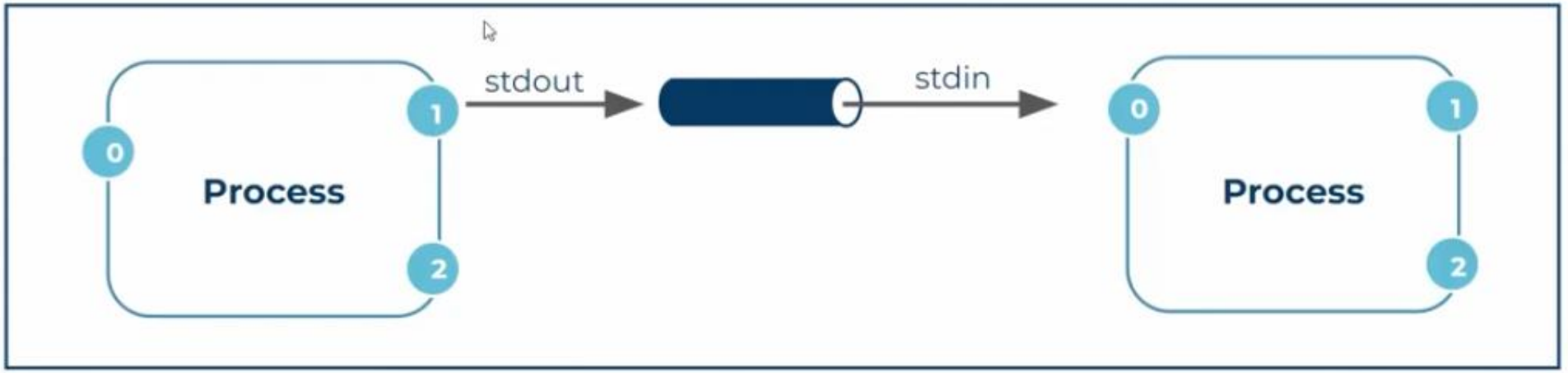
Discard Messages

- The special file **/dev/null** quietly discards channel output redirected to it and is always an empty file.

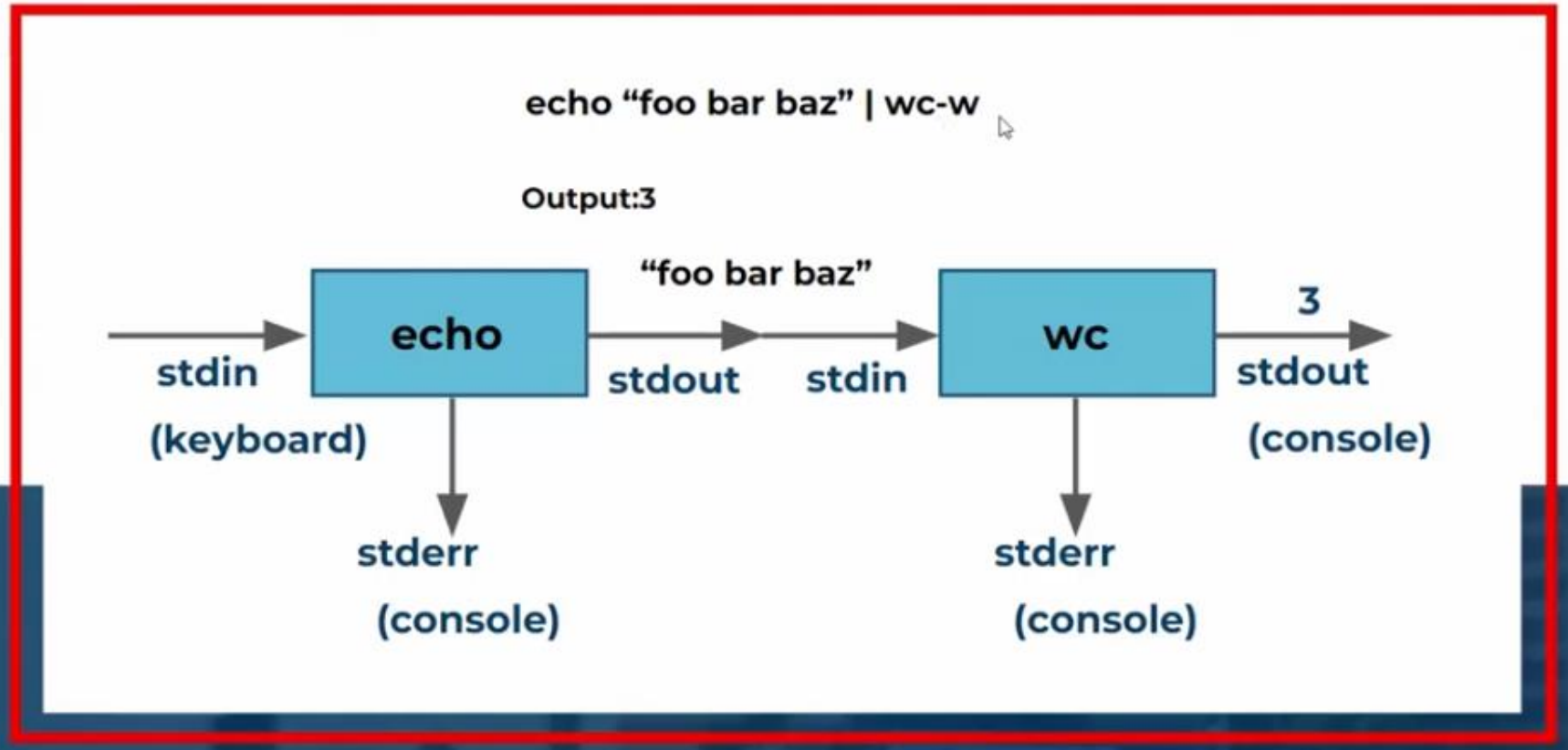
2>/dev/null	discard **stderr** error messages by redirecting to **/dev/null**	
>file 2>&1 &>file	redirect **stdout** and **stderr** to overwrite the same file	
>>file 2>&1 &>>file	redirect **stdout** and **stderr** to append to the same file	

Constructing Pipelines

- 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.



Constructing Pipelines



Pipeline Examples

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

```
[root@RHEL9 ~]# 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.

```
[root@RHEL9 ~]# ls | wc -l
```

Pipeline Examples

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

```
[root@RHEL9 ~]# ls -t | head -n 10 > /tmp/ten-last-changed-files
```

- Save process output and error messages to separate files.

```
[root@RHEL9 ~]# find /etc -name passwd > /tmp/output 2> /tmp/errors
```

Create and edit text files with vim

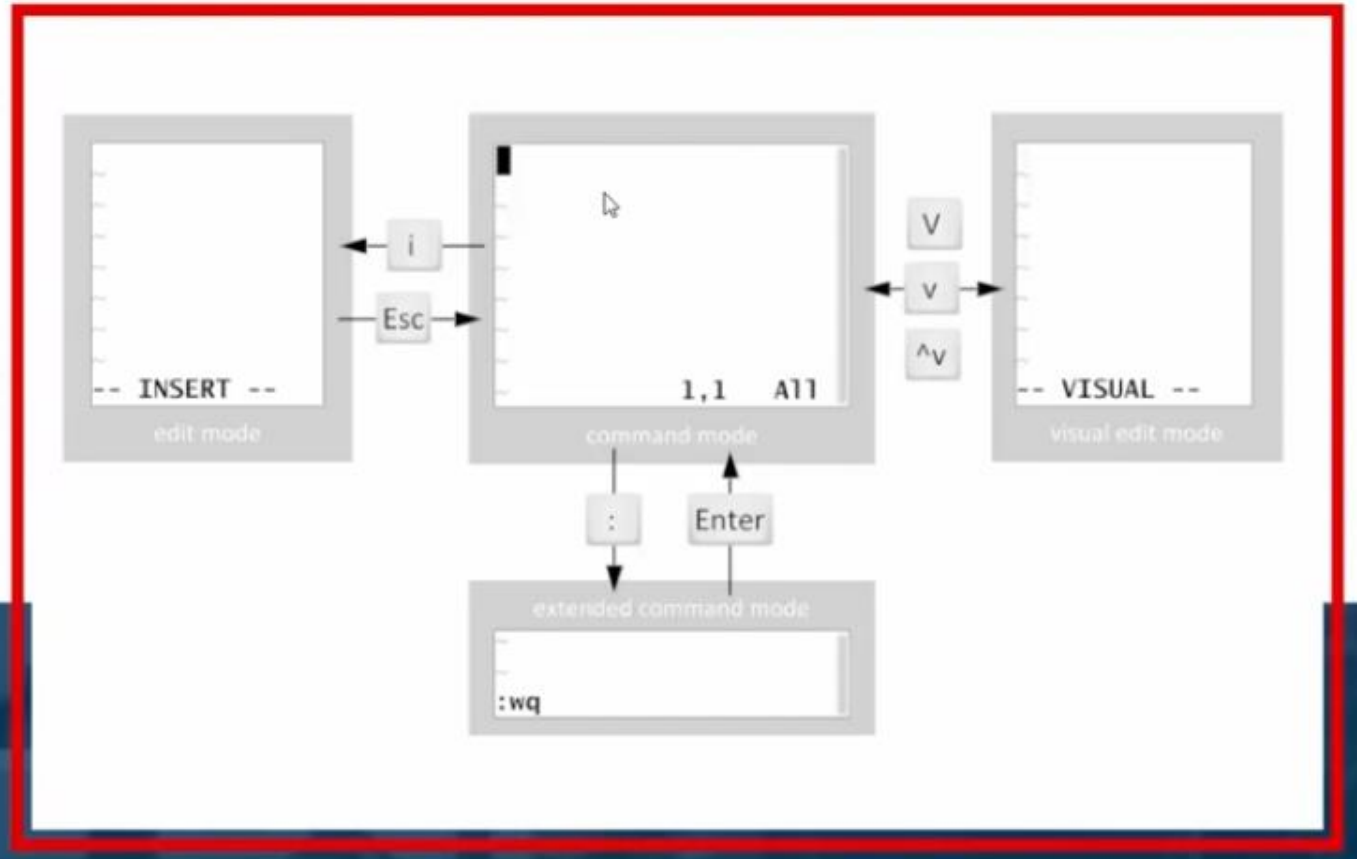
“ You can use any editor you want, but remember that vi is the text editor of the beast”

Richard Stallman



Vim Operating Modes

- Vim has several modes of operation, including **command mode, extended command mode, edit mode, and visual mode**. Depending on the mode, you may be issuing commands, editing text, or working with blocks of text.



Command Mode in Vim

- By default, Vim starts in “command” mode.
- Command mode can be accessed from other modes by pressing Esc.

yy →	Copy a line.
3yy →	Copy three lines.
dd →	Cut a line
p →	Paste after the current cursor.
P →	Paste before the current cursor.(Upper case)
/ →	To search forward
? →	To search backward
n →	Find the next match
N →	Find the previous match

Insert Mode in Vim

- In insert mode you can inserts characters just like a regular text editor.
- You can enter it by using an insert command from normal mode.

- **i** → insert (switches vim to insert mode)
- **a** → append (moves the cursor after the current character and enters insert mode)

- **I** → (moves the cursor to **the beginning** of the line and enters insert mode)
- **A** → (moves the cursor to **the end** of the line and enters insert mode)

- **o** → (inserts a new line **below** the current line and enters insert mode on the new line)
- **O** → (inserts a new line **above** the current one and enters insert mode on the new line)

Insert Mode in Vim

- In insert mode you can inserts characters just like a regular text editor.
- You can enter it by using an insert command from normal mode.

- **i** → insert (switches vim to insert mode)
- **a** → append (moves the cursor after the current character and enters insert mode)

- **I** → (moves the cursor to **the beginning** of the line and enters insert mode)
- **A** → (moves the cursor to **the end** of the line and enters insert mode)

- **o** → (inserts a new line **below** the current line and enters insert mode on the new line)
- **O** → (inserts a new line **above** the current one and enters insert mode on the new line)

Extended Mode in Vim

- Saving files

- **:wq** → Save and quit the current file.
- **:x** → Save the current file if there are unsaved changes, then quit.
- **:w** → Save the current file and remain in editor.
- **:q** → Quit the current file (only if there are no unsaved changes).
- **:q!** → Quit the current file, ignoring any unsaved changes.
- **:10** → Jump to line number 10.

Extended Mode in Vim

- Search and Replace

  - **:%s/old/new/g** → The syntax for replacing every occurrence of a string in the entire text.

- Execute a command in vim

  - **:.!date** → The current cursor position.
  - **:3!date** → In the third line.

- Vim options

  - **:set number** → Enables line numbers.
  - **:set nunumber** → Disable line numbers.

Visual Mode in Vim

The visually identified text can be deleted, copied, changed, or modified with any other Vim editing command.

- There are 3 keystrokes:
  - **Character mode: v** → **VISUAL** will appear at the bottom of the screen.
  - **Line mode: Shift+V** → **VISUAL LINE** will appear at the bottom of the screen.
  - **Block mode: Ctrl+V** → **VISUAL BLOCK** will appear at the bottom of the screen.

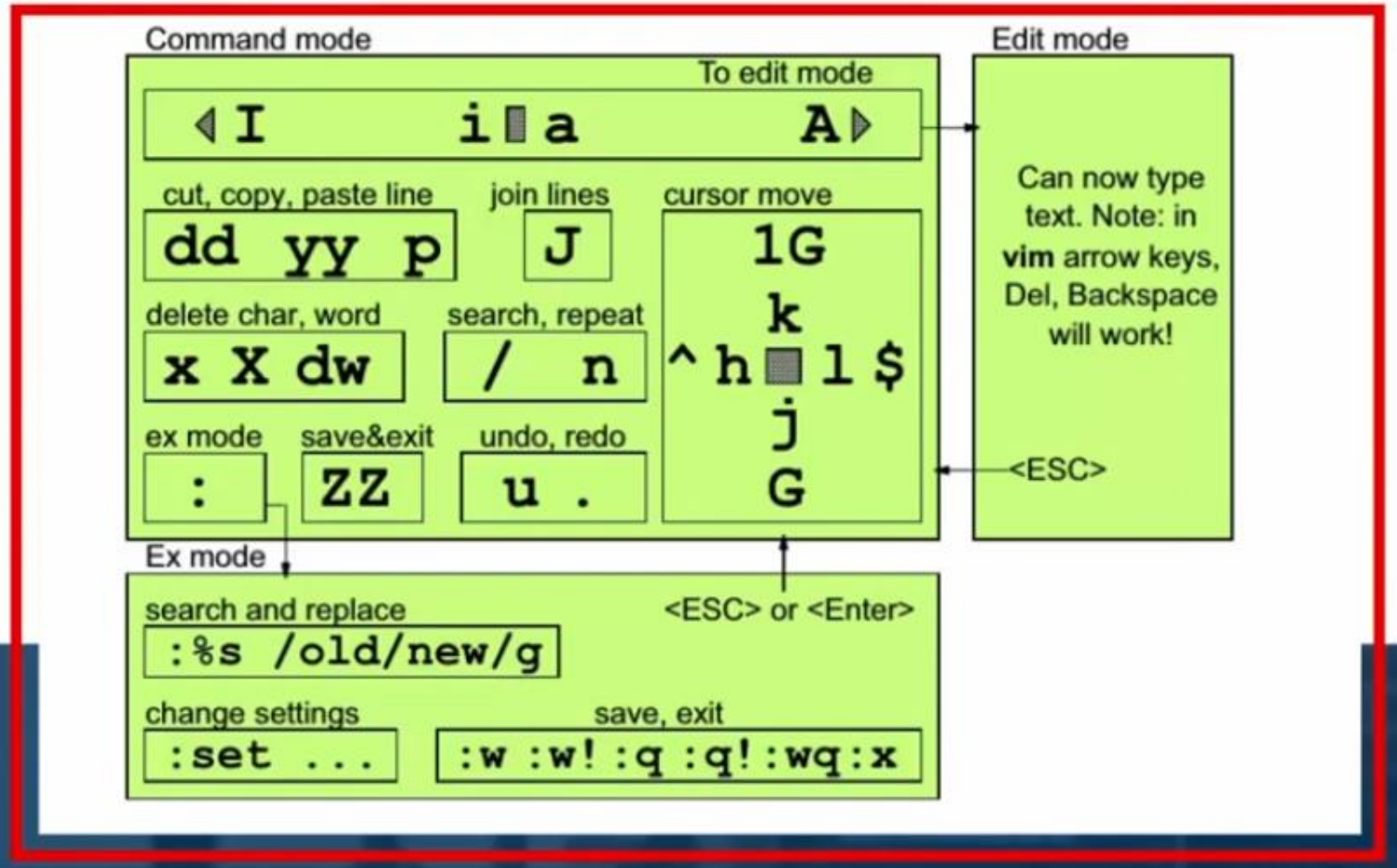
Visual Mode in Vim

- Press **d** key to delete the text.
- Press **y** to copy the task.
- Press **u** to undo and start again.
- Press **p** to paste the text.
- Press **c** to change the highlighted text. The highlighted text will disappear, and you will be in Insert mode where you can add new text

Visual Mode in Vim

- Press **d** key to delete the text.
- Press **y** to copy the task.
- Press **u** to undo and start again.
- Press **p** to paste the text.
- Press **c** to change the highlighted text. The highlighted text will disappear, and you will be in Insert mode where you can add new text

Vim Cheat Sheet





CHANGING THE SHELL ENVIRONMENT

- After completing this section, you should be able to set shell variables to help run commands, and edit Bash startup scripts to set shell and environment variables to modify the behavior of the shell and programs run from the shell.

Assigning Values to Variables

- Assign a value to a shell variable using the following syntax:

```
VARIABLENAME=value
```

- Variable names can contain uppercase or lowercase letters, digits, and the underscore character (_).
- For example, the following commands set shell variables:

```
[user@host ~]$ COUNT=40
[user@host ~]$ first_name=John
[user@host ~]$ file1=/tmp/abc
[user@host ~]$ _ID=RH124
```

Assigning Values to Variables

- Remember, this change only affects the shell in which you run the command, not any other shells you may be running on that server.
- You can use the **set** command to list all shell variables that are currently set

```
[user@host ~]$ set | less
```

PATH

- Contains a list of colon separated directories that contain programs:

```
[user@host ~]$ echo $PATH
/home/user/.local/bin:/home/user/bin:/usr/share/Modules/bin:/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin
```

- When you run a command such as ls, the shell looks for the executable file ls in each of those directories in order, and runs the first matching file it finds. (On a typical system, this is **/usr/bin/ls**.)

Setting variables automatically

- You can edit the Bash startup scripts.
- The exact scripts that run depend on how the shell was started, whether it is an interactive login shell, an interactive non-login shell, or a shell script.
- Assuming the default **/etc/profile**, **/etc/bashrc**, and **~/.bash_profile** files, if you want to make a change to your user account that affects all your interactive shell prompts at startup, edit your **~/.bashrc file**.
- For example, you could set that account's default editor to nano by editing the file to read:

```
# .bashrc

# Source global definitions
if [ -f /etc/bashrc ]; then
 . /etc/bashrc
fi

# User specific environment
PATH="$HOME/.local/bin:$HOME/bin:$PATH"
export PATH
```

Setting variables automatically

- You can edit the Bash startup scripts.
- The exact scripts that run depend on how the shell was started, whether it is an interactive login shell, an interactive non-login shell, or a shell script.
- Assuming the default **/etc/profile**, **/etc/bashrc**, and **~/.bash_profile** files, if you want to make a change to your user account that affects all your interactive shell prompts at startup, edit your **~/.bashrc file**.
- For example, you could set that account's default editor to nano by editing the file to read:

```
# .bashrc

# Source global definitions
if [ -f /etc/bashrc ]; then
 . /etc/bashrc
fi

# User specific environment
PATH="$HOME/.local/bin:$HOME/bin:$PATH"
export PATH
```

Unsetting and unexporting variables

- To **unset** and unexport a variable entirely:

```
[user@host ~]$ echo $x5
[user@host ~]$ unset x
[user@host ~]$ echo $x
[user@host ~]$
```

- To **unexport** a variable without unsetting it :

```
[user@host ~]$ export -n PS1
```

SUMMARY

In this chapter, you learned:

- Use I/O redirection to read standard input from a file or write the output or errors from a process to a file.
- Pipelines can be used to connect standard output from one process to standard input of another process, and can be used to format output or build complex commands.
- You should know how to use at least one command-line text editor, and Vim is generally installed.
- Shell variables can help you run commands and are unique to a particular shell session.
- Environment variables can help you configure the behavior of the shell or the processes it starts.

RH124

Managing Local Users and Groups



Red Hat | Enterprise Linux



Managing Local Users and Groups

Goal

- Create, manage, and delete local users and groups and administer local password policies.

Objectives

- Describe the purpose of users and groups on a Linux system.
- Switch to the superuser account to manage a Linux system, and grant other users superuser access using the **sudo** command.
- Create, modify, and delete locally defined user accounts.
- Create, modify, and delete locally defined group accounts.
- Set a password management policy for users, and manually lock and unlock user accounts.

WHAT IS A USER?

- A user account is used to provide security boundaries between different people and programs that can run commands.
- The system distinguishes user accounts by the unique identification number assigned to them, the user ID or UID.
- There are three main types of user account: the **superuser**, **system users**, and **regular users**.

UID Ranges

- **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.
- **UID 1000+** is the range available for assignment to regular users.
- You can use the **id** command to show information about the currently logged-in user.

```
[user01@host ~]$ id
```

uid=1000(user01) gid=1000(user01) groups=1000(user01)

```
[user01@host]$ id user02
```

uid=1002(user02) gid=1001(user02) groups=1001(user02)

/etc/passwd file used to store information about local users.

- Each line in the **/etc/passwd** file contains information about one user.
- It is divided up into seven colon-separated fields.

```
1user01:2x:31000:41000:5User One:6/home/user01:7/bin/bash
```

1. Username for this user (`user01`).
2. The user's password used to be stored here in encrypted format. That has been moved to the **`/etc/shadow`** file, which will be covered later. This field should always be `x`.
3. The UID number for this user account (`1000`).
4. The GID number for this user account's primary group (`1000`). Groups will be discussed later in this section.
5. The real name for this user (`User One`).
6. The home directory for this user (**`/home/user01`**). This is the initial working directory when the shell starts and contains the user's data and configuration settings.
7. The default shell program for this user, which runs on login (**`/bin/bash`**). For a regular user, this is normally the program that provides the user's command-line prompt. A system user might use **`/sbin/nologin`** if interactive logins are not allowed for that user.

WHAT IS A GROUP?

- Groups can be used to grant access to files to a set of users instead of just a single user.
- Internally, the system distinguishes groups by the unique identification number assigned to them, the group ID or **GID**.
- The mapping of group names to GIDs is defined in databases of group account information.
- By default, systems use the **/etc/group** file to store information about local groups.

```
1group01:2x:310000:4user01,user02,user03
```

1. Group name for this group (`group01`).
2. Obsolete group password field. This field should always be `x`.
3. The GID number for this group (`10000`).
4. A list of users who are members of this group as a supplementary group (`user01`, `user02`, `user03`). Primary (or default) and supplementary groups are discussed later in this section.

Primary Groups and Supplementary Groups

- When you create a new regular user, a new group with the same name as that user is created.
- That group is used as the primary group for the new user, and that user is the only member of this User Private Group.
- Users may also have supplementary groups. Membership in supplementary groups is determined by the **/etc/group** file. Users are granted access to files based on whether any of their groups have access.
- It doesn't matter if the group or groups that have access are primary or supplementary for the user.
- For example, if the user user01 has a primary group user01 and supplementary groups wheel and webadmin, then that user can read files readable by any of those three groups.

The **id** command can also be used to find out about group membership for a user.

```
[user03@host ~]$ id
uid=1003(user03) gid=1003(user03) groups=1003(user03),10(wheel),10000(group01)
 context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
```

Gaining superuser access

- Most operating systems have some sort of superuser, a user that has all power over the system. In Red Hat Enterprise Linux this 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. To perform tasks such as installing or removing software and to manage system files and directories, users must escalate their privileges to the root user.

Switching users

- The **su** command allows users to switch to a different user account. If you run **su** from a regular user account, you will be prompted for the password of the account to which you want to switch. When root runs **su**, you do not need to enter the user's password.

```
[user01@host ~]$ su – karim
```

- If you omit the user name, the **su** or **su** - command attempts to switch to root by default.

```
[user01@host ~]$ su -
Password:
[root@host ~]#
```

Running commands with sudo

- In some cases, the root user's account may not have a valid password at all for security reasons.
- In this case, users cannot log in to the system as root directly with a password, and **su** cannot be used to get an interactive shell. One tool that can be used to get root access in this case is **sudo**.
- Unlike **su**, **sudo** normally requires users to enter their own password for authentication, not the password of the user account they are trying to access. That is, users who use **sudo** to run commands as root do not need to know the root password. Instead, they use their own passwords to authenticate access.
- For example, when **sudo** is configured to allow the user01 user to run the command **usermod** as root, user01 could run the following command to lock or unlock a user account:

```
[user01@host ~]$ sudo usermod -L user01
```

Running commands with sudo

- If a user tries to run a command as another user, and the **sudo** configuration does not permit it, the command will be blocked, the attempt will be logged, and by default an email will be sent to the root user.

```
[user02@host ~]$ sudo tail /var/log/secure

[sudo] password for user02:

[user02@host ~]$
```

user02 is not in the sudoers file.
This incident will be reported.

Configuring Sudo

- The main configuration file for **sudo** is **/etc/sudoers**. To avoid problems if multiple administrators try to edit it at the same time, it should only be edited with the special **visudo** command.
- For example, the following line from the **/etc/sudoers** file enables **sudo** access for members of group wheel.

```
%wheel ALL=(ALL) ALL
```

- In this line, **%wheel** is the user or group to whom the rule applies. A % specifies that this is a group, group wheel. **The ALL=(ALL)** specifies that on any host that might have this file, wheel can run any command. The final **ALL** specifies that wheel can run those commands as any user on the system.

- To enable full **sudo** access for the user user01

```
user01 ALL=(ALL) ALL
```

- To enable full sudo access for the group group01

```
%group01 ALL=(ALL) ALL
```

Creating Users from the Command Line

- The **useradd username** command creates a new user named username. It sets up the user's home directory and account information, and creates a private group for the user named username. At this point the account does not have a valid password set, and the user cannot log in until a password is set.

- The **useradd --help** command displays 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 does not affect existing users.

Modifying Existing Users from the Command Line

- The **usermod --help** command displays the basic options that can be used to modify an account. Some common options include:

USERMOD OPTIONS:	USAGE
`-c, --comment COMMENT`	Add the user's real name to the comment field.
`-g, --gid GROUP`	Specify the primary group for the user account.
`-G, --groups GROUPS`	Specify a comma-separated list of supplementary groups for the user account.
`-a, --append`	Used with the `-G` option to add the supplementary groups to the user's current set of group memberships instead of replacing the set of supplementary groups with a new set.
`-d, --home HOME_DIR`	Specify a particular home directory for the user account.
`-m, --move-home`	Move the user's home directory to a new location. Must be used with the `-d` option.
`-s, --shell SHELL`	Specify a particular login shell for the user account.
`-L, --lock`	Lock the user account.

Creating Groups from the Command Line

```
[user01@host ~]# sudo groupadd -g 10000 group01
[user01@host ~]# tail /etc/group
```

The **-g** option specifies a particular GID for the group to use.

Modifying Existing Groups from the Command Line

```
[user01@host ~]# groupmod -n group0022 group02
```

The **-n** option specifies a new name for the group

Deleting Groups from the Command Line

```
[user01@host ~]# groupdel group0022
```

Changing Group Membership from the Command Line

- Use the **usermod –g** command to change a user's primary group.

```
[user01@host ~]$ id user02
uid=1006(user02) gid=1008(user02) groups=1008(user02)
[user01@host ~]$ sudo usermod -g group01 user02
[user01@host ~]$ id user02
uid=1006(user02) gid=10000(group01) groups=10000(group01)
```

Changing Group Membership from the Command Line

- Use the **usermod -aG** command to add a user to a supplementary group.

```
[user01@host ~]$ id user03
uid=1007(user03) gid=1009(user03) groups=1009(user03)
[user01@host ~]$ sudo usermod -aG group01 user03
[user01@host ~]$ id user03
uid=1007(user03) gid=1009(user03) groups=1009(user03),10000(group01)
```

The use of the **-a** option makes **usermod** function in append mode. Without **-a**, the user will be removed from any of their current supplementary groups that are not included in the **-G** option's list.

Managing user passwords

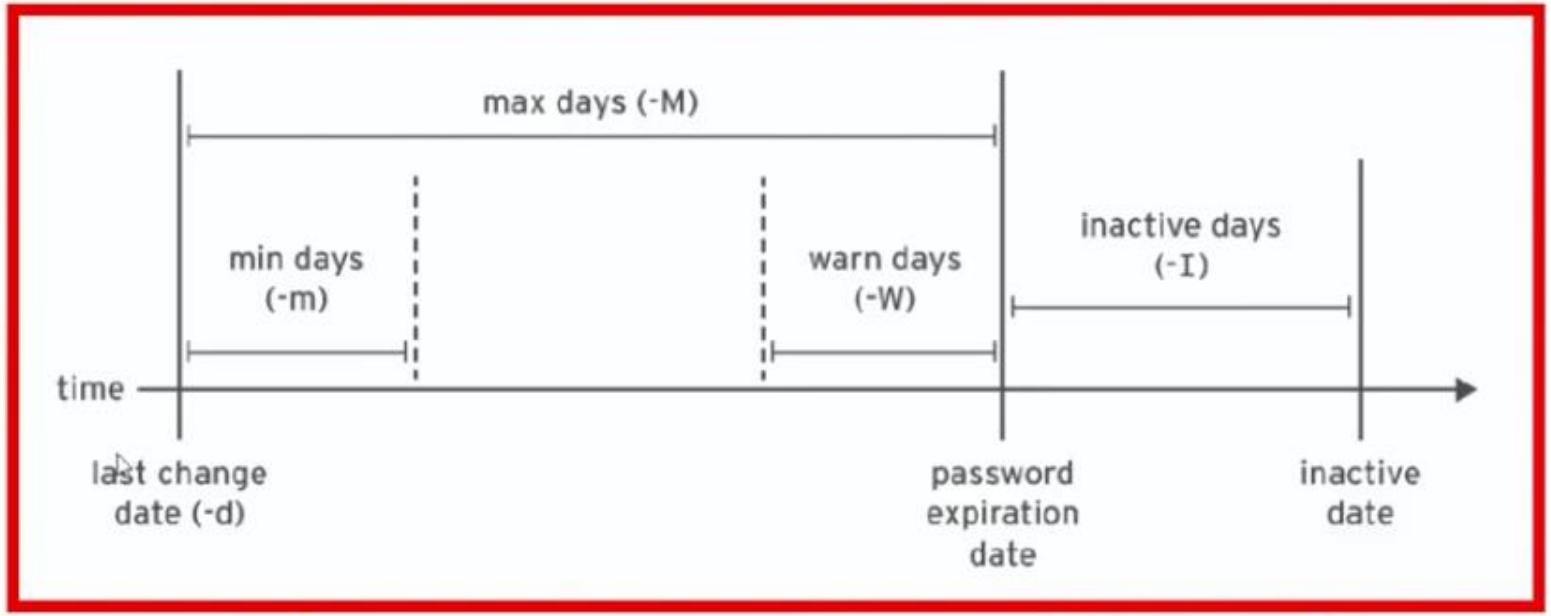
- Like **/etc/passwd**, each user has a line in the **/etc/shadow** file. A sample line from **/etc/shadow** with its nine colon-separated fields is shown below.

```
❶user03:❷$6$CSsX...output omitted...:❸17933:❹0:❺99999:❻7:❼2:❽18113:❾
```

❶ Username of the account this password belongs to.
❷ The *encrypted password* of the user. The format of encrypted passwords is discussed later in this section.
❸ The day on which the password was last changed. This is set in days since 1970-01-01, and is calculated in the UTC time zone.
❹ The minimum number of days that have to elapse since the last password change before the user can change it again.
❺ The maximum number of days that can pass without a password change before the password expires. An empty field means it does not expire based on time since the last change.
❻ Warning period. The user will be warned about an expiring password when they login for this number of days before the deadline.
❼ Inactivity period. Once the password has expired, it will still be accepted for login for this many days. After this period has elapsed, the account will be locked.
❽ The day on which the password expires. This is set in days since 1970-01-01, and is calculated in the UTC time zone. An empty field means it does not expire on a particular date.
❾ The last field is usually empty and is reserved for future use.

Configuring password aging

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




```
[root@host ~]# chage -m 0 -M 90 -W 7 -I 14 user03
```

The preceding **chage** command uses the **-m**, **-M**, **-W**, and **-I** options to set the minimum age, maximum age, warning period, and inactivity period of the user's password, respectively.

Restricting access

- The **usermod** command can lock an account with the **-L** option.

```
[user01@host ~]$ sudo usermod -L user03
[user01@host ~]$ su - user03
Password: redhat
su: Authentication failure
```

- If a user leaves the company, the administrator may lock and expire an account with a single **usermod** command.

```
[root@host ~]# usermod -L -e 2021-12-14 karim
```

- The preceding **usermod** command uses the **-e** option to set the account expiry date for the given user account.
- The **-L** option locks the user's password.

The nologin Shell

- The **nologin** shell acts as a replacement shell for the user accounts not intended to interactively log into the system.
- It is wise from the security standpoint to disable the user account from logging into the system when the user account serves a responsibility that does not require the user to log into 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 in to the system directly, the **nologin** shell closes the connection.

```
[root@host ~]# usermod -s /sbin/nologin user03
[root@host ~]# su - user03
```

This account is currently not available.

SUMMARY

In this chapter, you learned:

- There are three main types of user account: the superuser, system users, and regular users.
- A user must have a primary group and may be a member of one or more supplementary groups.
- The three critical files containing user and group information are **/etc/passwd**, **/etc/group**,and **/etc/shadow**.
- The **su** and **sudo** commands can be used to run commands as the superuser.
- The **useradd**, **usermod**, and **userdel** commands can be used to manage users.
- The **groupadd**, **groupmod**, and **groupdel** commands can be used to manage groups.
- The **chage** command can be used to configure and view password expiration settings for users.

RH124

Controlling Access to Files



Red Hat | Enterprise Linux

mooca



Controlling Access to Files

Goal

- Set Linux file-system permissions on files and to interpret the security effects of different permission settings.

Objectives

- List the file system permissions on files and directories, and interpret the effect of those permissions on access by users and groups.
- Change the permissions and ownership of files using command-line tools.
- Control the default permissions of new files created by users, explain the effect of special permissions, and use special permissions and default permissions to set the group owner of files created in a particular directory.

MOOCA

Linux file-system permissions

- File permissions control access to files. The Linux file permissions system is simple but flexible, which makes it easy to understand and apply, yet still able to handle most normal permission cases easily.

- Files have 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.

- Three categories of permissions apply: **read**, **write**, and **execute**.

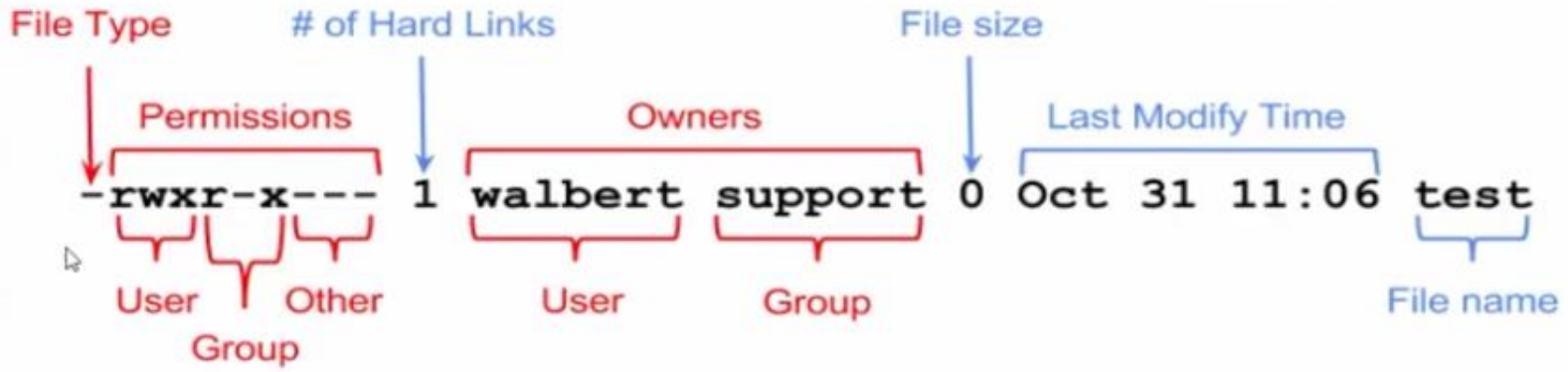
Effects of Permissions on Files and Directories

PERMISSION	EFFECT ON FILES	EFFECT ON DIRECTORIES
r (read)	Contents of the file can be read.	Contents of the directory (the file names) can be listed.
w (write)	Contents of the file can be changed.	Any file in the directory can be created or deleted.
x (execute)	Files can be executed as commands.	Contents of the directory can be accessed. (You can change into the directory, read information about its files, and access its files if the files' permissions allow it.)

Viewing file and directory permissions and ownership

The **-l** option of the ls command shows more detailed information about file permissions and ownership,

You can use the **-d** option to show detailed information about a directory itself, and not its contents.



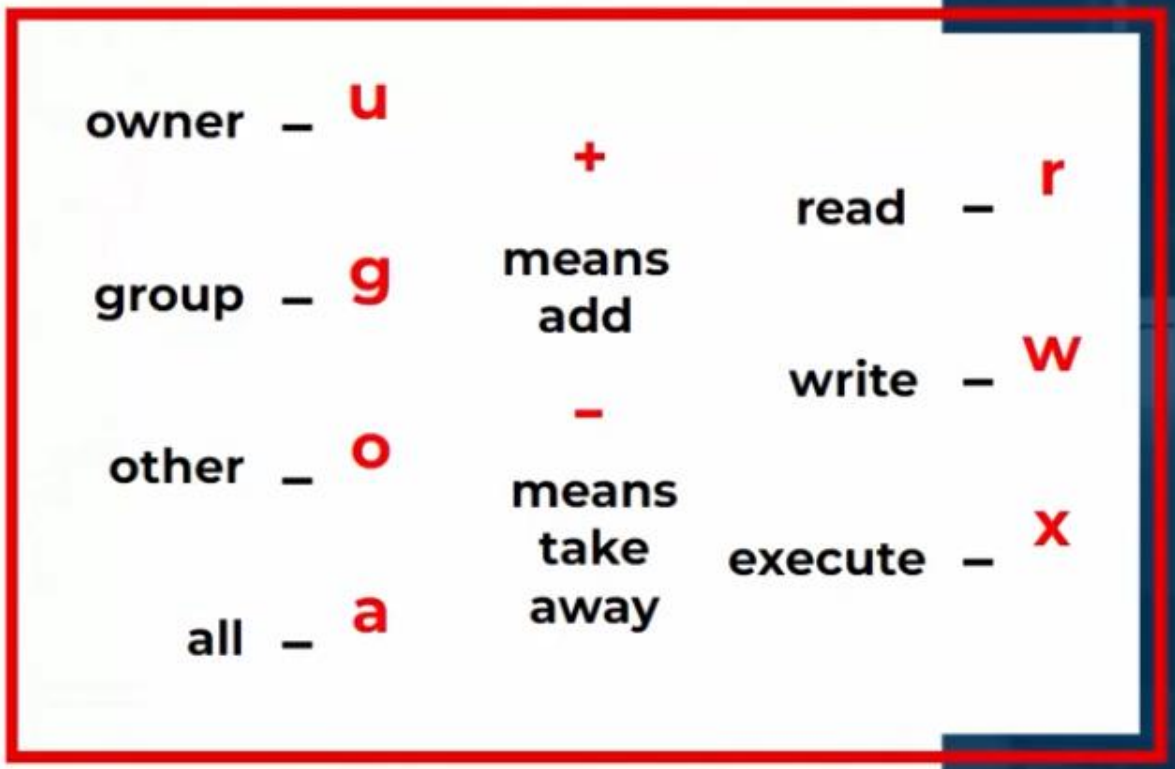
- Each file has an owner
- Each file is assigned to a group
- Permissions can only be changed by the owner and root

Changing file and directory permissions

- The command used to change permissions from the command line is **chmod**

- **Symbolic method**

u, g, o, a (for user, group, other, all)
+, -, = (for add, remove, set exactly)
r, w, x (for read, write, executable)



Symbolic method examples

```
[root@lab01 ~]# chmod g+w file1
[root@lab01 ~]# chmod o+w file1
[root@lab01 ~]# chmod u-w file1
[root@lab01 ~]# chmod u+w,g+wx,o+r file1
[root@lab01 ~]# chmod go-rw file1
[root@lab01 ~]# chmod u=rw,g=r,o=r file1
[root@lab01 ~]# chmod a+x file1
[root@lab01 ~]# chmod a=rw file1
[root@lab01 ~]# chmod u= file1
[root@lab01 ~]# chmod +rw file1
[root@lab01 ~]# chmod =rw file1
[root@lab01 ~]# chmod -R g+rwx dir1
```

(resets all old permissions)

or chmod ugo+x file1

or chmod ugo=rw file1

(reomves all permissions from owner)

or chmod a=rw file1

Numeric method

```
[root@lab01 ~]# chmod 754 file1
[root@lab01 ~]# chmod 400 file1
[root@lab01 ~]# chmod -R 755 dir1
```

(--rwx,r-x,r)

(---,---,--r)

Octal	Decimal	Permission	Represntation
000	0(0+0+0)	no permission	---
001	1(0+0+1)	Execute	--x
010	2(0+2+0)	Write	-w-
011	3(0+2+1)	Write + Execute	-wx
100	4(4+0+0)	Read	r--
101	5(4+0+1)	Read + Execute	r-x
110	6(4+2+0)	Read + Write	rw-
111	7(4+2+1)	Read + Write + Execute	rwx

Numeric method

```
[root@lab01 ~]# chmod 754 file1
[root@lab01 ~]# chmod 400 file1
[root@lab01 ~]# chmod -R 755 dir1
```

(--rwx,r-x,r)

(---,---,--r)

Octal	Decimal	Permission	Represntation
000	0(0+0+0)	no permission	---
001	1(0+0+1)	Execute	--x
010	2(0+2+0)	Write	-w-
011	3(0+2+1)	Write + Execute	-wx
100	4(4+0+0)	Read	r--
101	5(4+0+1)	Read + Execute	r-x
110	6(4+2+0)	Read + Write	rw-
111	7(4+2+1)	Read + Write + Execute	rwx

Required permissions

Command	Source directory	Source file	Target Directory
cd	x	N/A	N/A
ls	x, r	N/A	N/A
mkdir, rmdir	x, w	N/A	N/A
cat, less	x	r	N/A
cp	x	r	x, w
cp -r	x, r	r	x, w
mv	x, w	none !!!	x, w
vi	x, r	r, w	N/A
rm	x, w	none !!!	N/A

MOOCA

Changing file and directory user or group ownership

- Only root can change the ownership of a file.
- Root or the file's owner can change group ownership.

```
[root@lab01 ~]# chown karim file1
[root@lab01 ~]# chown karim dir1
[root@lab01 ~]# chown -R karim dir1
[root@lab01 ~]# chown :sales file1
[root@lab01 ~]# chgrp sales file1
[root@lab01 ~]# chown karim:sales file1
[root@lab01 ~]# chown -R karim:sales dir1
```

(change the group ownership)
(change the group ownership)
(change the owner and group)

Special permissions

- Special permissions provide additional access-related features over and above what the basic permission types allow. This section details the impact of special permissions, summarized in the table below.

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.
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 access to the directory can only remove files that they own; they cannot remove or force saves to files owned by other users.

Setuid

- **setuid** permission on an executable file means that commands run as the user owning the file, not as the user that ran the command.
- One example is the **passwd** command:

```
[user@host ~]$ ls -l /usr/bin/passwd
-rwsr-xr-x. 1 root root 32648 Aug 10  2021 /usr/bin/passwd
```

- In a long listing, you can identify 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 is replaced by an uppercase **S**.

Setgid

- The special permission **setgid** on a directory means that files created in the directory inherit their group ownership from the directory, rather than inheriting it from the creating user.
- An example of this is the **/run/log/journal** directory:

```
[user@host ~]$ ls -ld /run/log/journal
drwxr-sr-x+ 3 root systemd-journal 60 Sep 24 16:21 /run/log/journal
```

Sticky bit

- 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**:

```
[user@host ~]$ ls -ld /tmp
drwxrwxrwt. 16 root root 4096 Sep 28 03:37 /tmp
```

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:

```
[user@host ~]# chmod g+s directory
```

- Set the setgid bit and add read/write/execute permissions for user and group, with no access for others, on directory:

```
[user@host ~]# chmod 2770 directory
```

Default file permissions

- When you create a new file or directory, it is assigned initial permissions. There are two things that affect these initial permissions.
- The first is whether you are creating a **regular file** or a **directory**. The second is the current **umask**.
- The **umask** command without arguments will display the current value of the shell's umask:

```
[root@client ~]# umask
0022
[karim@client ~]$ umask
0002
```

Default file permissions

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

```
[karim@client ~]$ echo "umask 007" >> ~/.bashrc
```

(Log out and log in again)

SUMMARY

In this chapter, you learned:

- Files have three categories to which permissions apply. A file is owned by a user, a single group, and other users. The most specific permission applies. User permissions override group permissions and group permissions override other permissions.
- The **ls** command with the **-l** option expands the file listing to include both the file permissions and ownership.
- The **chmod** command changes file permissions from the command line. There are two methods to represent permissions, symbolic (letters) and numeric (digits).
- The **chown** command changes file ownership. The **-R** option recursively changes the ownership of a directory tree.
- The **umask** command without arguments displays the current umask value of the shell. Every process on the system has a umask. The default umask values for Bash are defined in the **/etc/profile** and **/etc/bashrc** files.

RH124

Monitoring and Managing Linux Processes



Red Hat | Enterprise Linux

MOOCA

Monitoring and Managing Linux Processes

Goal

- Evaluate and control processes running on a Red Hat Enterprise Linux system.

Objectives

- Get information about programs running on the system so that you can determine status, resource use, and ownership, so you can control them.
- Control and terminate processes using signals.
- Change the priority of running processes.

Monitoring and Managing Linux processes

- A process is a running instance of a launched, executable program.
- Every new process is assigned a unique process ID (PID) for tracking and security.
- The PID and the parent's process ID (PPID) are elements of the new process environment.
- Any process may create a child process.

Listing processes

- The **ps** command is used for listing current processes. It can provide detailed process information, including:
  - User identification (UID), which determines process privileges
  - Unique process identification (PID)
  - 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
  - The current process state

- The most common set of options, **aux**, displays all processes including processes without a controlling terminal. A long listing (options **lax**) provides more technical detail. The similar UNIX syntax uses the options -**ef** to display all processes

```
[user@host ~]$ 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
root         2  0.0  0.0      0     0 ?        S    17:45   0:00 [kthreadd]
root         3  0.0  0.0      0     0 ?        S    17:45   0:00 [ksoftirqd/0]
root         5  0.0  0.0      0     0 ?        S<   17:45   0:00 [kworker/0:0H]
root         7  0.0  0.0      0     0 ?        S    17:45   0:00 [migration/0]
...output omitted...
[user@host ~]$ ps lax
F   UID   PID  PPID PRI  NI    VSZ   RSS WCHAN  STAT TTY      TIME COMMAND
4     0     1     0  20   0  51648  7504 ep_pol Ss   ?       0:03 /usr/lib/
systemd/
1     0     2     0  20   0      0     0 kthrea S    ?       0:00 [kthreadd]
1     0     3     2  20   0      0     0 smpboo S    ?       0:00 [ksoftirqd/0]
1     0     5     2   0 -20      0     0 worker S<   ?       0:00 [kworker/0:0H]
1     0     7     2 -100  -      0     0 smpboo S    ?       0:00 [migration/0]
...output omitted...
[user@host ~]$ ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 17:45 ?        00:00:03 /usr/lib/systemd/systemd --
switched-ro
root         2     0  0 17:45 ?        00:00:00 [kthreadd]
root         3     2  0 17:45 ?        00:00:00 [ksoftirqd/0]
root         5     2  0 17:45 ?        00:00:00 [kworker/0:0H]
root         7     2  0 17:45 ?        00:00:00 [migration/0]
...output omitted...
```

- The output of **ps** displays once. Use **top** for a process display that dynamically updates.
- **ps** can display in tree format using **pstree** so you can view relationships between parent and child processes.
- PID #1 is for **systemd** (the first process in the system) ---> like initd in RHEL6 :)

```
[root@lab01 ~]# pstree
OR
[root@lab01 ~]# ps fax
[root@lab01 ~]# pstree -p
[root@ lab01 ~]# pstree -p karim
[root@lab01 ~]# pgrep -u user -l
```

(process status tree)

(process status tree)

(Display PID of each process)

(Show process name of a specific user)

Controlling jobs

- Job control is a feature of the shell which allows a single shell instance to run and manage multiple commands.
- 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 cannot 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 any number of independent **background** processes. A job is part of exactly one session: the one belonging to its controlling terminal.

- 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**) in the terminal.

```
sleep 10000
^Z
[1]+  Stopped                 sleep 10000
[user@host ~]$
```

- To start the suspended process running in the background, use the **bg** command with the same job ID.

```
[user@host ~]$ bg %1
[1]+ sleep 10000 &
```

- The **ps j** command displays information relating to jobs.

Killing processes

- The **kill** command sends a signal to a process by PID number. Despite its name, the **kill** command can be used for sending any signal, not just those for terminating programs. You can use the **kill -l** command to list the names and numbers of all available signals

```
[root@RHEL9 ~]# 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
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO       30) SIGPWR
31) SIGSYS      34) SIGRTMIN    35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```

Killing processes

```
[root@master ~]# kill -l
[root@master ~]# pidof vim
4123
[root@master ~]# kill 4123
[root@master ~]# pidof vim
7073
[root@master ~]# kill -9 7073
[root@master ~]# kill -SIGKILL 7073


[root@master ~]# pkill vim
[root@master ~]# killall vim
[user@ master ~]$pkill -U user
```

(List all signals)

(Default is SIGTERM 15)

(Default is SIGTERM 15)

- kill -15 --> to terminate the process cleanly
- kill -9 --> aggressive termination (used in emergency termination but not recommended !)
- kill -1 --> SIGHUB --> to reload an re-read the configurations (configuration reload without termination)

Logging users out administratively

- You may need to log other users off for any of a variety of reasons. To name a few of the many possibilities:
  - the user committed a security violation.
  - the user may have an unresponsive system.
  - the user may have overused resources.
  - the user has improper access to materials.
- . In these cases, you may need to administratively terminate their session using signals.
- First identify the PID numbers to be killed using **pgrep**, which operates much like **pkill**, including using the same options, except that **pgrep** lists processes rather than killing them.

```
[root@host ~]# pgrep -l -u bob
6964 bash
6998 sleep
6999 sleep
7000 sleep
[root@host ~]# pkill -SIGKILL -u bob
[root@host ~]# pgrep -l -u bob
[root@host ~]#
```

Real-time process monitoring

- The top program is a dynamic view of the system's processes.

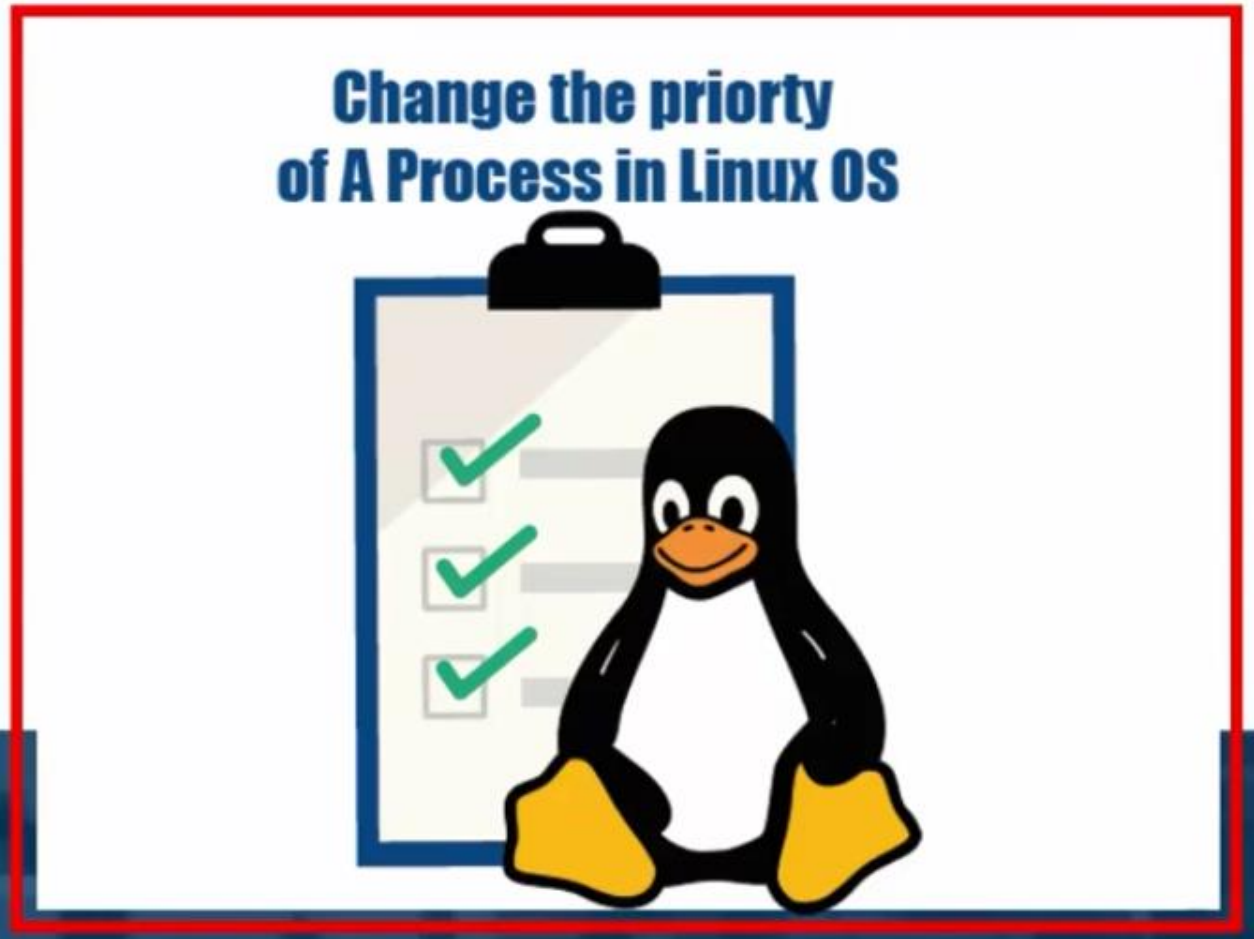
```
[root@master ~]# top

[root@master ~]# top -n 2

[root@master ~]# top -d 2
```

Specifies the maximum number of iterations, or frames, top should produce before ending.

Number of second between each update (the delay between screen updates) !!

type 1	to show all cpu cores
type s	to change the default refresh rate which is 3 seconds
type h	for help
type k	to kill a process
type r	to renice a process
type M	to change the display to sort by the amount of memory
type P	to change the display to sort by the CPU utilization
type n	to change the number of processes shown
type w	to save current display configuration
type q	to quit

Managing priority of Linux processes



Linux process scheduling and multitasking:

- The way Linux (and other operating systems) can run more processes is by employing a technique called time-slicing.
- The part of the Linux kernel that performs this switching is called the process scheduler.
- There are exactly 40 different levels of niceness a process can have (-20 to 19).
- By default, processes will inherit their nice level from their parent, which is usually 0.
- Higher nice levels indicate less priority, while lower nice levels indicate a higher priority.
- Only root is allowed to set negative nice levels and lower the nice level on existing processes.
- Un privileged users are only allowed to set positive nice levels, and they are only allowed to raise the nice level on their existing process but cannot lower them.

How to Set Linux Process Priority Using nice and renice Commands

- The nice command is used to start a process with a user defined priority.

```
[root@lab01 ~]# nice vim text &
[1] 9182
[root@lab01 ~]# nice -n vim text &
```

(Default is 10)

- The renice command is used to change the priority of a currently running process.

```
[root@lab01 ~]# renice 19 9182
[root@lab01 ~]#top
```

(19 is the new value)

(press r to renice a process)

SUMMARY

In this chapter, you learned:

- A process is a running instance of an executable program. The **ps** command is used to list processes.
- Each terminal is its own session and can have foreground process and independent background processes. The **jobs** command displays processes within a terminal session.
- A signal is a software interrupt that reports events to an executing program. The **kill**, **pkill**, and **killall** commands use signals to control processes.

RH124

Controlling Services and Daemons



Red Hat | Enterprise Linux

mooca



Controlling Services and Daemons

Goal

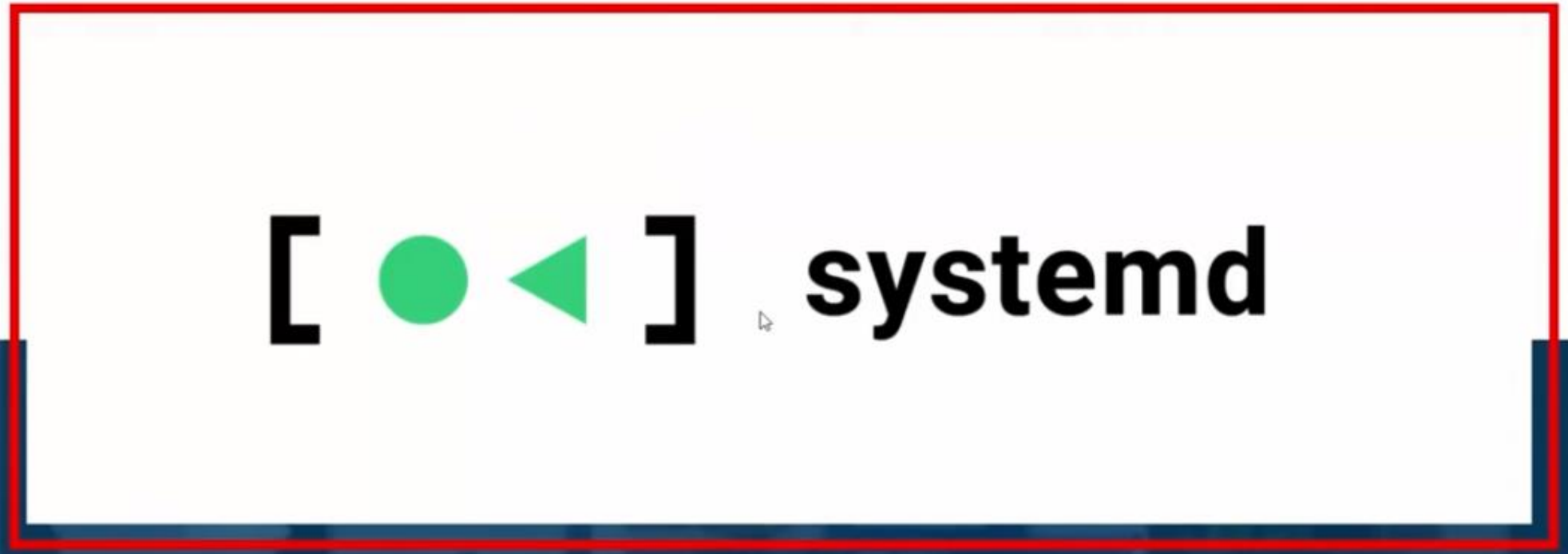
- Control and monitor network services and system daemons using Systemd.

Objectives

- List system daemons and network services started by the systemd service and socket units.
- Control system daemons and network services, using **systemctl**.

Introduction to systemd

- In Red Hat Enterprise Linux, the first process that starts (PID 1) is **systemd**.
- The **systemd** daemon manages startup for Linux, including service startup and service management in general. It activates system resources, server daemons, and other processes both at boot time and on a running system.



Listing service units

- You use the **systemctl** command to explore the current state of the system. For example, the following command lists all currently loaded service units

```
[root@host ~]# systemctl list-units --type=service
```

(List all active services)

- By default, the **systemctl list-units --type=service** command lists only the service units with **active** activation states.

Viewing service states

- View the status of a specific unit with **systemctl status name.type**. If the unit type is not provided, **systemctl** will show the status of a service unit, if one exists.

```
[root@host ~]# systemctl status sshd.service
```

```
● sshd.service - OpenSSH server daemon
   Loaded: loaded (/usr/lib/systemd/system/sshd.service; enabled; vendor preset: enabled)
   Active: active (running) since Mon 2021-12-13 16:26:33 EET; 8h ago
     Docs: man:sshd(8)
           man:sshd_config(5)
 Main PID: 43828 (sshd)
    Tasks: 1 (limit: 5677)
   Memory: 8.1M
```

Service unit information

FIELD	DESCRIPTION
Loaded	Whether the service unit is loaded into memory.
Active	Whether the service unit is running and if so, how long it has been running.
Main PID	The main process ID of the service, including the command name.
Status	Additional information about the service.

Service state in the output of systemctl

KEYWORD	DESCRIPTION
loaded	Unit configuration file has been processed.
active (running)	Running with one or more continuing processes.
active (exited)	Successfully completed a one-time configuration.
active (waiting)	Running but waiting for an event.
inactive	Not running.
enabled	Is started at boot time.
disabled	Is not set to be started at boot time.
static	Cannot be enabled, but may be started by an enabled unit automatically.

Verifying the status of a services

- The **systemctl** command provides methods for verifying the specific states of a service. For example, use the following command to verify that the a service unit is currently active (running):

```
[root@host ~]# systemctl is-active sshd.service
```

active or inactive

```
[root@host ~]# systemctl is-enabled sshd.service
```

enable or disable

```
[root@host ~]# systemctl is-failed sshd.service
```

active or failed

- To list all the failed units, run the **systemctl --failed --type=service** command

Controlling system services

- To start a service:

```
[root@host ~]# systemctl start sshd.service
```

- To restart a service:

```
[root@host ~]# systemctl restart sshd.service
```

Controlling system services

- To stop a currently running service:

```
[root@host ~]# systemctl stop sshd.service
```

- To reload a service:

(Re-read the configuration file)

```
[root@host ~]# systemctl reload sshd.service
```

Masking and unmasking services

```
[root@host ~]# systemctl mask sendmail.service

[root@host ~]# systemctl list-unit-files --type=service | grep sendmail
sendmail.service                          masked

[root@host ~]# systemctl start sendmail.service
Failed to start sendmail.service: Unit sendmail.service is masked.

[root@host ~]# systemctl unmask sendmail
```

Enabling services to start or stop at boot

- To start a service at boot, use the systemctl enable command.

```
[root@root ~]# systemctl enable sshd.service
```

- To disable the service from starting automatically

```
[root@host ~]# systemctl disable sshd.service
```

Summary of systemctl commands

TASK	COMMAND
View detailed information about a unit state.	`systemctl status UNIT`
Stop a service on a running system.	`systemctl stop UNIT`
Start a service on a running system.	`systemctl start UNIT`
Restart a service on a running system.	`systemctl restart UNIT`
Reload the configuration file of a running service.	`systemctl reload UNIT`
Completely disable a service from being started, both manually and at boot.	`systemctl mask UNIT`
Make a masked service available.	`systemctl unmask UNIT`
Configure a service to start at boot time.	`systemctl enable UNIT`
Disable a service from starting at boot time.	`systemctl disable UNIT`
List units required and wanted by the specified unit.	`systemctl list-dependencies UNIT`

SUMMARY

In this chapter, you learned:

- **systemd** provides a method for activating system resources, server daemons, and other processes, both at boot time and on a running system.
- Use the **systemctl** to start, stop, reload, enable, and disable services.
- Use the **systemctl status** command to determine the status of system daemons and network services started by systemd.
- The **systemctl list-dependencies** command lists all service units upon which a specific service unit depends.
- systemd can mask a service unit so that it does not run even to satisfy dependencies.



Configuring and Securing SSH

Goal

- Configure secure command-line service on remote systems, using OpenSSH.

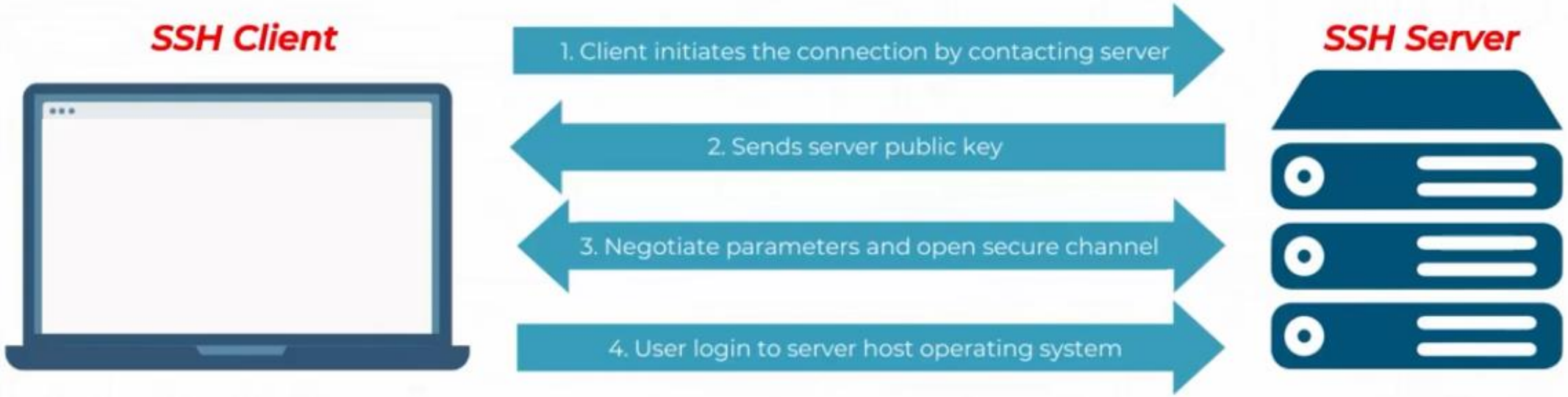
Objectives

- Log in to a remote system and run commands using ssh.
- Configure key-based authentication for a user account to log in to remote systems securely without a password.
- Restrict direct logins as root and disable password-based authentication for the OpenSSH service.

Mooca

WHAT IS OPENSSH?

- OpenSSH implements the Secure Shell or SSH protocol in the Red Hat Enterprise Linux systems.
- The SSH protocol enables systems to communicate in an encrypted and secure fashion over an insecure network.
- The first time a user uses ssh to connect to a particular server, the ssh command stores the server's public key in the user's *~/.ssh/known_hosts* file.



Secure shell examples

- The below *ssh* command would log you in on the remote server *remotehost* using the username *user02*, you are prompted by the remote system to authenticate with that user's password.

```
[user01@host ~]$ ssh user02@remotehost
user02@remotehost's password: shadowman
...output omitted...
[user02@remotehost ~]$
```

- This *ssh* command would run the *hostname* command on the *serverb* remote system as the *student* user without accessing the remote interactive shell.

```
[student@servera ~]$ ssh student@serverb hostname
student@serverb's password: student
serverb.lab.example.com
```

Secure shell examples

- The below *ssh* command would log you in on the remote server *remotehost* using the username *user02*, you are prompted by the remote system to authenticate with that user's password.

```
[user01@host ~]$ ssh user02@remotehost
user02@remotehost's password: shadowman
...output omitted...
[user02@remotehost ~]$
```

- This *ssh* command would run the *hostname* command on the *serverb* remote system as the *student* user without accessing the remote interactive shell.

```
[student@servera ~]$ ssh student@serverb hostname
student@serverb's password: student
serverb.lab.example.com
```

MOOCA

SSH Key-Based Authentication

- You can configure an SSH server to allow you to authenticate without a password by using keybased authentication. This is based on a private-public key scheme.

Generating SSH Keys

- To create a private key and matching public key for authentication, use the *ssh-keygen* command.
- By default, your private and public keys are saved in your *~/.ssh/id_rsa* and *~/.ssh/id_rsa.pub* files, respectively.

```
[user@host ~]$ ssh-keygen
```

- Once the SSH keys have been generated, they are stored by default in the .ssh/ directory of the user's home directory.
- The permission modes must be 600 on the private key and 644 on the public key.

Sharing the Public Key

- Before key-based authentication can be used, the public key needs to be copied to the destination system.
- The *ssh-copy-id* command copies the public key of the SSH keypair to the destination system.
- If you omit the path to the public key file while running *ssh-copy-id*, it uses the default */home/user/.ssh/id_rsa.pub* file.

```
[user@host ~]$ ssh-copy-id -i .ssh/ id_rsa.pub user@remotehost
```

Customizing open SSH service configuration

- OpenSSH service is provided by a daemon called sshd. Its main configuration file is */etc/ssh/sshd_config.*

Prohibit the superuser from logging in using SSH

- It is a good practice to prohibit direct login to the root user account from remote systems. Some of the risks of allowing direct login as root include:
- The OpenSSH server uses the *PermitRootLogin* configuration setting in the */etc/ssh/sshd_config* configuration file to allow or prohibit users logging in to the system as root.
- With the *PermitRootLogin* parameter to yes, people are permitted to log in as root.
- The SSH server (sshd) must be reloaded for any changes to take effect.

```
[root@host ~]# systemctl reload sshd
```

Prohibiting password-based authentication for SSH

- The OpenSSH server uses the *PasswordAuthentication* parameter in the */etc/ssh/sshd_config* configuration file to control whether users can use password-based authentication to log in to the system.
- The default value of *yes* for the PasswordAuthentication causes the SSH server to allow users to use password-based authentication while logging in.
- The value of *no* for PasswordAuthentication prevents users from using password-based authentication.
- Keep in mind that whenever you change the /etc/ssh/sshd_config file, you must reload the sshd service for changes to take effect.

SUMMARY

In this chapter, you learned:

- OpenSSH implements the Secure Shell or SSH protocol in the Red Hat Enterprise Linux systems.
- The ***ssh*** command allows users to access remote systems securely using the SSH protocol.
- A client system stores remote servers' identities in ***~/.ssh/known_hosts*** and ***/etc/ssh/ssh_known_hosts***.
- SSH supports both ***password-based*** and ***key-based*** authentication.
- The ***ssh-keygen*** command generates an SSH key pair for authentication. The ***ssh-copy-id*** command exports the public key to remote systems.
- The ***sshd*** service implements the SSH protocol on Red Hat Enterprise Linux systems.
- It is a recommended practice to configure sshd to disable remote logins as root and to require public key authentication rather than password-based authentication.

RH124

Analyzing and Storing Logs



Red Hat | Enterprise Linux

MOOCA



Analyzing and Storing Logs

Goal

- Locate and accurately interpret logs of system events for troubleshooting purposes.

Objectives

- Describe the basic logging architecture used by Red Hat Enterprise Linux to record events.
- Understand logrotate utility.
- View and manage logs using journalctl command.
- Configure the time zone and adjust the system time.

Mooca

System Logging

- Processes and the operating system kernel record a log of events that happen. These logs are used to audit the system and troubleshoot problems.
- Many systems record logs of events in text files which are kept in the /var/log directory. These logs can be inspected using normal text utilities such as **less** and **tail**.
- The **systemd journald** and **rsyslog** services handle the syslog messages in Red Hat Enterprise Linux 9.

Syslog messages are handled by two service:

- **systemd-journald**: The Journal is a component of systemd that is responsible for viewing and management of log files
- **rsyslog**: The rsyslog service sorts and writes syslog messages to the log files that do persist across reboots in /var/log.

Selected System Log Files

LOG FILE	TYPE OF MESSAGES STORED
`/var/log/messages`	Most syslog messages are logged here. Exceptions include messages related to authentication and email processing, scheduled job execution, and those which are purely debugging-related.
`/var/log/secure`	Syslog messages related to security and authentication events.
`/var/log/maillog`	Syslog messages related to the mail server.
`/var/log/cron`	Syslog messages related to scheduled job execution.
`/var/log/boot.log`	Non-syslog console messages related to system startup.

Overview of Syslog Priorities

- The following table lists the standard eight syslog priorities from highest to lowest.

CODE	PRIORITY	SEVERITY
0	emerg	System is unusable
1	alert	Action must be taken immediately
2	crit	Critical condition
3	err	Non-critical error condition
4	warning	Warning condition
5	notice	Normal but significant event
6	info	Informational event
7	debug	Debugging-level message

Reviewing syslog files

- The rsyslog service uses the facility and priority of log messages to determine how to handle them.

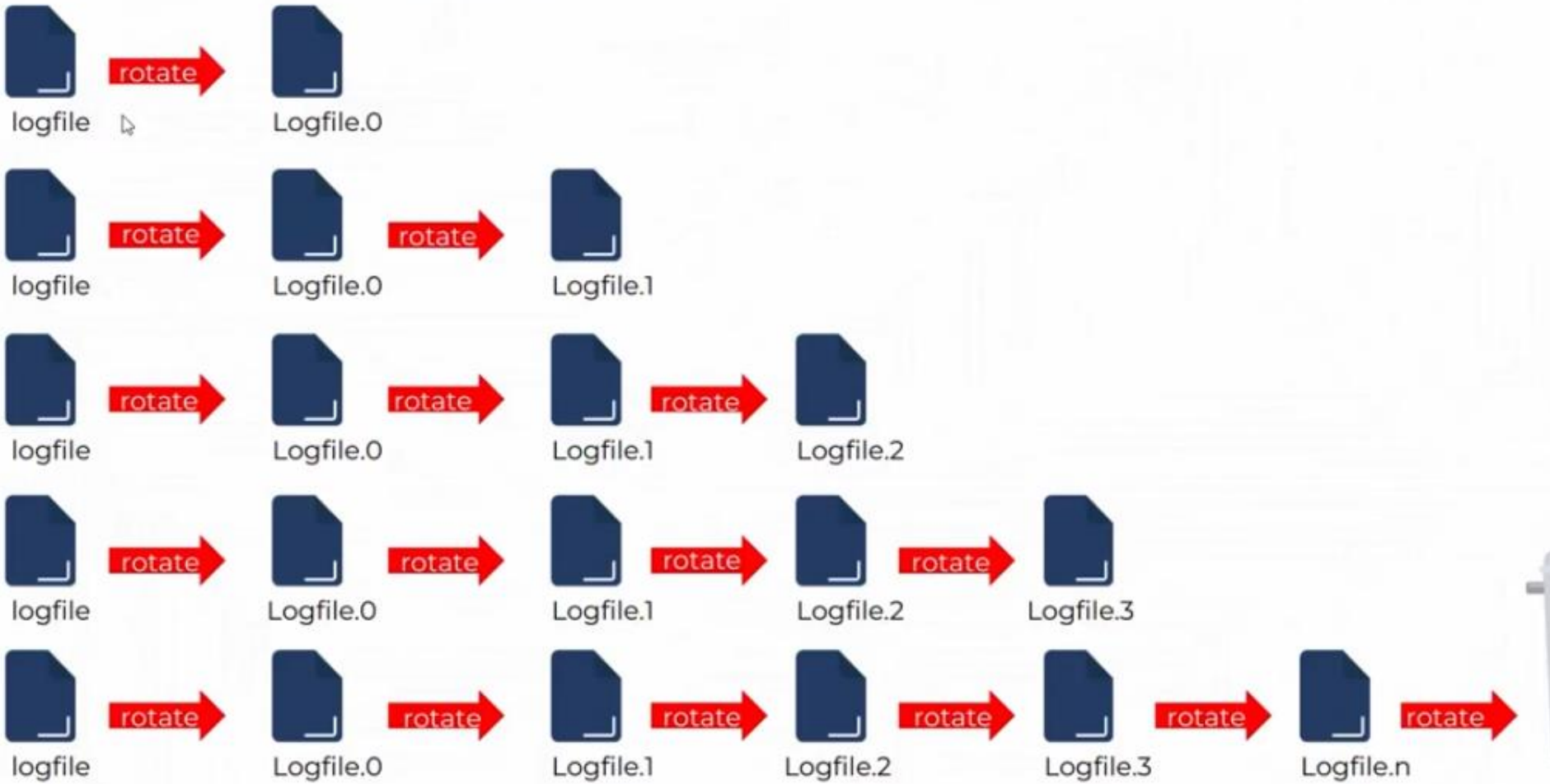
  This is configured by rules in the **/etc/rsyslog.conf** file.

Sample rules of rsyslog

```
#### RULES ####

# Log all kernel messages to the console.
# Logging much else clutters up the screen.
#kern.*                                                 /dev/console

# Log anything (except mail) of level info or higher.
# Don't log private authentication messages!
*.info;mail.none;authpriv.none;cron.none                /var/log/messages

# The authpriv file has restricted access.
authpriv.*                                              /var/log/secure

# Log all the mail messages in one place.
mail.*                                                  -/var/log/maillog


# Log cron stuff
cron.*                                                  /var/log/cron
```

Log File Rotation

- The **logrotate** tool rotates log files to keep them from taking up too much space in the file system containing the **/var/log** directory.



Monitoring Logs

- Monitoring one or more log files for events is helpful to reproduce problems and issues.
- **The tail -f /path/to/file** command outputs the last 10 lines of the file specified and continues to output new lines in the file as they get written.

```
[root@host ~]# tail -f /var/log/secure
```

Sending Syslog Messages Manually

- The **logger** command can send messages to the rsyslog service. By default, it sends the message to the user facility with the notice priority (**user.notice**) unless specified otherwise with the **-p** option.

- To send a message to the rsyslog service that gets recorded in the /var/log/boot.log log file and messages file

```
[root@host ~]# logger -p local7.notice "Log entry created on host"
```

Reviewing System Journal Entries

- The **journalctl** command highlights important log messages: messages at **notice** or **warning** priority are in bold text while messages at the **error** priority or higher are in red text.

- By default, **journalctl -n** shows the last 10 log entries. You can adjust this with an optional argument that specifies how many log entries to display. For the last five log entries, run the following **journalctl** command:

```
[root@host ~]# journalctl -n 5
```

Reviewing System Journal Entries

- Similar to the **tail -f** command, the **journalctl -f** command outputs the last 10 lines of the system journal and continues to output new journal entries as they get written to the journal. To exit the **journalctl -f** process, use the **Ctrl+C** key combination.

```
[root@host ~]# journalctl -f
```

- Run the following journalctl command to list journal entries at the err priority or higher:

```
[root@host ~]# journalctl -p err
```

Overview of Syslog Priorities

- The following table lists the standard eight syslog priorities from highest to lowest.

CODE	PRIORITY	SEVERITY
0	emerg	System is unusable
1	alert	Action must be taken immediately
2	crit	Critical condition
3	err	Non-critical error condition
4	warning	Warning condition
5	notice	Normal but significant event
6	info	Informational event
7	debug	Debugging-level message

MOOCA

Maintaining Accurate Time

- The **timedatectl** command shows an overview of the current time-related system settings, including current time, time zone, and NTP synchronization settings of the system.

```
[user@host ~]$ timedatectl
```

- A database of time zones is available and can be listed with the **timedatectl list-timezones** command.

- The superuser can change the system setting to update the current time zone using the **timedatectl set-timezone** command. The following **timedatectl** command updates the current time zone to **Africa/Cairo**

```
[root@host ~]# timedatectl set-timezone Africa/Cairo
```

Maintaining Accurate Time

- Use the **tzselect** command to determine the appropriate time zone
- Use the **timedatectl set-time** command to change the system's current time. The time is specified in the "YYYY-MM-DD hh:mm:ss" format, where either date or time can be omitted. The following **timedatectl** command changes the time to 09:00:00.

```
[root@host ~]# timedatectl set-time 9:00:00
```

SUMMARY

In this chapter, you learned:

- The **systemd-journald** and **rsyslog** services capture and write log messages to the appropriate files.
- The **/var/log** directory contains log files.
- Periodic rotation of log files prevent them from filling up the file system space.
- The systemd journals are temporary and do not persist across reboot.
- The time zone of the server can be updated based on its location.

RH124

Managing Networking



Red Hat | Enterprise Linux

MOOCA

Managing Networking

Goal

- Configure network interfaces and settings on Red Hat Enterprise Linux servers.

Objectives

- Configure and modify network Settings from the CLI.
- Manage network settings and devices using **nmcli**.
- Troubleshoot connectivity issues using CLI commands.
- Change a server's host name and the DNS configurations from the CLI.

Gathering network interface information

- Identifying Network Interfaces

The **ip link** command will list all network interfaces available on your system

```
[user@host ~]$ ip link show
```

- Displaying IP Addresses

```
[user@host ~]$ ip addr show ens160
```

- Displaying Performance Statistics

```
[user@host ~]$ ip -s link show ens160
```

Checking connectivity between hosts

- The **ping** command is used to test connectivity. The command continues to run until Ctrl+C is pressed unless options are given to limit the number of packets sent.

```
[user@host ~]$ ping -c3 192.168.1.9
```

- Displaying the Routing Table

  Use the **ip** command with the **route** option to show routing information.

```
[user@host ~]$ ip route
```

Checking connectivity between hosts

- Tracing Routes Taken by Traffic

To trace the path that network traffic takes to reach a remote host through multiple routers, use either **traceroute** or **tracepath**. This can identify whether an issue is with one of your routers or an intermediate one.

```
[user@host ~]$ tracepath access.redhat.com
```

Troubleshooting ports and services

- TCP services use **sockets** as end points for communication and are made up of an **IP address**, **protocol**, and **port number**.
- Services typically listen on standard ports while clients use a random available port. Well-known names for standard ports are listed in the **/etc/services** file.
- The **ss** command is used to display socket statistics. The ss command is meant to replace the older tool **netstat**, part of the net-tools package, which may be more familiar to some system administrators but which is not always installed

```
[user@host ~]$ ss -ta
```

OPTION	DESCRIPTION
-n	Show numbers instead of names for interfaces and ports.
-t	Show TCP sockets.
-u	Show UDP sockets.
-l	Show only listening sockets.
-a	Show all (listening and established) sockets.
-p	Show the process using the sockets.

MOOCA

Troubleshooting ports and services

- TCP services use **sockets** as end points for communication and are made up of an **IP address**, **protocol**, and **port number**.
- Services typically listen on standard ports while clients use a random available port. Well-known names for standard ports are listed in the **/etc/services** file.
- The **ss** command is used to display socket statistics. The ss command is meant to replace the older tool **netstat**, part of the net-tools package, which may be more familiar to some system administrators but which is not always installed

```
[user@host ~]$ ss -ta
```

OPTION	DESCRIPTION
-n	Show numbers instead of names for interfaces and ports.
-t	Show TCP sockets.
-u	Show UDP sockets.
-l	Show only listening sockets.
-a	Show all (listening and established) sockets.
-p	Show the process using the sockets.

Configure network from the command line

- **RHEL8**
  - The **nmcli** utility is used to create and edit connection files from the command line and save configuration files in the **/etc/sysconfig/network-scripts** directory

```
[root@RHEL9 ~]# ls /etc/sysconfig/network-scripts
lfcfg-ens160
```

Configure network from the command line

- **RHEL9** (Say goodbye to ifcfg-files, and hello to keyfiles)

  - The **nmcli** command can be used to modify the network configuration. Network configuration will be written to files in the **/etc/NetworkManager/system-connections/** directory
  - The old network-scripts package has been removed (it was deprecated in the RHEL 8) which means you'll not find anything in the **/etc/sysconfig/network-scripts** directory

```
[root@RHEL9 ~]# ls /etc/sysconfig/network-scripts
[root@RHEL9 ~]# ls /etc/NetworkManager/system-connections/
ens160.nmconnection
```

MOOCA

Viewing network information

- The **nmcli dev status** command displays the status of all network devices.

```
[user@host ~]$ nmcli dev status
```

- The **nmcli con show** command displays a list of all connections. To list only the active connections, add the --active option.

```
[user@host ~]$ nmcli con show
```

Adding a network connection

- The **nmcli con add** command is used to add new network connections.

The next example creates an **static-ens224** connection for the **ens224** device with a static IPv4 address, using the IPv4 address and network prefix **192.168.0.5/24** and default gateway **192.168.0.254**, but still **auto connects** at startup and saves its configuration into the same file.

```
[root@host ~]#nmcli connection add con-name static-ens224 type
ethernet ifname ens224 ipv4.addresses 192.168.1.55/24 gw4 192.168.1.1
connection.autoconnect yes ipv4.method manual
```

Modifying network connection setting

- The **nmcli con mod name** command is used to change the settings for a connection.

To set the IPv4 address to **192.0.2.2/24** and default gateway to **192.0.2.254** for the connection **static-ens3**:

```
[root@host ~]# nmcli con mod static-ens3 ipv4.addresses "192.0.2.2/24 192.0.2.254"
```

Deleting a network connection

- The **nmcli con del name** command deletes the connection named name from the system, disconnecting it from the device and removing the file **/etc/sysconfig/network-scripts/ifcfg-name** in case of **RHEL8** and removing the file **/etc/NetworkManager/system-connections/name.nmconnection** in case of **RHEL9.**

```
[root@host ~]# nmcli con del static-ens3
```

Who can modify network settings

- The root user can make any necessary network configuration changes with **nmcli**.
- Regular users that log in using ssh do not have access to change network permissions without becoming root.
- You can use the **nmcli gen permissions** command to see what your current permissions are.

```
[root@client ~]# nmcli gen permissions
PERMISSION                                                VALUE
org.freedesktop.NetworkManager.checkpoint-rollback        yes
org.freedesktop.NetworkManager.enable-disable-connectivity-check  yes
```

Editing network configuration files

- By default, changes made with **nmcli con mod** name are automatically saved to:
  - /etc/sysconfig/network-scripts/ifcfg-name (**RHEL8**)
  - /etc/NetworkManager/system-connections/name.nmconnection (**RHEL9**)
- The files can also be manually edited with a text editor. After doing so, run **nmcli con reload** so that NetworkManager reads the configuration changes.
- It is also possible to configure the network by directly editing the connection configuration files. Connection configuration files control the software interfaces for individual network devices. These files are usually named **/etc/sysconfig/network-scripts/ifcfg-name** or **/etc/NetworkManager/system-connections/name.nmconnection** .

Summary of commands

- The following table is a list of key **nmcli** commands discussed in this section.

COMMAND	PURPOSE
`nmcli dev status`	Show the NetworkManager status of all network interfaces.
`nmcli con show`	List all connections.
`nmcli con show` *name*	List the current settings for the connection *name*.
`nmcli con add con-name` *name*	Add a new connection named *name*.
`nmcli con mod` *name*	Modify the connection *name*.
`nmcli con reload`	Reload the configuration files (useful after they have been edited by hand).
`nmcli con up` *name*	Activate the connection *name*.
`nmcli dev dis` *dev*	Deactivate and disconnect the current connection on the network interface *dev*.
`nmcli con del` *name*	Delete the connection *name* and its configuration file.

Configuring host names and name resolution

- The **hostname** command displays or temporarily modifies the system's fully qualified host name.

```
[root@host ~]# hostname
host@example.com
```

- A static host name may be specified in the **/etc/hostname** file. The **hostnamectl** command is used to modify this file and may be used to view the status of the system's fully qualified host name.

```
[root@host ~]# hostnamectl set-hostname host@example.com
[root@host ~]# hostnamectl status
```

Configuring name resolution

- The DNS resolver is used to convert host names to IP addresses or the reverse. It determines where to look based on the configuration of the **/etc/nsswitch.conf** file. By default, the contents of the **/etc/hosts** file are checked first

```
[root@host ~]# cat /etc/hosts
```

- If an entry is not found in the **/etc/hosts** file, by default the stub resolver tries to look up the hostname by using a DNS nameserver. The **/etc/resolv.conf** file controls how this query is performed:

```
[root@host ~]# cat /etc/resolv.conf
```

Configuring name resolution

- NetworkManager updates the **/etc/resolv.conf** file using DNS settings in the connection configuration files. Use the nmcli to modify the connections.

```
[root@host ~]# nmcli con mod ens160 ipv4.dns 8.8.8.8
[root@host ~]# nmcli con down ens160
[root@host ~]# nmcli con up ens160
[root@host ~]# cat /etc/sysconfig/network-scripts/ifcfg-ens160
[root@host ~]# cat /etc/NetworkManager/system-connections/ens160.nmconnection
```

RHEL8

RHEL9

MOOCA

SUMMARY

In this chapter, you learned:

- **NetworkManager** is a daemon that monitors and manages network configuration.
- The **nmcli** command is a command-line tool for configuring network settings with NetworkManager.
- The system's static host name is stored in the **/etc/hostname** file. **The hostnamectl** command is used to modify or view the status of the system's host name and related settings.

RH124

Archiving and Transferring Files



Red Hat | Enterprise Linux





Archiving and Transferring Files

Goal

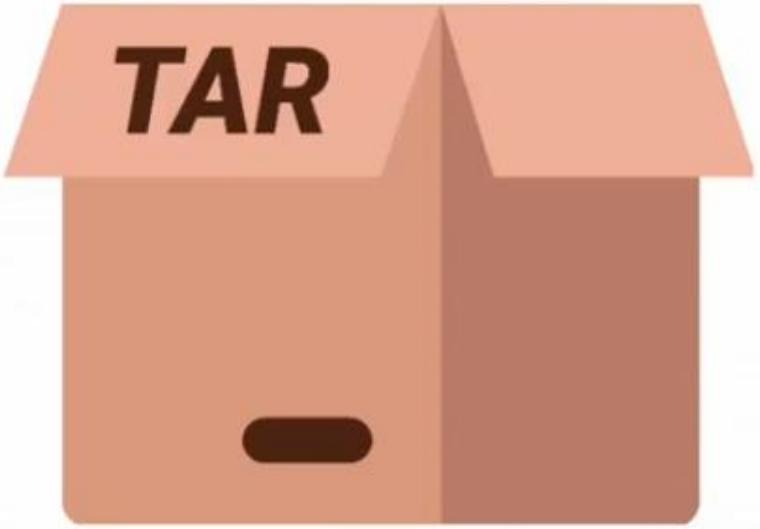
- Archive and copy files from one system to another.

Objectives

- Archive files and directories into a compressed file using tar, and extract the contents of an existing tar archive.
- Transfer files to or from a remote system securely using SSH.

Managing compressed TAR Archives

- Archiving and compressing files are useful when creating backups and transferring data across a network. One of the oldest and most common commands for creating and working with backup archives is the ***tar*** command.



Options of the tar Command

- The **tar** command expects one of the three following options:

OPTION	DESCRIPTION
`-c, --create`	Create a new archive.
`-x, --extract`	Extract from an existing archive.
`-t, --list`	List the table of contents of an archive.

Options of the tar Command

- Selected **tar** General Options

OPTION	DESCRIPTION
`-v, --verbose`	Verbose. Shows which files get archived or extracted.
`-f, --file=`	File name. This option must be followed by the file name of the archive to use or create.
`-p, --preserve-permissions`	Preserve the permissions of files and directories when extracting an archive, without subtracting the `umask`.

Archiving files and directories

- **The following command creates an** archive named *archive.tar* with the contents of *file1*, *file2*, and *file3* in the user's home directory.

```
[user@host ~]$ tar -cf archive.tar file1 file2 file3
```

- The above *tar* command can also be executed using the long version options.

```
[user@host ~]$ tar --file=archive.tar --create file1 file2 file3
```

- To create the tar archive named, */root/etc.tar*, with the */etc* directory as content as user root.

```
[root@host ~]# tar -cf /root/etc.tar /etc
```

Creating a compressed archive

- The *tar* command supports three compression methods.
  - gzip
  - bzip2
  - xz
- Use one of the following options to create a compressed ***tar*** archive:

OPTION	DESCRIPTION
`-z`, `--gzip`	Use gzip compression (`.tar.gz`).
`-j`, `--bzip2`	Use bzip2 compression (`.tar.bz2`). bzip2 typically achieves a better compression ratio than gzip.
`-J`, `--xz`	Use xz compression (`.tar.xz`). The xz compression typically achieves a better compression ratio than bzip2.

Compress and extract files using gzip, bzip2, and xz

- Additionally, ***gzip, bzip2, and xz*** can be used independently to compress single files.

```
[root@lab01 ~]# gzip etc.tar
[root@lab01 ~]# bzip2 abc.tar
[root@lab01 ~]# xz myarchive.tar
```

- The corresponding commands to decompress are ***gunzip, bunzip2, and unxz***.

```
[root@lab01 ~]# gunzip etc.tar.gz
[root@lab01 ~]# bunzip2 abc.tar.bz2
[root@lab01 ~]# unxz myarchive.tar.xz
```

Transferring files using secure copy

- The Secure Copy command, ***scp***, which is part of the OpenSSH suite, copies files from a remote system to the local system or from the local system to a remote system.
- The following example demonstrates how to copy the local ***/etc/yum.conf*** and ***/etc/hosts*** files on host, to the remote user's home directory on the remote host remote system:

```
[user@host ~]$ scp /etc/yum.conf /etc/hosts remoteuser@remotehost:/home/remoteuser
```

Transferring files using secure copy

- You can also copy a file in the other direction, from a remote system to the local file system. In this example, the file */etc/hostname* on remotehost is copyed to the local directory */home/user*.

```
[user@host ~]$ scp remoteuser@remotehost:/etc/hostname /home/user
```

- To copy a whole directory tree recursively, use the ***-r*** option. In the following example, the remote directory ***/var/log*** on remotehost is copied recursively to the local directory ***/tmp/*** on host.

```
[user@host ~]$ scp -r root@remoteuser:/var/log /tmp
```

Transferring files using the secure file transfer program

- To interactively upload or download files from a SSH server, use the Secure File Transfer Program, ***sftp.***
  A session with the ***sftp*** command uses the secure authentication mechanism and encrypted data transfer to and from the SSH server.

```
[user@host ~]$ sftp remoteuser@remotehost
remoteuser@remotehost's password: password
Connected to remotehost.
sftp>
```

SUMMARY

In this chapter, you learned:

- The **tar** command creates an archive file from a set of files and directories, extracts files from the archive, and lists the contents of an archive.
- The **tar** command provides a set of different compression methods reduce archive size.
- Besides providing a secure remote shell, the SSH service also provides the **scp** and **sftp** commands as secure ways to transfer files from and to a remote system running the SSH server.

RH124

Installing and Updating Software Packages



Red Hat | Enterprise Linux

MOOCA



Installing and Updating Software Packages

Goal

- Download, install, update, and manage software packages from Red Hat and Yum package repositories.

Objectives

- Install, update, or remove software packages using **Yum** and **RPM** tools.

MOOCA

Explaining and investigating RPM software packages

- RPM package files names consist of four elements (plus the .rpm suffix): ***name-versionrelease.architecture***



- ***NAME*** is one or more words describing the contents (***coreutils***).
- ***VERSION*** is the version number of the original software (***8.30***).
- ***RELEASE*** is the release number of the package based on that version, and is set by the packager, who might not be the original software developer (***4.el8***).
- ***ARCH*** is the processor architecture the package was compiled to run on. ***noarch*** indicates that this package's contents are not architecture-specific (as opposed to ***x86_64*** for 64-bit, ***aarch64*** for 64-bit ARM, and so on).

```
[root@server ~]# uname –a  and cat /etc/redhat-release
```

Explaining RPM software packages

- The *rpm* utility is a low-level tool that can get information about the contents of package files and installed packages.
- ***rpm -qa:*** List all installed packages
- ***rpm -qf FILENAME:*** Find out what package provides FILENAME
- ***rpm -q yum:*** List what version of the package is currently installed
- ***rpm -qi yum:*** Get detailed information about the package
- ***rpm -ql yum:*** List the files installed by the package
- ***rpm -qc yum:*** List just the configuration files installed by the package
- ***rpm -qd yum:*** List just the documentation files installed by the package
- ***rpm -q --scripts openssh-server:*** List shell scripts that run before or after the package is installed or removed
- ***rpm -q --changelog:*** list change information for the package

Installing RPM packages

- The *rpm* command can also be used to install an RPM package that you have downloaded to your local directory.
- ***i*** → install package(s)
- ***v*** → Verbosity provide more detailed output
- ***h*** → Installation progress ####

```
[root@host ~]# rpm -ivh wonderwidgets-1.0-4.x86_64.rpm
Verifying...                          ################################# [100%]
Preparing...                          ################################# [100%]
Updating / installing...
   1:wonderwidgets-1.0-4              ################################# [100%]
[root@host ~]#
```

Installing and updating software packages with YUM

- The low-level *rpm* command can be used to install packages, but it is not designed to work with package repositories or resolve dependencies from multiple sources automatically.
- *Yum* is designed to be a better system for managing RPM-based software installation and updates. The ***yum*** command allows you to install, update, remove, and get information about software packages and their dependencies.

Finding Software with Yum

- ***yum help*** displays usage information.
- ***yum list*** displays installed and available packages

```
[user@host ~]$ yum list 'http*'
```

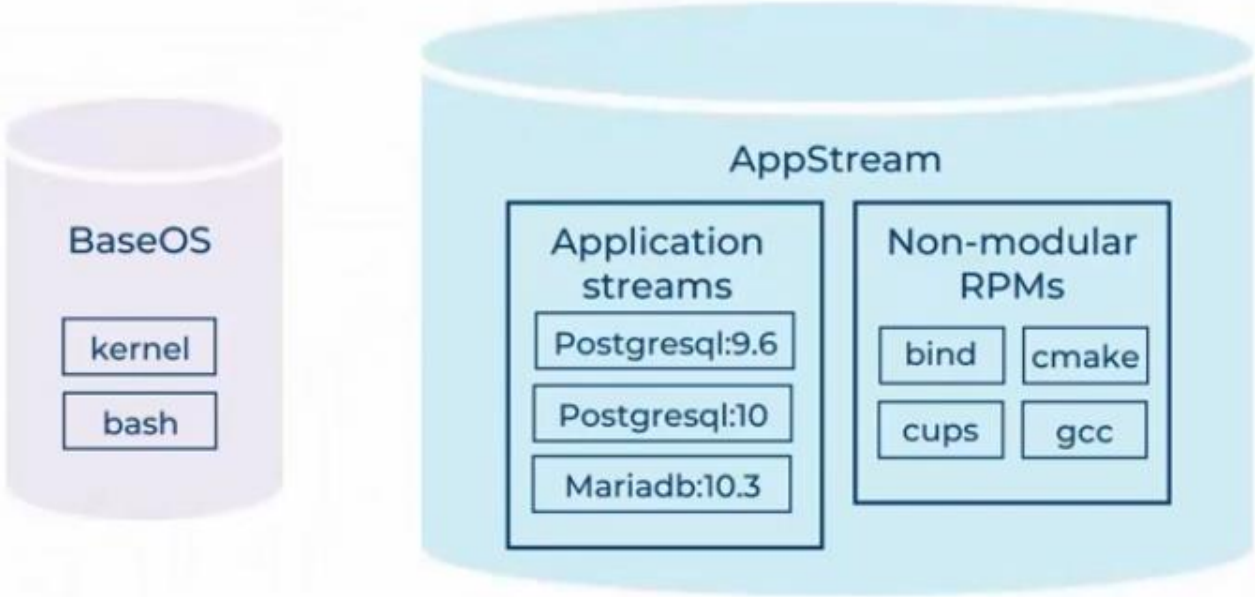
- ***yum search KEYWORD*** lists packages by keywords found in the name and summary fields only.

```
[user@host ~]$ yum search all 'web server'
```

Introduction to application stream

Two main repositories:

- The ***BaseOS*** repository mainly concerns the operating system.
- The ***AppStream*** repository contains application modules and non-modular RPMs.



IMPORTANT

- Both ***BaseOS*** and ***AppStream*** are a necessary part of a Red Hat Enterprise Linux 9 system.

Installing and updating software packages with YUM

- The low-level *rpm* command can be used to install packages, but it is not designed to work with package repositories or resolve dependencies from multiple sources automatically.
- *Yum* is designed to be a better system for managing RPM-based software installation and updates. The ***yum*** command allows you to install, update, remove, and get information about software packages and their dependencies.

Finding Software with Yum

- ***yum help*** displays usage information.
- ***yum list*** displays installed and available packages

```
[user@host ~]$ yum list 'http*'
```

- ***yum search KEYWORD*** lists packages by keywords found in the name and summary fields only.

```
[user@host ~]$ yum search all 'web server'
```

Installing and updating software packages with YUM

- ***yum update PACKAGENAME*** obtains and installs a newer version of the specified package, including any dependencies.
- With no PACKAGENAME specified, it installs all relevant updates.

```
[user@host ~]$ yum update
```

- ***yum remove PACKAGENAME*** removes an installed software package, including any supported packages.

```
[user@host ~]$ yum remove httpd
```

- ***yum history*** displays a summary of install and remove transactions

```
[user@host ~]$ yum history
```

Summary of RPM query commands

- Installed packages can be queried directly with the *rpm* command. Add the *-p* option to query a package file before installation.

COMMAND	TASK
`rpm -qa`	List all RPM packages currently installed
`rpm -q NAME`	Display the version of NAME installed on the system
`rpm -qi NAME`	Display detailed information about a package
`rpm -ql NAME`	List all files included in a package
`rpm -qc NAME`	List configuration files included in a package
`rpm -qd NAME`	List documentation files included in a package
`rpm -q --changelog NAME`	Show a short summary of the reason for a new package release
`rpm -q --scripts NAME`	Display the shell scripts run on package installation, upgrade, or removal

Summary of YUM commands

- Packages can be located, installed, updated, and removed by name or by package groups.

TASK:	COMMAND:
List installed and available packages by name	`yum list [NAME-PATTERN]`
List installed and available groups	`yum group list`
Search for a package by keyword	`yum search KEYWORD`
Show details of a package	`yum info PACKAGENAME`
Install a package	`yum install PACKAGENAME`
Install a package group	`yum group install GROUPNAME`
Update all packages	`yum update`
Remove a package	`yum remove PACKAGENAME`
Display transaction history	`yum history`

SUMMARY

In this chapter, you learned:

- Software is provided as RPM packages, which make it easy to install, upgrade, and uninstall software from the system.
- The ***rpm*** command can be used to query a local database to provide information about the contents of installed packages and install downloaded package files.
- ***yum*** is a powerful command-line tool that can be used to install, update, remove, and query software packages.
- Red Hat Enterprise Linux 9 uses Application Streams to provide a single repository to host multiple versions of an application's packages and its dependencies.

Accessing Linux File Systems

Goal

- Access, inspect, and use existing file systems on storage attached to a Linux server.

Objectives

- List the block devices and file systems.
- Mount and unmount file systems and partitions in Linux.
- Search for files on mounted file systems using the find and locate commands.

File system and mount points

- To make the contents of a file system available in the file-system hierarchy, it must be **mounted** on an empty directory.

- This directory is called a **mount point**. Once mounted, if you use **ls** to list that directory, you will see the contents of the mounted file system, and you can access and use those files normally. Many file systems are automatically mounted as part of the boot process.

Examining file systems

- To get an overview of local and remote file system devices and the amount of free space available, run the **df** command
- To improve readability of the output sizes you can use human-readable option: **-h**

```
[user@host ~]$ df -h
Filesystem      Size  Used Avail Use% Mounted on
devtmpfs        892M     0  892M   0% /dev
tmpfs           915M     0  915M   0% /dev/shm
tmpfs           915M   17M  899M   2% /run
tmpfs           915M     0  915M   0% /sys/fs/cgroup
/dev/vda3       8.0G  1.4G  6.7G  17% /
/dev/vda1      1014M  166M  849M  17% /boot
tmpfs           183M     0  183M   0% /run/user/1000
```

Examining file systems

- For more detailed information about space used by a certain directory tree, use the **du** command. The **du** command has **-h** option to convert the output to human-readable format.
- The **du** command shows the size of all files in the current directory tree recursively.

```
[root@host ~]# du -h /var/log
...output omitted...
176K  /usr/share/smartmontools
184K  /usr/share/nano
8.0K  /usr/share/cmake/bash-completion
8.0K  /usr/share/cmake
369M  /usr/share
```

Identifying the block device

- Use the **lsblk** command to list the details of a specified block device or all the available devices.

```
[root@RHEL9 ~]# lsblk
NAME         MAJ:MIN RM  SIZE RO TYPE MOUNTPOINTS
sr0           11:0    1 1024M  0 rom
nvme0n1      259:0    0   20G  0 disk
├─nvme0n1p1 259:1    0  512M  0 part /boot
├─nvme0n1p2 259:2    0   10G  0 part /
├─nvme0n1p3 259:3    0    4G  0 part /home
├─nvme0n1p4 259:4    0    1K  0 part
└─nvme0n1p5 259:5    0    1G  0 part [SWAP]
```

Mounting and unmounting file systems

- There are two common ways to specify the file system on a disk partition to the **mount** command:
  - With the name of the device file in **/dev** containing the file system.
  - With the **UUID** written to the file system, a universally-unique identifier.
- The following example mounts the file system in **the /dev/vdb1** partition on the directory **/mnt/data**.

```
[root@host ~]# mount /dev/vdb1 /mnt/data
```

Mounting by file-system UUID

- One stable identifier that is associated with a file system is its **UUID**, a very long hexadecimal number that acts as a universally-unique identifier. This UUID is part of the file system and remains the same as long as the file system is not recreated.

- The **lsblk -fp** command lists the full path of the device, along with the UUIDs and mount points, as well as the type of file system in the partition. If the file system is not mounted, the mount point will be blank

```
[root@host ~]# lsblk -fp
NAME        FSTYPE LABEL UUID                                 MOUNTPOINT
/dev/vda
├─/dev/vda1 xfs          23ea8803-a396-494a-8e95-1538a53b821c /boot
├─/dev/vda2 swap         cdf61ded-534c-4bd6-b458-cab18b1a72ea [SWAP]
└─/dev/vda3 xfs          44330f15-2f9d-4745-ae2e-20844f22762d /
/dev/vdb
└─/dev/vdb1 xfs          46f543fd-78c9-4526-a857-244811be2d88
```

Mounting by file-system UUID

- Mount the file system by the UUID of the file system

```
[root@host ~]# mount UUID="46f543fd-78c9-4526-a857-244811be2d88" /mnt/data
```

Unmounting file systems

- To unmount a file system, the **umount** command expects the mount point as an argument.
- Unmounting is not possible if the mounted file system is in use. For the **umount** command to succeed, all processes needs to stop accessing data under the mount point.
- In the example below, the **umount** fails because the file system is in use (the shell is using /mnt/data as its current working directory), generating an error message.

```
[root@host ~]# umount /mnt/data
```

```
[root@host ~]# cd /mnt/data
[root@host data]# umount /mnt/data
umount: /mnt/data: target is busy.
```

Unmounting file systems

- The **lsof** command lists all open files and the process accessing them in the provided directory. It is useful to identify which processes currently prevent the file system from successful unmounting.

```
[root@host data]# lsof /mnt/data
```

Locating files on the system

- A system administrator needs tools to search for files matching certain criteria on the file system.
- This section discusses two commands that can search for files in the file-system hierarchy.
  - The **locate** command searches a pregenerated index for file names or file paths and returns the results instantly.
  - The **find** command searches for files in real time by crawling through the file-system hierarchy.

Locating file by name

- The **locate** command finds files based on the name or path to the file. It is fast because it looks up this information from the **mlocate** database.
- However, this database is not updated in real time, and it must be frequently updated for results to be accurate.
- This also means that locate will not find files that have been created since the last update of the database.
- The **locate** database is automatically updated every day. However, at any time the root user can issue the **updatedb** command to force an immediate update.

```
[root@host ~]# updatedb
```

Locating file by name

- The **locate** command restricts results for unprivileged users. In order to see the resulting file name, the user must have search permission on the directory in which the file resides.

```
[user@host ~]$ locate passwd
```

- The **-i** option performs a case-insensitive search. With this option, all possible combinations of upper and lowercase letters match the search. The **-n** option limits the number of returned search results by the locate command.

```
[user@host ~]$ locate -i messages
[user@host ~]$ locate -n 5 messages
```

Searching for files in real time

- The **find** command locates files by performing a real-time search in the file-system hierarchy.
- It is **slower** than **locate**, but more accurate.
- It can also search for files based on criteria other than the file name, such as the permissions of the file, type of file, its size, or its modification time.
- The find command looks at files in the file system using the user account that executed the search.
- The user invoking the find command must have read and execute permission on a directory to examine its contents.

To find a file by name

- to search for files named sshd_config starting from the / directory, run the following command:

```
[root@host ~]# find / -name sshd_config
```

- In the following example, search for files starting in the / directory that end in .txt:

```
[root@host ~]# find / -name '*.txt'
```

- Use the ***-iname*** option to search files with case-insensitive text, messages, in their names in the / directory on host.

```
[root@host ~]# find / -iname '*messages*'
```

Searching files based on ownership

- The **find** command can search for files based on their ownership or permissions. Useful options when searching by owner are **-user** and **-group**, which search by name, and **-uid** and -**gid**, which search by ID.
  - Search for files owned by user in the **/home/user** directory on host.

```
[user@host ~]$ find -user user
```

  - Search for files owned by the group user in the **/home/user** directory on host.

```
[user@host ~]$ find -group user
```

Searching files based on ownership

- Search for files owned by user **ID 1000** in the **/home/user** directory on host.

```
[user@host ~]$ find -uid 1000
```

- The **-user**, and **-group** options can be used together to search files where file owner and group owner are different. The example below list files that are both owned by user root and affiliated with group mail.

```
[root@host ~]# find / -user root -group mail
```

Searching files based on permissions

- The **-perm** option is used to look for files with a particular set of permissions. Permissions can be described as octal values, with some combination of 4, 2, and 1 for read, write, and execute.
- The following command matches any file for which the user has read, write, and execute permissions, members of the group have read and write permissions, and others have read-only access:

```
[root@host ~]# find /home -perm 764
```

- To match files for which the user has at least write and execute permissions, and the group has at least write permissions, and others have at least read access:

```
[root@host ~]# find /home -perm -324
```

Searching files based on permissions

- The **-perm** option is used to look for files with a particular set of permissions. Permissions can be described as octal values, with some combination of 4, 2, and 1 for read, write, and execute.

- The following command matches any file for which the user has read, write, and execute permissions, members of the group have read and write permissions, and others have read-only access:

```
[root@host ~]# find /home -perm 764
```

- To match files for which the user has at least write and execute permissions, and the group has at least write permissions, and others have at least read access:

```
[root@host ~]# find /home -perm -324
```

Searching files based on permissions

- To match files for which the user has read permissions, or the group has at least read permissions, or others have at least write access:

```
[root@host ~]# find /home -perm /442
```

Searching files based on size

- The find command can look up files that match a size specified with the -size option, followed by a numerical value and the unit. Use the following list as the units with the -size option:
  - **k**, for kilobyte
  - **M**, for megabyte
  - **G**, for gigabyte

- The example below shows how to search for files with a size of 10 megabytes, rounded up.

```
[user@host ~]$ find -size 10M
```

- To search the files with a size more than 10 gigabytes.

```
[user@host ~]$ find -size +10G
```

- To list all files with a size less than 10 kilobytes

```
[user@host ~]$ find -size -10k
```

Searching files based on Modification time

- The **-mmin** option, followed by the time in minutes, searches for all files that had their content changed at **n** minutes ago in the past. The file's timestamp is always rounded down. It also supports fractional values when used with ranges (**+n** and **-n**).

- To find all files that had their file content changed 120 minutes ago on host, run:

```
[root@host ~]# find / -mmin 120
```

- The **+** modifier in front of the amount of minutes looks for all files in the / that have been modified more than n minutes ago. In this example, files that were modified more than 200 minutes ago are listed.

```
[root@host ~]# find / -mmin +200
```

Searching files based on Modification time

- The **-** modifier changes the search to look for all files in the / directory which have been changed less than n minutes ago. In this example, files that were modified less than 150 minutes ago are listed.

```
[root@host ~]# find / -mmin -150
```

Searching files based on type

- The **-type** option in the **find** command limits the search scope to a given file type. Use the following list to pass the required flags to limit the scope of search:
  - **f**, for regular file
  - **d**, for directory
  - **l**, for soft link
  - **b**, for block device
- Search for all directories in the /etc directory on host.

```
[root@host ~]# find /etc -type d
```

- Search for all soft links on host.

```
[root@host ~]# find / -type l
```

- Generate a list of all block devices in the /dev directory on host:

```
[root@host ~]# find /dev -type b
```

SUMMARY

In this chapter, you learned:

- Storage devices are represented by a special file type called block device.
- The **df** command reports total disk space, used disk space, and free disk space on all mounted regular file systems.
- The **mount** command allows the root user to manually mount a file system.
- All processes need to stop accessing the mount point in order to successfully unmount the device.
- The **find** command performs a real-time search in the local file systems to find files based on search criteria.