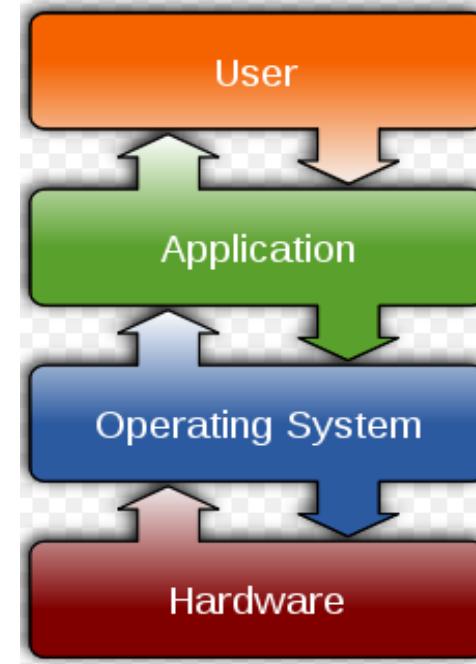


Chapter 1 | Introduction to Linux

- Operating Systems roles and functions
- Linux history and origin
- The concept of Free and Open Source software
- Linux philosophy and principles

What is an Operating System

- The operating system is a special computer program (software) that controls the computer (hardware)

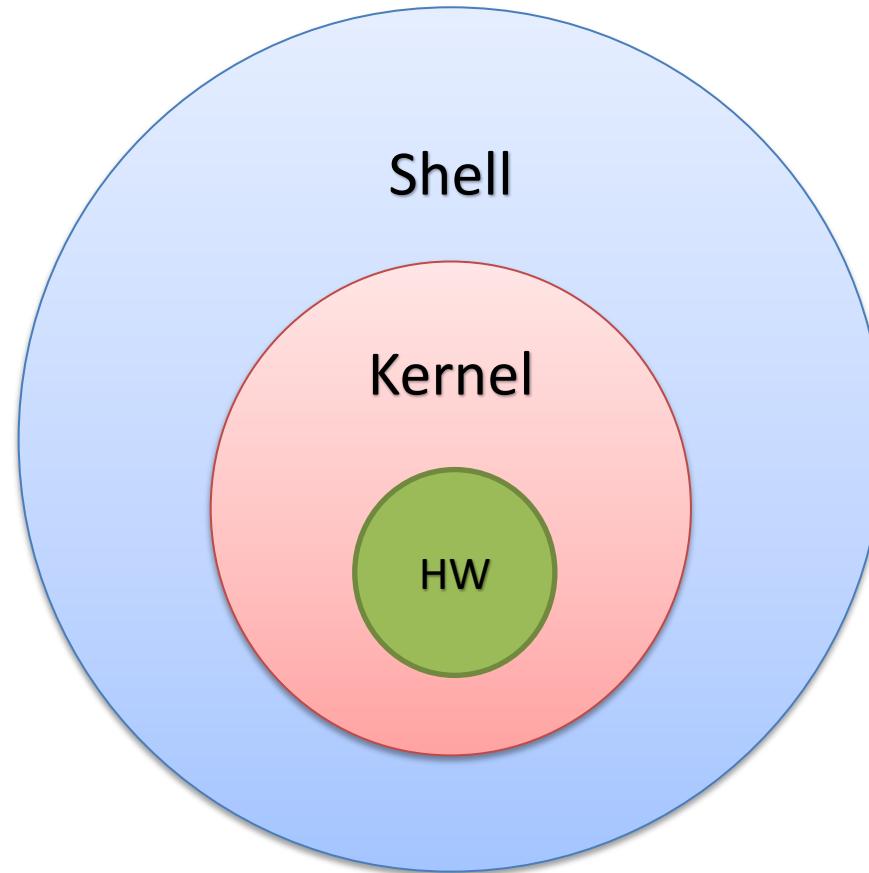


Operating systems functions



Operating systems Big Elements

- Kernel
- Shell



What is Linux?

- Linux is a Unix-like computer operating system assembled under the model of free and open source software development and distribution
- Linux can be found in many devices:
 - Personal computers
 - Servers
 - Mainframes
 - Supercomputers
 - Embedded systems (Mobile phones, Tablet computers, Network routers, Televisions and Video game consoles).

Linux History and Origins

- Unix
- The GNU Project (1983)
- MINIX
- Linux (1991)



The Open Source Idea

- When programmers can read, distribute and change the software code:
 - Other programmers can adapt it, fix it, debug it.
 - And they can do it more quickly rather than developers at conventional companies.
 - The resulting software will be more flexible and of a better quality because more people have tested it in more different conditions than the closed software developer ever can.

The Four essential freedoms

A program is free software if the program's users have the four essential freedoms:

1. **Freedom 0**: The freedom to run the program as you wish, for any purpose .
2. **Freedom 1**: The freedom to study how the program works, and change it so it does your computing as you wish. Access to the source code is a precondition for this.
3. **Freedom 2**: The freedom to redistribute copies so you can help others.
4. **Freedom 3**: The freedom to distribute copies of your modified versions to others .

Benefits come with open source

- From the software owner point of view:
 - I can Sell the software (but the code must still remain free).
 - I can provide training services.
 - The software will spread faster.
- From the developer point of view:
 - Writing my name inside the code and credits section will add a new value to my CV

Linux Distributions

Desktop distributions

- Fedora
- Ubuntu
- Linux Mint

Mobile distributions

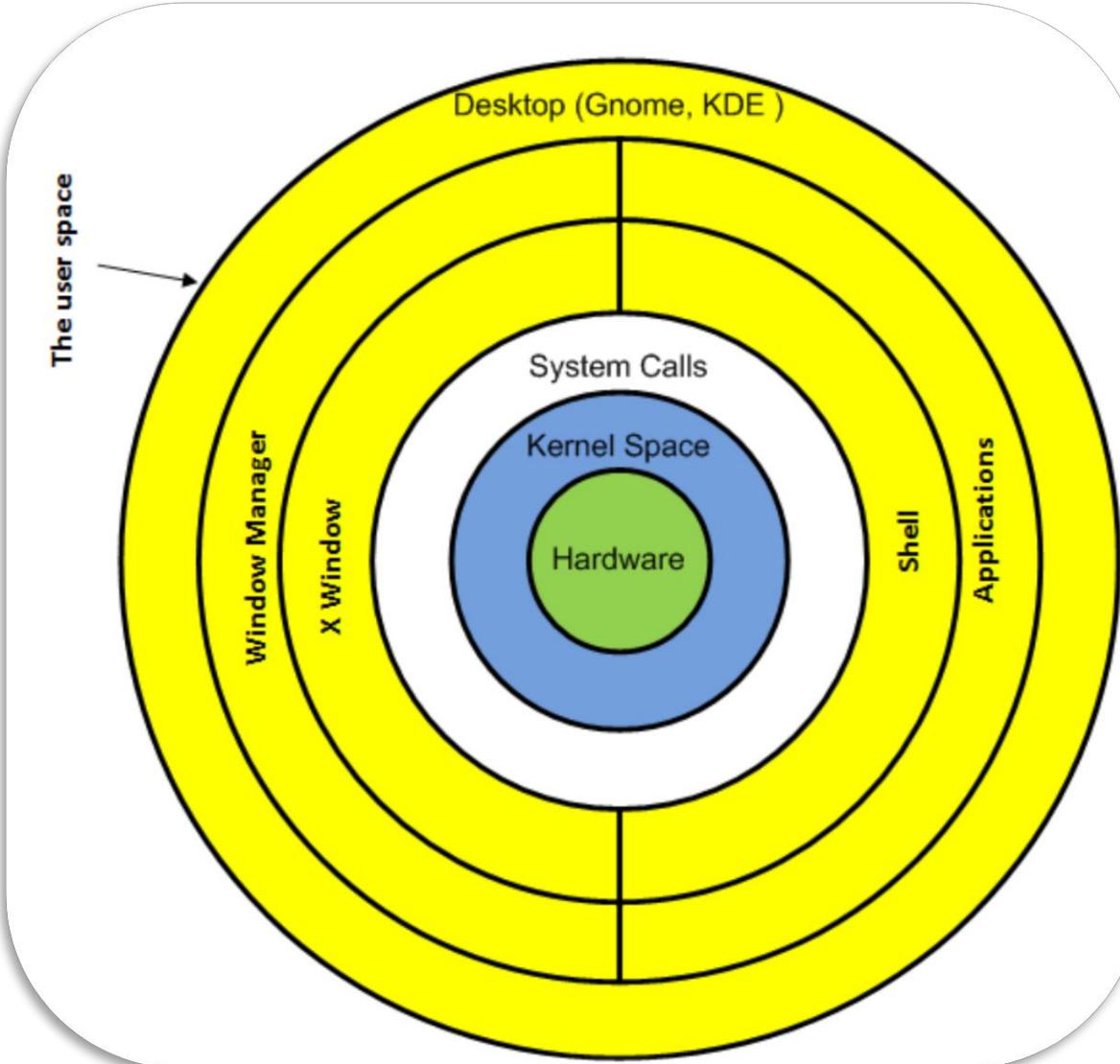
- Android
- Ubuntu
- Firefox OS

Server distributions

- Red Hat
- Debian
- OpenSUSE
- Slackware
- CentOS



Linux System architecture



Linux System architecture

- Hardware
- The kernel
- The Shell (sh, bash, csh, ksh)
- Users Applications
- The X Window system



Linux philosophy and principles

- Everything is a file (including hardware).
- Small, single-purpose programs.
- Make each program do one thing well.
- Ability to chain programs together to perform complex tasks.
- Avoid captive user interfaces.
- Configuration data stored in text.



Chapter 2 | Linux Usage Basics

- Logging in and out (GUI & Text).
- Switching users identities.
- Changing users passwords.
- Editing text files.

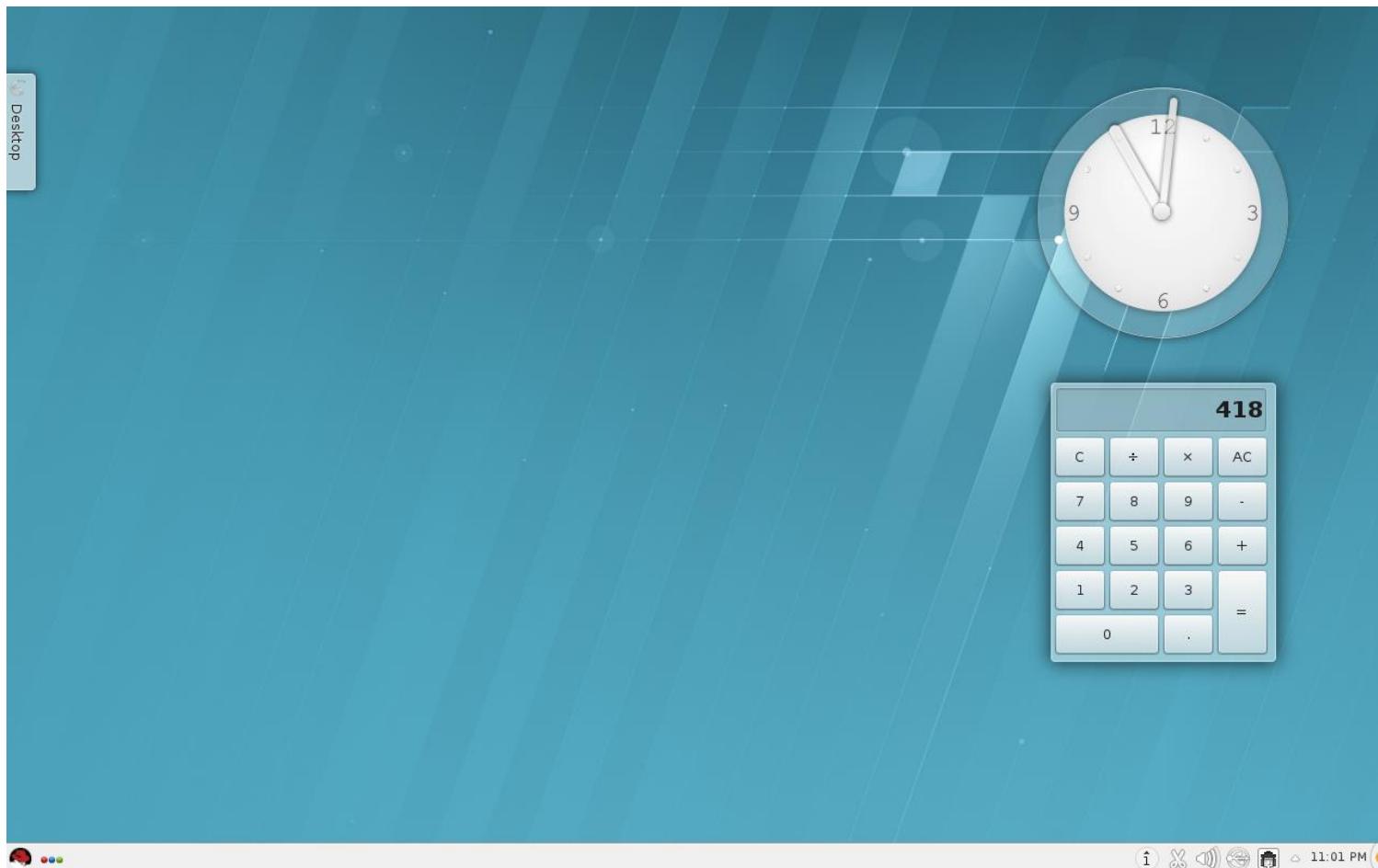
Logging into the system

- Graphical login
- Virtual console login (text-based)



Linux Desktops

- KDE



Linux Desktops

- GNOME



Access the command line from X

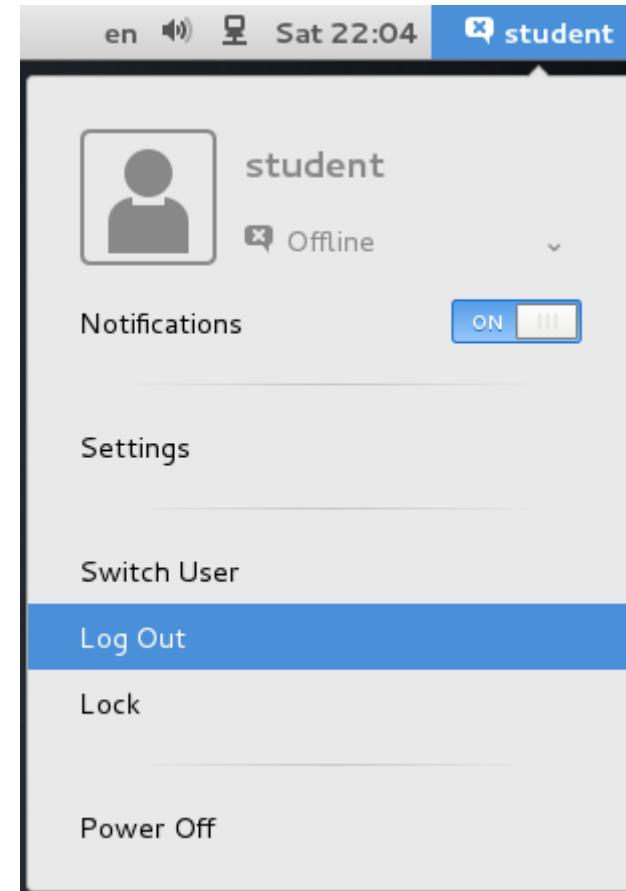
- From the Gnome desktop you can open a terminal that give you the shell prompt by clicking on:

Applications → Utilities → Terminal.



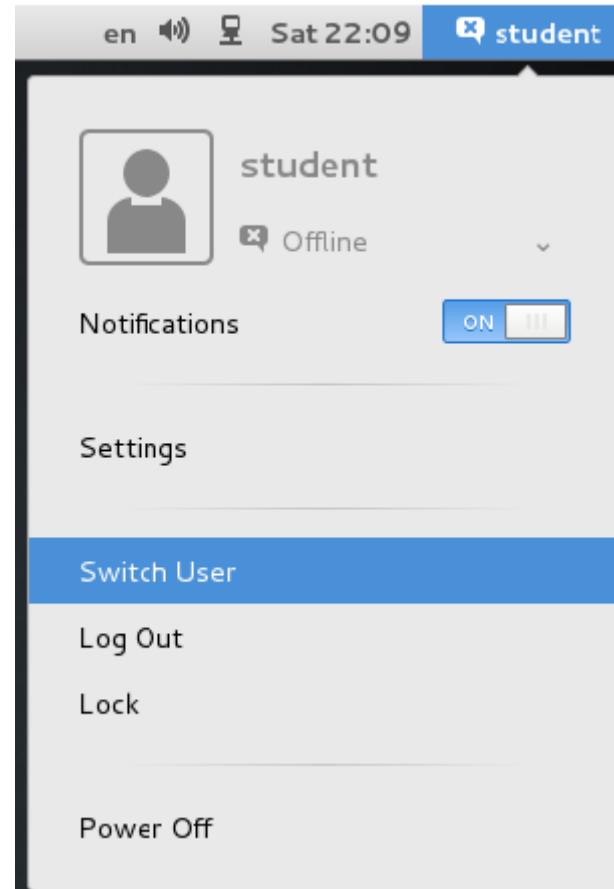
Logging out users

- The Logging out process terminates the user session completely.



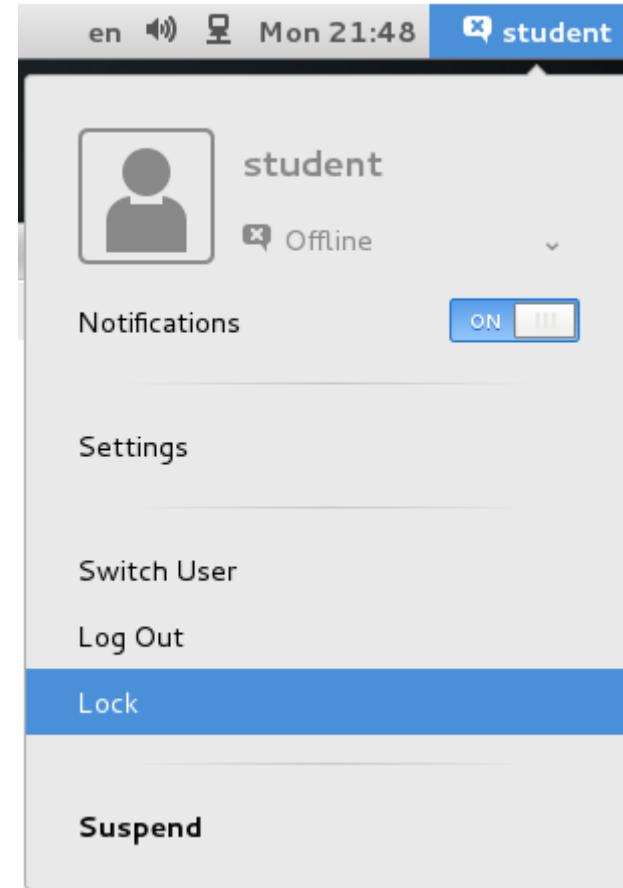
Switching users (GUI)

- Switch to another user to become the current working user while the previously logged users' sessions still running.
- The process of switching users does not terminates the previous user session



Locking Screen

- GUI lock doesn't work with virtual consoles
- Use “vlock” to lock the current virtual console
- Use “vlock -a” to lock all virtual consoles



The “root” user

- The “root” user is a special administrative account, also called the super user.
- The “root” has near complete control over the system and a nearly unlimited capacity to **damage** it!
- It is always recommended to login as a normal user, and then escalate your privileges to the “root” user only when needed



Changing user Identity (CLI)

`$ su [-] <user>`

- `$ su student` Change to user student
- `$ su - student` Change to user student

- `$ su visitor` Change to user visitor
- `$ su - visitor` Change to user visitor

- `$ su` Change to user “root”
- `$ su -` Change to user “root”

LAB

01 - LAB TTY & GUI Login

Changing the user password using GUI Tools

- Navigate to Application → Activities Overview → type Setting → Users, then click the Password box .
- Enter the current password.
- Enter the new password.
- Confirm the new password.

Changing the user password using command line

- Open the command shell
- Type the command “passwd” | “userpasswd” at the shell prompt
- Enter the current password
- Enter the new password
- Confirm the new password

Password strength-checking (for regular users)

- It must not too short (At least 6 characters)
- It must not be too simple, use a combination of letters, capitalization, numbers and punctuations for the best results.
- It must not be based on a dictionary word.



Password strength-checking (for the root users)

- Note: as the system administrator (the **root** user)
 - You may set any user's password to anything you like.
 - Only you well be warned if you don't meet normal password complexity requirements.

The user's home directory

- A home directory is a file system directory on a multi-user operating system containing files for a given user of the system.
- It provides a mechanism for separating and protecting the files of the individual users while allowing them to use the system simultaneously.

Editing plain text files

- gedit demonstration.
- nano demonstration.

LAB

02 - LAB (Linux Usage Basics)

Chapter 3 | Running Commands And Getting Help

- The command line syntax and structure
- Getting help

The Command Line Interface (CLI)

The Command Line Interface (CLI)

- The command line interface is a text based user interface in which the user type commands and receives a response back from the system.
- provided by a program called the shell

The Command Line Interface (CLI)

- The standard prompt has the following format:
`[<Login name>@<Machine name> <current directory>]${`
- The “\$” is replaced by “#” if the shell is running as the “root” user.

The Command Syntax

- The general format of the command is:

command [options] [arguments] ↵

Command : Specifies what the system will do.

Option : Specifies how the command is run (a modifier).

Options start with:

- Single dash (-) **Short option**
- or Double dash (--) **Long option**

Argument : Specifies what is to be affected (a file, a directory, or text).

Commands without options or arguments

- \$ uname
- \$ date
- \$ clear
- \$ cal
- \$ ls

Commands with only options

- \$ uname -a
- \$ uname -s
- \$ uname --kernel-name
- \$ uname -r
- \$ uname --kernel-release
- \$ uname -r -s
- \$ uname -s -r
- \$ uname -rs
- \$ uname -sr
- \$ uname --kernel-release --kernel-name

Command with only arguments

- \$ cal 11 2017
- \$ ls Desktop

Commands with both options and arguments

- \$ ls -l Desktop
- \$ cal -3 5 2018

Command Types

- **Internal commands :**
 - These commands are made available by the shell itself.
- **External commands :**
 - The shell does not execute these commands by itself but launches executable files.
- Use the “type” command to find out the type of a command (internal or external)

\$ type echo (internal command)

\$ type date (external command)



Getting Help

Getting Help

Getting Help in GUI Environment

Getting Help in GUI Environment

From GNOME Desktop:

- Application → Documentation → Help
- F1 (While you are on any GNOME Application)

From GUI pts:

- \$ yelp
- \$ gnome-help

Getting Help

Getting Help in Textual Environment

Getting Help (the --help option)

- --help gives a quick overview of the command syntax.

\$ date --help

\$ uname --help

- Note:
 - Anything between ([]) is optional.
 - Anything followed by (...) represents a list of arbitrary-length of that thing.
 - (|) mean that you can choose any one of them.

Getting Help (the whatis command)

- The “whatis” command give a short description about a certain command .
- \$ whatis clear
 - clear (1) - clear the terminal screen
- \$ whatis cal date
 - cal (1) - displays a calendar
 - cal (1p) - print a calendar
 - Date::Format (3pm) - Date formatting subroutines
 - Date::Parse (3pm) - Parse date strings into time values
 - date (1) - print or set the system date and time
 - date (1p) - write the date and time



Getting Help for internal commands and keywords

\$ help

List all internal commands and keywords

\$ help <name>

Find out more about <name>

\$ help help

Find out more about the “help” internal
command itself



The Manual Pages (man pages)

- The “man pages” are the official collection of quick reference sheets for Linux systems.

```
$ man [options] <command>
```

The Manual Pages (Sections)

- The Linux manual can be thought of as a single large book which is divided into sections or chapters.
- Each section contains man pages relevant to a particular type of information

The Manual Pages (Sections)

Section	Types of man pages
1	User commands
2	Kernel system calls
3	Library functions
4	Special files and devices
5	File formats
6	Games
7	Conventions, standards and miscellaneous
8	System administration commands
9	Linux kernel API (internal kernel calls) <u>Note:</u> Section 9 is relatively recent addition to Linux and not all the documentation on man sections discuss it



The Manual Pages (Sections)

- Two sections of the manual may contain pages that have the same name.
- In order to distinguish between the pages, man page usually add the section number in parentheses after the name of the man page.

\$ man passwd

\$ man 5 passwd



The Man Page Layout

- The man pages, themselves, have a standard layout

Section	Purpose
NAME	The name, or names, for the command along with a short description.
SYNOPSIS	This section specifies the command syntax.
CONFIGURATION	Rarely present, but describes any configuration options.
DESCRIPTION	Usually a brief description, but some are quite long.
OPTIONS	A brief description of each option.
EXIT STATUS	For commands, this value indicates if there was an error. An exit value of zero means no errors.



The Man Page Layout

Section	Purpose
RETURN VALUE	Usually appears on man pages for library functions.
ERRORS	Describes the error codes returned in the EXIT STATUS.
ENVIRONMENT	Describes any environmental variables used by the command.
FILES	Usually refers to related configuration files.
NOTES	If present, contains any special notes that do not fit within another section.
BUGS	Contains information about special limitations, as true bugs are fixed through new versions.
EXAMPLE	When present, provides useful examples of how to use the command.
AUTHOR	The authors who wrote the man page.
SEE ALSO	A handy reference to any other man pages that are related to the current man page.

Reading the SYNOPSIS section

- \$ man man

bold text

italic text

[-abc]

-a | -b

argument ...

[expression] ...

type exactly as shown.

replace with appropriate argument.

any or all arguments within [] are optional.

options delimited by | cannot be used together.

argument is repeatable.

entire expression within [] is repeatable.



Navigating The Manual Pages

Command	Action
Space or PageDown	Scroll forward one screen
b or PageUp	Scroll back one screen
Down Arrow	Scroll forward one line
Up Arrow	Scroll back one line
g	Go to the first line
G	Go to the last line
/string	Search forward for string
n	Repeat previous search forward
N	Repeat previous search backward
q	Exit man and return to the prompt
h	Display the summary of the <u>less</u> command (used by the man utility)

Identifying Man Pages By Keyword

`man -k <keyword>`

`apropos <keyword>`

This will display all relevant man pages including chapters

- `$ man -k calendar`
- `$ apropos calendar`

cal	(1)	- displays a calendar
cal	(1p)	- print a calendar
difftime	(3p)	- compute the difference between two calendar time values

Getting Help (the Info Documentation)

- Info pages provided in the form of books which are made up of hyperlinked info nodes (like web sites)
- This format is more flexible than man pages allowing more discussion of complex commands and concepts.

`$ pinfo <command>`

Note:

Typing only “pinfo” without any arguments provides an index to all info topics

Navigating The Info Pages (pinfo)

Command	Action
PageDown	Read the next page
PageUp	Read the previous page
Down Arrow	Move between topics for section
Up Arrow	Move between topics for section
/	Search for a pattern
n	Display the next topic
p	Display the previous topic
u	Up a level to overview
q	Quit reading the documentation

The Extended Documentation in /usr/share/doc

- most other documentation is found in the /usr/share/doc directory.
- Installing new software in Linux may install additional documentation (**not documented in man pages or info pages**) inside /usr/share/doc in subdirectories named by the software package name.

The Official Documentation

- Each Linux distribution provides documentation.
- See “**Product Documentation for Red Hat Enterprise Linux 7**” at
https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/

LAB

03 - LAB (Running Commands and Getting Help)

Chapter 4 | Files and Directories

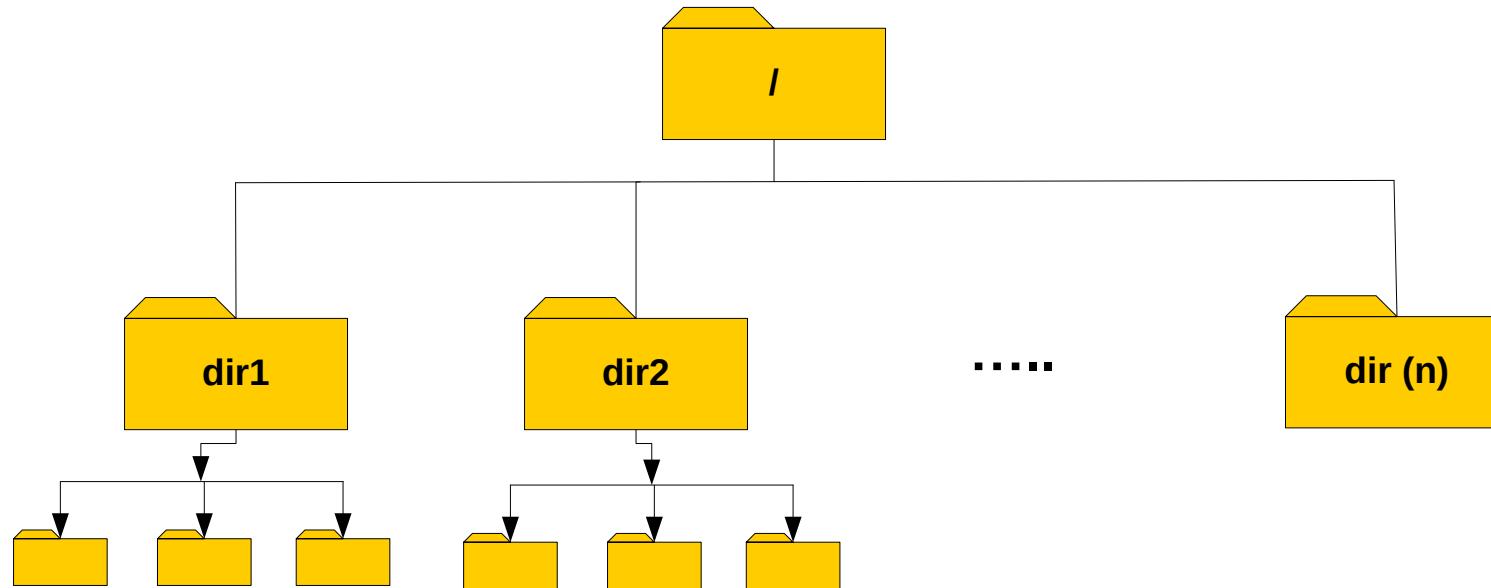
- Linux Hierarchy Tree
- Working with Files and Directories
- Using Links

The Linux files hierarchy concept

- The Linux files hierarchy concept
 - Files
 - Directories
- In Linux files and directories are organized into a single-rooted inverted tree structure, which mean that the whole tree has a single parent directory to all other files and directories.

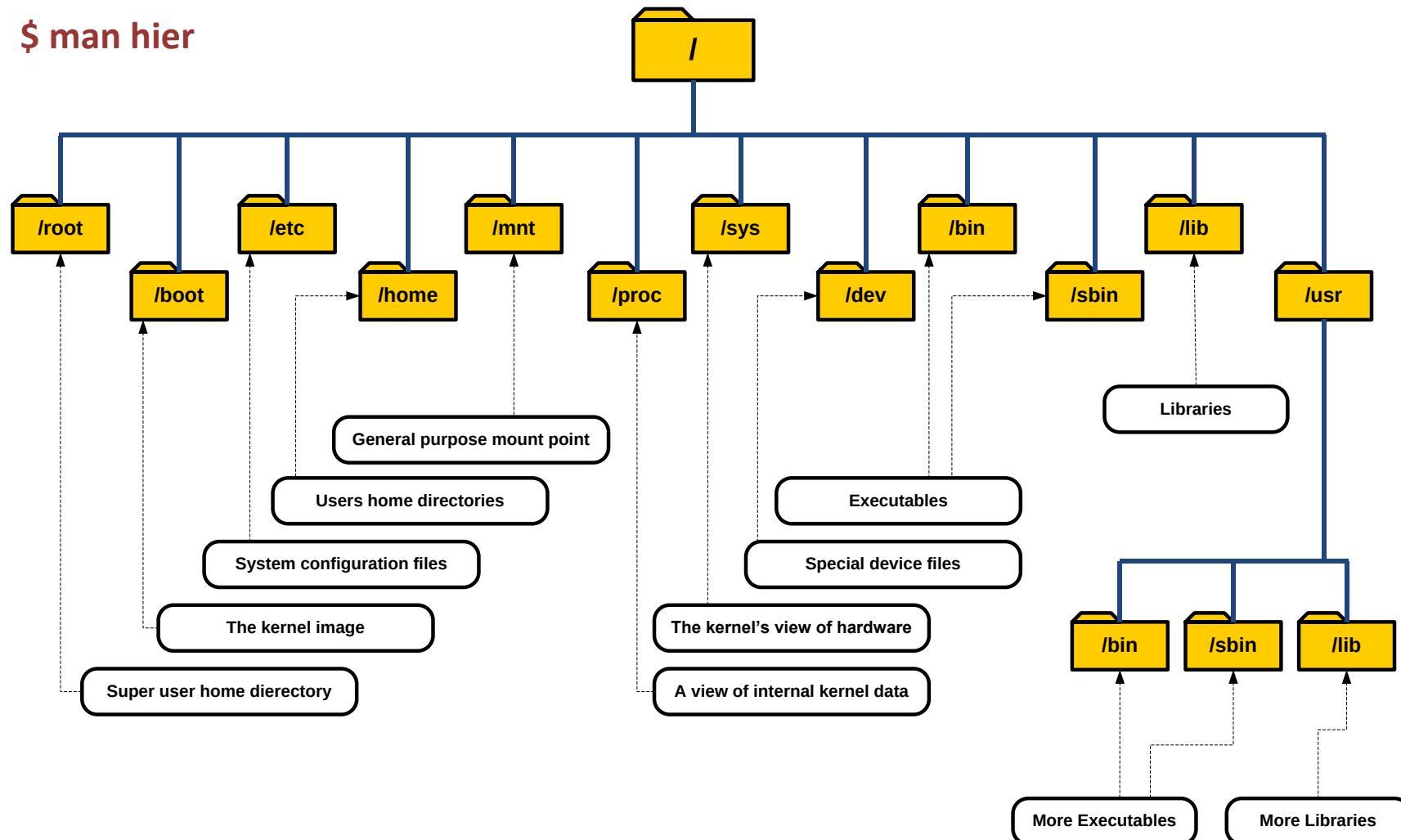


The Linux files hierarchy concept



The Linux Directory Tree

\$ man hier



The Linux Directory Tree

Note:

In Linux, the term root is used in several different ways:

1. **The root account (the super user, who has permission to do anything)**
2. **The root account's home directory (/root)**
3. **The root directory for the entire file system (/).**



The Directory Tree and File Systems

- A Linux system's directory tree usually extends over more than one partition on disk and removable media.
- A file system determines the method of arranging and managing data on a storage medium, such as ext2, ext3, ext4, ReiserFS, XFS, JFS, btrfs, ...).
- Whatever the file system being used, Linux present to the user the same thing :
“a tree-structured hierarchy of file and directory”



File types (categories)

- In Linux systems “Everything is a file”, even the hardware.

File Type	Description
Plain files (Regular files)	Includes texts, graphics, sound files, etc., but also executable programs.
Directories	Also called “folders”; their function, is to help structure storage.
Symbolic links	Contain a path specification redirecting accesses to the link to a different file (similar to “ <u>shortcuts</u> ” in Windows).
Device files	These files serve as interfaces to arbitrary devices such as disk drives. For example, the file /dev/sda represents the first hard disk. Every write or read access to such a file is redirected to the corresponding device.



File and Directory Names

- All characters are valid, except the forward slash (/).
- Case sensitive.
- No suffixes to characterize a file's type (e.g .txt,.mp3, ...etc).
- The dot (.) is a completely ordinary character within a file name.
- The files that their names begin with a dot are hidden files.

Working with files and directories

Current working directory

- The current directory is the directory in which a user is working at a given time.
- Every user is always working within a directory
- `$ pwd`

Displaying the Directory Contents

- `$ ls` List the contents of the current directory.
- `$ ls -a` List all the contents of the current directory, including hidden files.
- `$ ls -l` List the contents of the current directory in the long format.

-rw-rw----	1	user1	user1	1319	Nov 18 15:20	file1
-rw-rw----	1	user1	user1	368	Nov 18 15:20	file2
drwxr-xr-x	5	user1	user1	4096	Nov 18 15:20	dir1
drwxr-xr-x	4	user1	user1	4096	Nov 18 15:20	dir2

File Type
Permissions
Links
Owner
Group
Size
Last modification date and time
Name

Displaying the Directory Contents

- **\$ ls -al** List all the contents of the current directory, including hidden files in the long format.
- **\$ ls -F** Append indicators to the file names according to the file's type.
 - * Executable file.
 - / Directory.
 - @ Symbolic link.
 - none Plain file.

Displaying the Directory Contents

- `$ ls <filename>` List the filename.
- `$ ls <directory>` List the contents of the specified directory.

- `$ ls -d <directory>` List the directory name instead of contents.
- `$ ls -ld <directory>` List the directory name in long format.

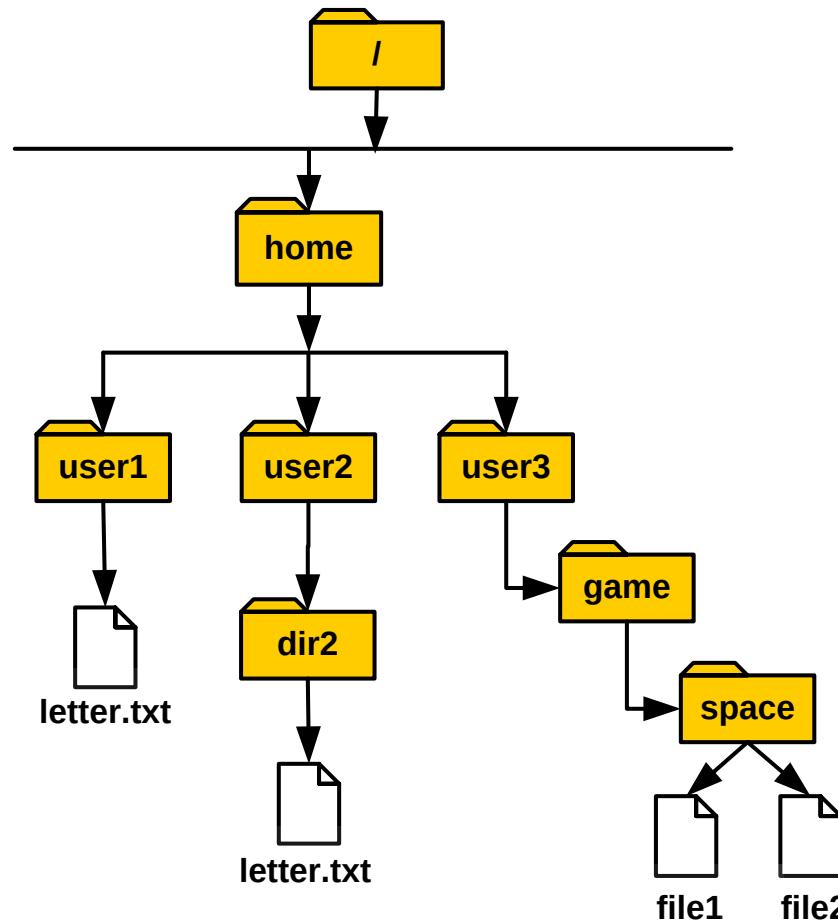
- `$ls -R <directory>` List subdirectories recursively
`$ls --recursive <directory>`

- `$ls -r <directory>` List the directory contents in reverse order using
`$ls --reverse <directory>`

Changing the current working directory

- `$ cd` Change the current working directory to be the user home directory.
- `$ cd <directory>` Change the current working directory to be the specified directory
- `$ cd -` Change to the previous working directory

Absolute and Relative Path Names



Absolute and Relative Path Names

Absolute Path

Gives the complete route of the location of a file or directory

Always starts at the top of the hierarchy (the root directory)

Not dependent on your current location in the hierarchy

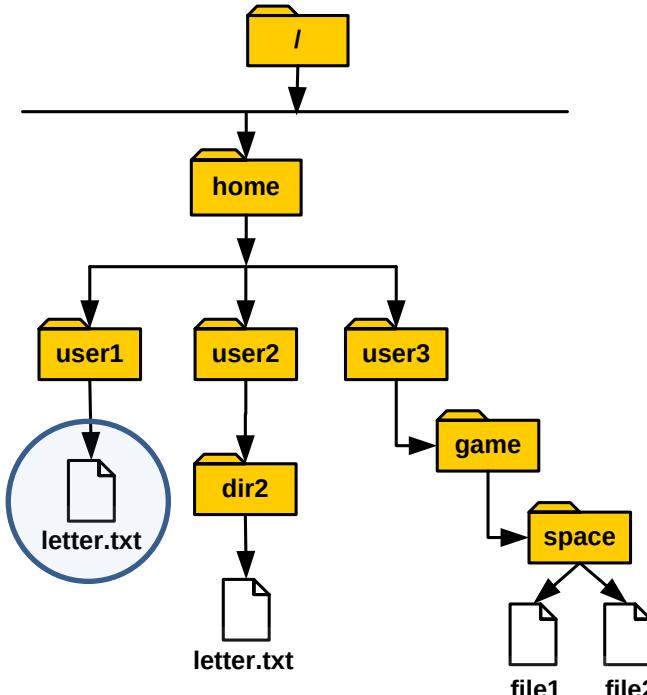
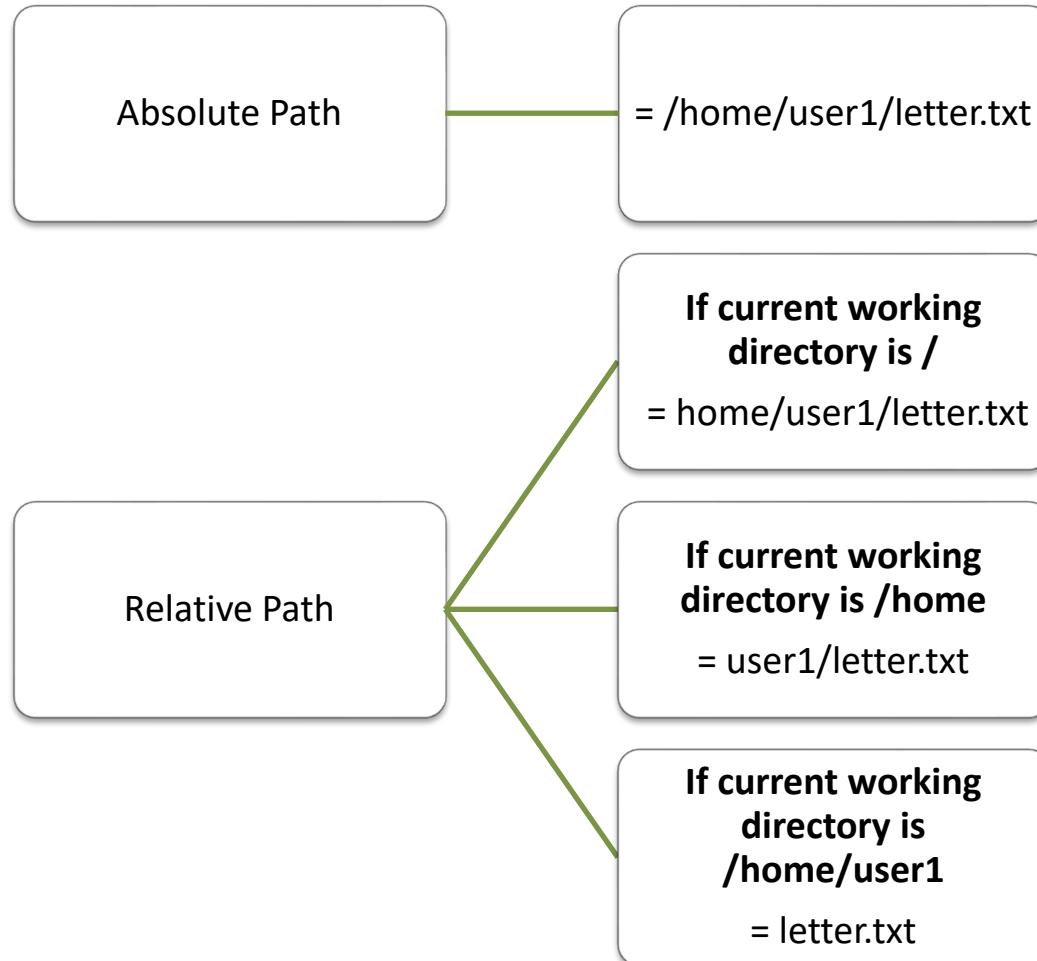
Relative Path

Always starts at your current location in the hierarchy

Is unique relative to your current location only

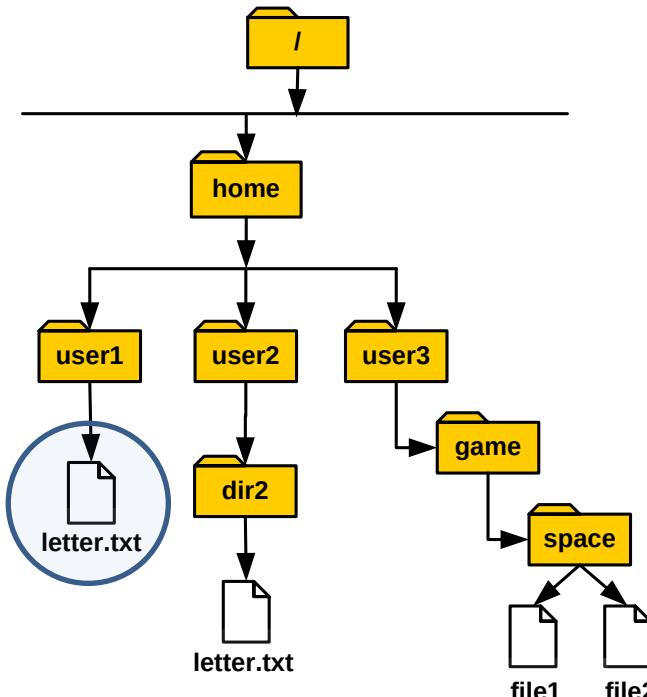
Is often shorter than the absolute path name

Absolute and Relative Path Names for the file letter.txt inside user1



Absolute and Relative Path Names

<u>Command</u>	<u>Result</u>
\$cd /home/user1	
\$pwd	/home/user1
\$ls	letter.txt
\$cd ..	
\$pwd	/home
\$cd -	
\$pwd	/home/user1
\$cd ../user3	
\$pwd	/home/user3
\$ls -ld game/space	drwxrwxr-x. 2 user1 user1 4096 Nov 20 11:14 game/space/
\$cd game/space	
\$pwd	/home/user3/game/space
\$cd ../../user2/dir2	
\$pwd	/home/user2/dir2
\$ls letter.txt	letter.txt
\$ls .	letter.txt
\$ls ../.	dir2
\$cd /	
\$pwd	/
\$ls	
/home/user3/game/space/file1	/home/user3/game/space/file1



Creating and Deleting Directories

- `$ mkdir <Directory name>` Create a new directory
- `$mkdir -p <dir1>/<dir2>/<dir3>` Create nested directories in one step

Note:

To create nested directories in a single step, you can use the `-p` option, otherwise the command assumes that all directories in a path name except the last one already existed.

- `rmdir <Directory name>` Delete empty directories
- `rm -r <Directory name>` Delete the directories and all of its contents

Note:

By default, “rm” does not remove directories. Use the --recursive (`-r` or `-R`) option to remove each listed directory, too, along with all of its contents



Creating and Deleting files

- Using editors like gedit, vim and nano
- `$ touch file1` Create an empty file called file1 in the current directory.
- `$ cat > file1` Create a file called file1 that contain the lines you will write.
[Ctrl+d] Finish waiting for input
- `$ rm file1` Remove file1
- `$ rm -i file1` Request confirmation before deletion

Determining File Content

- To guess at the type of data contained in a file you can use the “file” command
- \$ file /bin/ls executable
- \$ file /etc/passwd ASCII text
- \$ file Desktop directory



Displaying File Content

- `$ cat <file>` Print the whole file contents on screen
- `$ more <file>` Display files on a page-by-page basis.
- `$ less <file>` Similar to the “more” command



Copying Files and Directories

```
$ cp [options] <source_file(s)> <destination>
```

Copying single file

```
$cp file1 file2
```

```
$cp -i file1 file2
```

```
$cp file1 dir1
```

Copying a list of files

```
$cp file1 file2  
dir1
```

Copying directories

```
$cp -r dir1 dir2
```



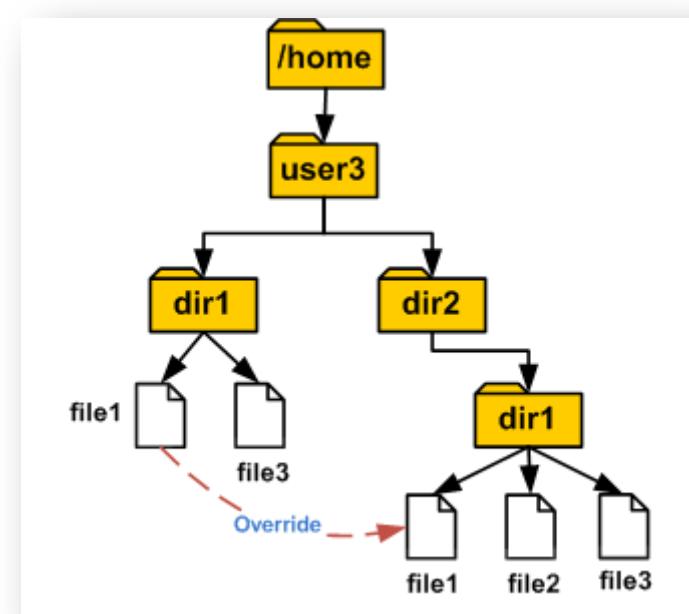
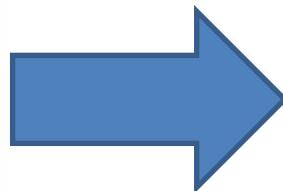
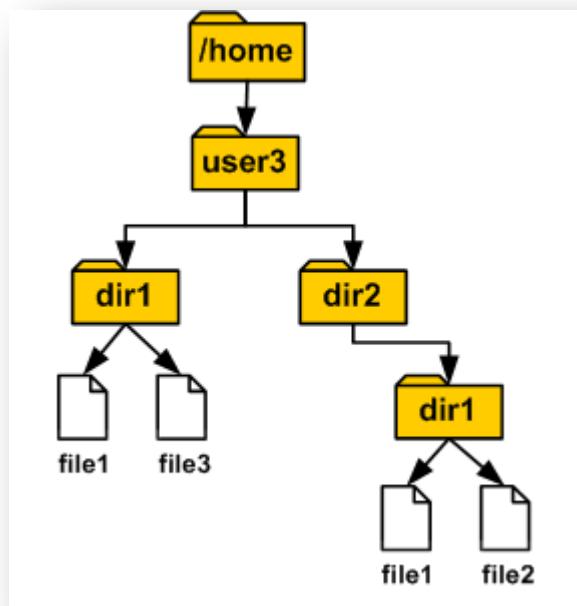


Question

What happen if we apply the following command on the shown file tree ?

```
$ cd /home/user3
```

```
$ cp -r dir1 dir2
```



It will merge the contents of 'dir1' into 'dir2/dir1' and override the original 'dir2/dir1/file1'

Moving and Renaming Files and Directories

```
$ mv [options] <source_file(s)> <destination>
```

- `$mv file1 file2` Rename file1 to file2 (overwriting file2 if existed)
 - `$mv -i file1 file2` **Prompt before overwrite file2 if it already existed**
 - `$mv file1 file2 dir1` Move file1,file2 into dir1
 - `$mv dir1 dir2` **Move dir1 into dir2 if dir2 existed (no need to -r) or rename dir1 to dir2 if dir2 was not existed**



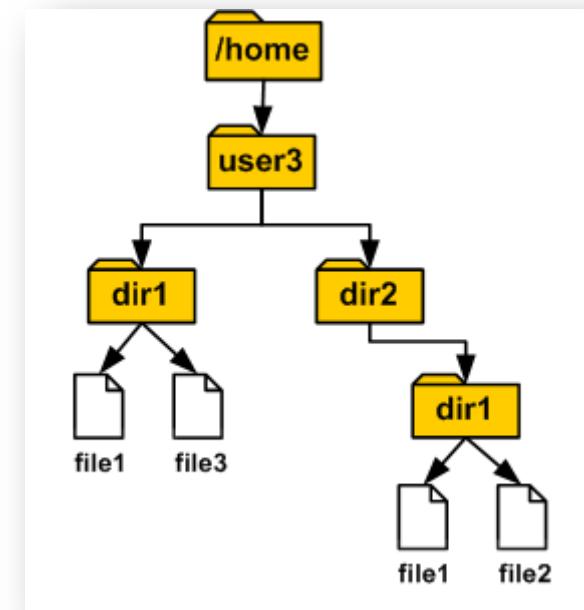
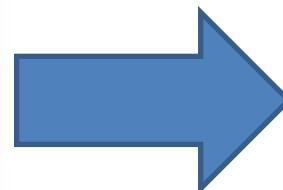
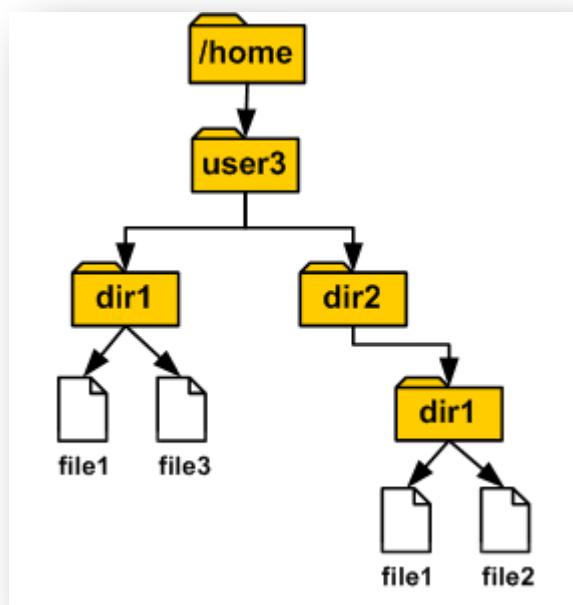


Question

What happen if we apply the following command on the shown file tree ?

```
$ cd /home/user3
```

```
$ mv dir1 dir2
```



Cannot move 'dir1' to 'dir2/dir1': because the Directory 'dir2/dir1' is already exist

Moving a Directory and Its Contents

- When you move a single directory to a target directory that does not exist, that directory is renamed with the target directory name.
- When you move multiple directories to a target directory that does not exist, an error message is displayed:
`mv: target `target directory' is not a directory`

Using Links

Soft Links

Soft Links (Symbolic Links)

- A soft link contains a text string that is a path to another file or directory called the “target”.
- Soft links does not link directly to the inode but to the name of the file (the hard link).
- They can link to files on other devices, as well as on directories (Span across **filesystems**).

Creating Soft Links

\$ ln -s <target path> <link path>

- **\$ ln -s /var/log mylog** Create mylog file as a symbolic link that refer to the directory /var/log
- **\$ ln -s /etc/passwd** Create a soft link “passwd” in the current working directory
- **\$ ls -li**



Soft Links

- Note:
 - The symbolic link is a second file that exists independently of its target (**separate inode**):
 - If a symbolic link is deleted, its target remains unaffected.
 - If a symbolic link points to a target, and sometime later that target is moved, renamed or deleted, the symbolic link will be **broken**.
 - The output of `ls -l` on a symbolic link will show that the type of the file is a soft link (first character = `l`).

Deleting Soft Links

- `$ rm <soft link name>`
- `$ unlink <soft link name>`
- **Important:**
 - Don't use `-r` or `-f` with `rm` when deleting soft links to directories, this will **remove the original target contents.**
 - Don't put the trailing slash “`/`” to the soft link name to directories.



DEMO

Files and directories management
Using Nautilus

LAB

04 - LAB (Files and Directories)



Chapter 5 | Introduction to Users and Groups

- Users & Groups Organization
- Commands to query Users & Groups

Users & Groups

- What is a user ?
- What is a group ?



Users organization

- Every user is assigned a unique User ID number
 - 0 identifies root.
 - > 0 up to 999, for system accounts.
 - From 1000 and higher, for normal accounts.
- Users' names and UIDs are stored in /etc/passwd
- Users are assigned a home directory and a program that is run when they log in (usually a shell).
- Users cannot read, write or execute each others' files without permission.

Groups organization

- Users are assigned to groups, Each group is assigned a unique Group ID number (gid) which are stored in /etc/group.
- Each user is given his own **primary group** and can be add to another **supplementary** groups for additional access.
- All users in a group can share files that belong to the group.

Getting users and groups information

- System → Administration → Users & Groups
 - \$ less /etc/passwd
 - \$ less /etc/shadow
 - \$ id Get info. about the current user
 - \$ id <user> Get info. about the specified user
-
- \$ groups Get info. about the current user groups
 - \$ groups <group> Get info. about the specified group

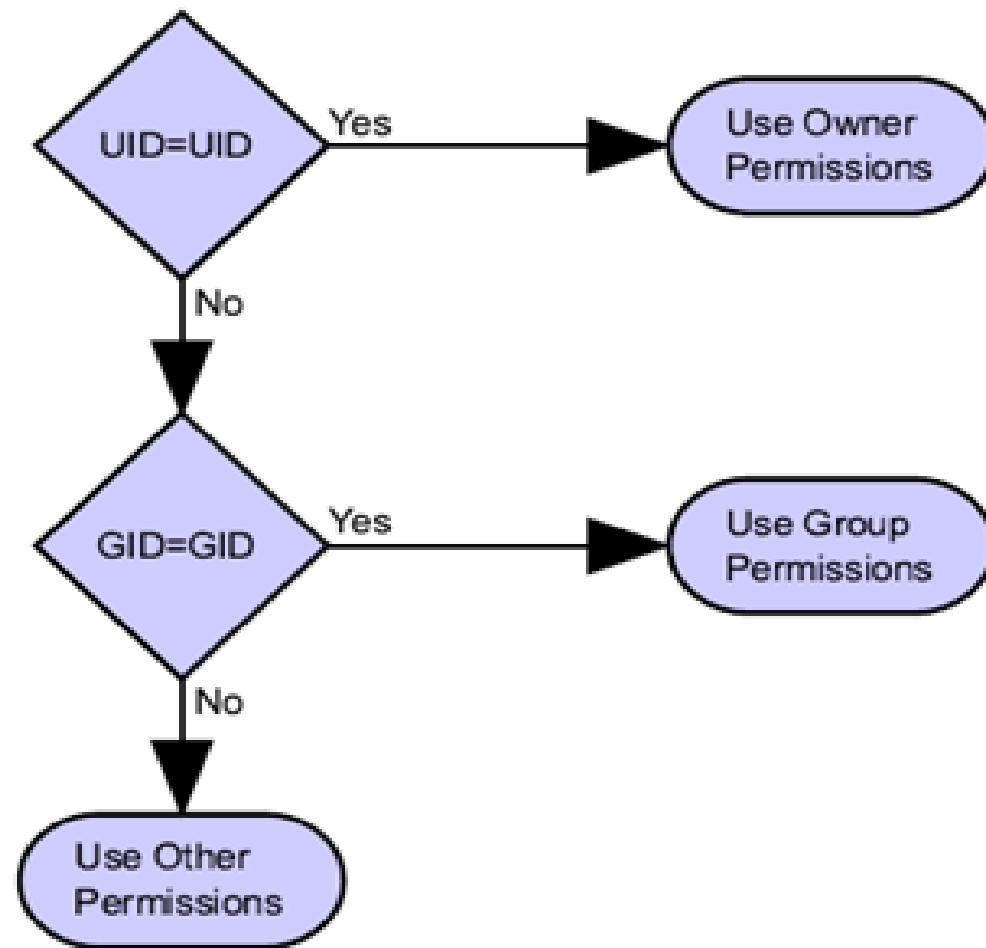
Chapter 6 | Files Security

- Basic files security (UGO model)
- Special permissions
- File Access Control List (ACL)

UGO Permissions concept

- Every file is owned by a UID and a GID.
- Every process runs as a UID and one or more GIDs.
- Usually determined by who runs the process
Three access categories:
 - Processes running with the same UID as the file (**User**).
 - Processes running with the same GID as the file (**Group**).
 - All other processes (**Other**).

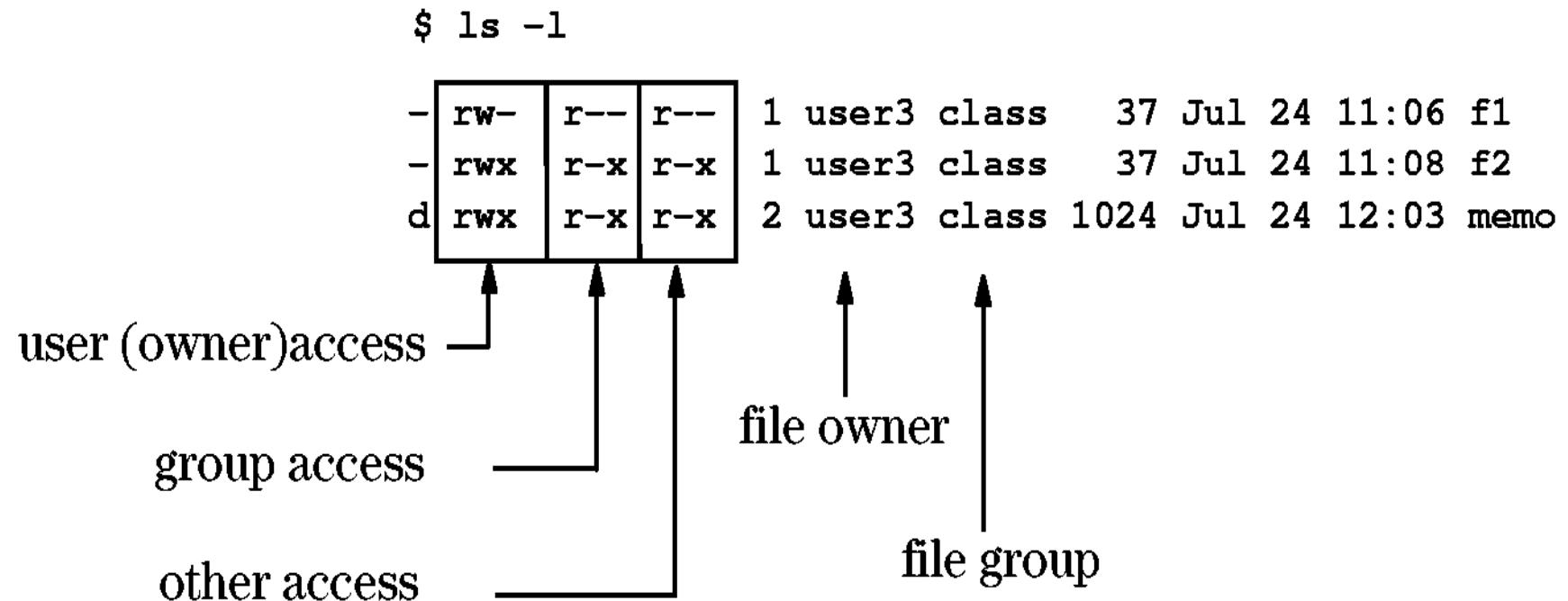
Applying Permissions



Permission Types

Permission	Effect on File	Effect on Directory
r (Read)	Display the contents of a file by (e.g. cat, more)	Display the names of files and directories inside by (e.g. ls)
w (Write)	Change the contents of a file by (e.g. gedit, vim)	Create, delete and change names of files & directory by (e.g. touch, mv and rm)
x (Execute)	Run script by (e.g. sh)	Open the directory by (e.g. cd)

Examining Permissions



Changing Permissions

chmod [options] [mode[,mode...]] filename...

- The mode can be specified in either symbolic mode or octal numeric mode

Changing file permissions using symbolic mode

Permission Set code	Change type code	Permission to modify code
u (Owner)	+ (Add)	r (Read)
g (Group)	- (Remove)	w (Write)
o (Other)	= (Set equal to)	x (Execute)
a (All)		X (Execute only if the file is a directory or already has execute permission)
		s S (SUID or SGID)
		t T (Sticky bit)
		u (Existing owner permission)
		g (Existing group permission)
		o (Existing others permission)
		- nothing

Changing file permissions using symbolic mode

Examples of Symbolic Permissions with *chmod*

Command	Initial Permissions	End Permissions
chmod a+x bigprogram	rw-r--r--	rwxr-xr-x
chmod ug=rw report.tex	r-----	rw-rw----
chmod o-rwx bigprogram	rwxrwxr-x	rwxrwx---
chmod g=u report.tex	rw-r--r--	rw-rw-r--
chmod g-w,o-rw report.tex	rw-rw-rw-	rw-r-----



Changing file permissions using numeric octal mode

Octal number	Octal number
4 (Read)	6 = 4+2 (Read, Write)
2 (Write)	5 = 4+1 (Read, Execute)
1 (Execute)	3 = 2+1 (Write, Execute)
	7 = 4+2+1 (Read, Write, Execute)



Changing file permissions using numeric octal mode

Command	Result
\$ chmod 640 filename \$ ls -l filename	-rw-r----. 1 student student 0 Jan 17 12:51 filename
\$ chmod 000 filename \$ ls -l filename	----- 1 student student 0 Jan 17 12:51 filename

Note:

here we disable all permissions on a file

Notes about permissions

- The delete operation depends on the permissions of the parent directory not the permissions of the file itself .
- **With respect to files**
 - (w) and (x) permissions does not work without (r).
- **With respect to directories**
 - At least both (w) and (x) permissions should be exist to be able to delete from the directory.



Setting permission using symbolic links

- If you change the permission or the owner/group ownerships to a symbolic link, the action will be applied to the original target file not the link file itself.

```
$ touch f1  
$ ln -s f1 ll  
$ id                                show all groups you are a member  
$ chmod a=- ll  
$ chmod u+rwx,go+w ll  
$ chgrp cdrom ll      Change the group ownership to  
                      one of the groups you are a member  
$ ls -l f1;ls -l ll
```

Setting Default Permissions with “umask”

- When creating a new file or a new directory, some default permissions are set.
- These default permissions are determined by the “umask” shell setting.
- This shell setting is applied to all users when login into the system.

Setting default permissions with “umask”

Final Permission = Maximum Permission – Umask value

- Maximum permissions for files = 666
- Maximum permissions for directories = 777

\$ umask [value]

By default:

= **002** for normal users

= **022** for “root” and system users





Setting Default Permissions with “umask”

Example	Result
\$ umask	002 (normal user)
\$ touch file1;mkdir dir1 \$ ls -l	drwxrwxr-x. → dir1 -rw-rw-r--. → file1
# umask	022 (the root user)
# touch file1;mkdir dir1 # ls -l	drwxr-xr-x. → dir1 -rw-r--r--. → file1

For files	For directories
Maximum permissions = 666	Maximum permissions = 777
umask value(for users) = 002	umask value(for users) = 002
umask value (for root) = 022	umask value (for root) = 022
(-) -----	(-) -----
Final Permissions(users) = 664 rw- rw- r--	Final Permissions(users) = 775 rwx rwx r-x
Final Permissions(root) = 644 rw- r-- r--	Final Permissions(root) = 755 rwx r-x r-x

Setting Default Permissions with “umask”

Example	Result
\$ umask 222	222
\$ touch file1;mkdir dir1 \$ ls -l	dr-xr-xr-x. → dir1 -r--r--r--. → file1

For files	For directories
Maximum permissions = 666	Maximum permissions = 777
umask value = 222	umask value = 222
(-) -----	(-) -----
Final Permissions = 444 r-- r-- r--	Final Permissions = 555 r-x r-x r-x

Setting Default Permissions with “umask”

- The way “umask” works in RHEL may be surprising, especially if you’re coming from a different Unix-style environment.
- You **cannot** configure “umask” to allow the automatic creation of new **files automatically with executable permissions**.
- This promotes security: “if fewer files have executable permissions, fewer files are available for a cracker to use to run programs to break through your system”.

Setting Default Permissions with “umask”

Example	Result
\$ umask 111	111
\$ touch file1;mkdir dir1 \$ ls -l	drw-rw-rw-. → dir1 -rw-rw-rw-. → file1

For files	For directories
Maximum permissions = 666	Maximum permissions = 777
umask value = 111	umask value = 111
(-) -----	(-) -----
Expected Permissions = 555 r-x r-x r-x	Expected Permissions = 666 rw- rw- rw-
Applied Permissions = 666 rw-rw-rw-	Applied Permissions = 666 rw- rw- rw-

- No matter what the value of umask, new files in RHEL can no longer be automatically created with executable permissions.
- Use “chmod” to set executable permissions on a specific file.



DEMO

Changing Permissions using Nautilus

The file Ownership

Changing File Ownership

- Only the owner of a file has control over the attributes and access to a file.
- Only “root” can change a file's owner.
- Only “root” or the owner can change a file's group.
- The owner of a file may change the group of the file to any group of which that owner is a member



Changing File Ownership

chown <newowner>[:<newgroup>] <filename>

chown <new owner> {<file>|<dir>}

chown -R <new owner> <dir>

chown <new owner>:<new group > {<file>|<dir>}

chown -R <new owner>:<new group > <dir>



Changing File Ownership

Command	Result
<pre>\$ ls -l filename</pre>	<pre>-rw-rw-r--. 1 student student 0 Jan 17 13:11 filename</pre>
<pre>As the root user execute the command: # chown visitor filename # ls -l filename</pre>	<pre>-rw-rw-r--. 1 visitor student 0 Jan 17 13:11 filename</pre>

Changing File group membership with chgrp

#chgrp <new owner> {<file>|<dir>}

#chgrp -R <new owner> {<dir>}

Command	Result
\$ ls -l filename	-rw-rw-r--. 1 student student 0 Jan 17 13:11 filename
s chgrp visitor filename	
# ls -l filename	-rw-rw-r--. 1 student visitor 0 Jan 17 13:11 filename

The Default Ownership

- When a user creates a file, default ownership is applied:
 - The user who creates the file automatically becomes user owner.
 - The primary group of that user automatically becomes group owner (the group registered in /etc/passwd).

LAB

06 - LAB (The files security)

Special Permissions

SUID – SGID – Sticky Bit

Special Permissions

Special permission	Effect on files	Effect on directories
u+s (suid) or 4 at the first digit (numeric mode)	Execute a program (or an executable) with the rights of the owner, not the rights of the user who execute it.	No effect
g+s (sgid) or 2 at the first digit (numeric mode)	Execute a program (or an executable) with the rights of the group, not the rights of the user who execute it.	Set the group ownership of the newly created files in this directory = the group ownership of the directory rather than the group ownership of the user who create the file.
o+t (sticky) or +t or 1 at the first digit (numeric mode)	No effect	User with write permission on the directory can only remove files that they own, they can't remove files owned by other users.

(setuid) special permission

\$ chmod u+s <file>

\$ chmod 4755 <file>

- Effect on files not directories.
- Appear as “s” or “S” at the location of the execute bit in the owner section in permissions set string when typing “ls -l”.
- If the file is already executable it appears as “Small **s**”.
- If the file is not executable it appears as “Capital **S**”.
- Applying the setuid to an executable mean to execute with the rights of the owner permission, not the caller permission.

(setuid) special permission

Command	Output
# ls -lh /usr/bin/passwd	-rwsr-xr-x 1 root root 59K May 17 2017 /usr/bin/passwd
# ls -lh /etc/shadow	On Debian: -rw-r----- 1 root shadow 1.5K Jun 6 11:48 /etc/shadow
	On Redhat: -----. 1 root root 1042 Feb 26 11:02 /etc/shadow
\$ ls -l /bin/su	-rwsr-xr-x 1 root root 40536 May 17 2017 /bin/su



(setgid) special permission

\$ chmod g+s {<file> | <dir> }

\$ chmod 2755 {<file> | <dir>}

- Effect on both files and directories.
- Appear as “s” or “S” at the location of the execute bit in the group section in permissions set string when typing “ls -l”.
- If the file/directory is already executable it appears as “Small **s**”.
- If the file/directory is not executable it appears as “Capital **S**”.
- Applying the setgid to an executable file means to execute with the rights of the group permission, not the caller permission.
- Applying the setgid to a directory means that the **newly create files** in that directory will take the group ownership as the directory not as the creator.

(setgid) special permission

- \$ su –
- # groupadd guest
- # mkdir /shared_folder
- # chmod 775 /shared_folder
- # chgrp guest /shared_folder
- # usermod -aG guest student
- # usermod -aG guest visitor
- # chmod g+s /shared_folder

(sticky bit) special permission

\$ chmod o+t <dir>

\$ chmod 1755 <dir>

- Effect only on directories.
- Appear as “t” or “T” at the location of the execute bit in the other section in permissions set string when typing “ls -l”.
- If the directory is already executable it appears as “Small t”.
- If the file/directory is not executable it appears as “Capital T”.
- Applying the sticky bit on a directory means that users can’t change or delete other files owned by different users, only they can modify and delete their own files.

(sticky bit) special permission

- # chmod o+t /shared_folder

Or

- # chmod 1775 /shared_folder

Special Permissions

- You must be **extremely careful** when you set special permissions, because special permissions constitute a security risk.



LAB

06 - LAB (Special Permissions)

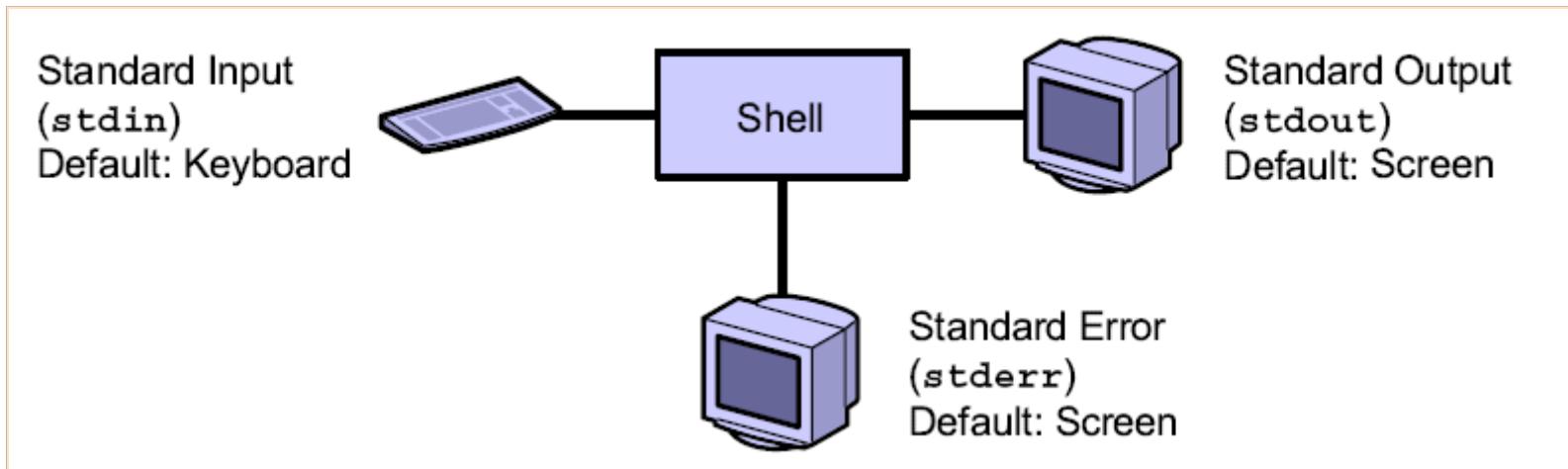
Chapter 7 | The BASH shell

- Input and Output redirection
- Using the BASH shell

Input and Output Redirection

Input and Output Redirection

- Linux provides three I/O channels to programs:



Channel Number	Stream	
0	STDIN	Standard input (keyboard by default).
1	STDOUT	Standard output (terminal window by Default).
2	STDERR	Standard error (terminal window by default).



Input and Output Redirection

- STDOUT and STDERR can be redirected to files:

command <operator> <filename>

- Supported operators include:

>	Redirect STDOUT to file and overwrite the previous file contents if the file exists.
>>	Redirect STDOUT to file, and append to the file if the file exist
2>	Redirect STDERR to file
&>	Redirect all output to file
<	Redirect STDIN from a file instead of the keyboard

Redirecting program output to a File

Command	Result
\$ cat <write some text> press ctrl+d to exit	Here, the cat command read from the default standard input (Keyboard) and redirect the output to the default standard output (Screen).
\$ cat > file1 <write some text> press ctrl+d to exit	Here, the cat command read from the default standard input (Keyboard) and redirect the output to file1.
\$ cat >> file1 <write some text> press ctrl+d to exit	Here, the cat command read from the default standard input (Keyboard) and append the output to file1.
\$ > file1	Clear all the contents of file1

Combining output and errors

Command	Result
\$ ls -l file1 fackfile	Show both the output and error message on the screen.
\$ ls -l file1 fackfile > out.txt	Save the output to the file out.txt and show the error message on screen.
\$ ls -l file1 fackfile > out.txt 2>err.txt	Save the output to the file out.txt and save the error message to err.txt. Note: Here, the number 2 is the error channel number.
\$ ls -l file1 fackfile > out.txt 2>&1 or \$ ls -l file1 fackfile &> out.txt	Save the output and error to the same file out.txt.
\$ ls -l file1 fackfile > out.txt 2>/dev/null	Save the output to the file out.txt and suppress errors.



Redirecting program input from a file

Command	Result
\$ cat < file1	Here, the cat command read from file1 and redirect the output to the default standard output (Screen).
\$ mail student	Here the mail command read the input from keyboard and send the entered data (subject & Message) to user student
\$ mail student < file1 Press ctrl+d when finish	Here, send the contents of file1 to user student
\$ tr 'A-Z' 'a-z' <Enter some text> Press ctrl+d when finish	Here the tr command read from the keyboard and replace any lowercase letter to the uppercase one and show the output on the screen
\$ tr 'A-Z' 'a-z' < /etc/passwd	Here, the tr command read the file /etc/passwd and replace any lowercase letter to the uppercase one and show the output on the screen Note: The original /etc/password did not changed.

Redirecting program output to another program (Piping)

- Linux offers a way of linking commands directly via pipes: A program's output automatically becomes another program's input.

<command1> | <command2>

Redirecting program output to another program (Piping)

Without Piping	With Piping
\$ ls -laF > tempfile \$ less tempfile	\$ ls -laF less

- Instead of first redirecting the output of “ls -laF” to a file and then looking at that file using less, you can do the same thing in one step without an intermediate file.

Redirecting program output to another program (Piping)

Command	Result
\$ cat /etc/passwd wc -l	Display the contents of the /etc/passwd file using the cat command, then send the output to the standard input channel of the wc command which count the number of lines sent to it.
\$ cat /etc/passwd grep root	Display the contents of the /etc/passwd file using the cat command, then send the output to the standard input channel of the grep command, this will filter the result so showing only the lines containing the text “root”.

The **BASH** Shell

Using the Bourne Again Shell

Command Editing Tricks

Key stroke	Result
Home or CTRL+a	Moves to beginning of line
End or CTRL+e	Moves to end of line
CTRL+l (the small character of L)	Clear the screen
CTRL+u	deletes to beginning of line
CTRL+k	deletes to end of line
CTRL+arrow	moves left or right by word
CTRL+t	Swap the two characters in front of and under the cursor
CTRL+c	Interrupt a command
CTRL+d	End the input (for login shells: log off)

The Tab Key completion facility

Command	Result
\$ nau<tab>	\$ nautilus
\$ ls /h<tab>/u<tab>	\$ ls /home/user1
\$ ls<tab><tab>	ls lscgroup lsinitrd lspci lssubsys lsattr lscpu lsmod lspcmcia lsusb lsb_release lshal lsof lss16toppm



The History

Command	Result
\$ history	View all previous commands 1 su - 2 ls 3 clear 4 pwd 5 mkdir dir2 6 ls dir2 dir1 7 cp -r dir1 dir2 8 ls dir2 9 history
\$!2	Run the command number 2
\$!mk	Run the last command that began with the letters "mk" (here, command number 5)
\$ Esc .	Esc followed by dot will recall the last written command from history
\$ Alt+.	Recall the last written command from history

Dealing with file names with special characters

\$ & ; () { } [] * ? ~ < > ! “ ‘

- “\” to escape a single special character
- ‘....’ or “.....” to escape several special characters

Command	Result
\$ touch new file \$ ls -g	Create 2 file: -rw-rw-r--. 1 student 0 Nov 29 13:19 file -rw-rw-r--. 1 student 0 Nov 29 13:19 new
\$ touch new\ file \$ ls -g	Create single file -rw-rw-r--. 1 student 0 Nov 29 13:21 new file
\$ touch 'new file' \$ touch "new file" \$ ls -g	Create single file -rw-rw-r--. 1 student 0 Nov 29 13:21 new file
\$ touch file\;_\"(1\") \$ touch 'file;_\"(1")' \$ ls -g	Create single file -rw-rw-r--. 1 student 0 Nov 29 13:23 file;_\"(1")

Writing a command on more than one line

Command	Result
\$cp -r \ /home/student/dir1 \ /home/student/dir2/	Copy dir1 inside dir2

Multiple commands on one line

Command	Result
\$ echo Today is; date	Today is Fri 5 Dec 12:12:47 CET 2008 Note: The second command will be executed once the first is done.

Command aliasing

Command	Result
\$ alias dir="ls" \$ dir	list files and directories like ls
\$ alias rm='rm -i' \$ rm file1	rm: remove regular empty file 'file1'? y Note: The -i switch prompts the user for confirmation before a file is deleted or overwritten with the rm command.
\$ alias rm='mv -t ~/.local/share/Trash/files' \$ rm file1	Instead of removing file1, it will move it to the GNOME trash for the user
\$ alias p='vim /etc/passwd' \$ p	Open /etc/passwd in the vim editor
\$ unalias <alias name>	Remove the alias

Making aliases permanent

- Aliases remain in effect only during the current login session (i.e., until the user logs out or the computer is shut down).
- To keep your aliases permanent:
 - Write your aliases in **.bashrc** file inside your home directory **or**
 - Edit the system-wide **/etc/bashrc** for all users.

Note:

The system needs to be restarted before system-wide aliases can take effect

Command exit status

Command	Result
ls \$ echo \$?	0 Note: 0 = no errors occurred in the last command (ls).
\$ ls non_existed_file	ls: cannot access non_existed_file: No such file or directory
\$ echo \$?	2 Note: 2 = errors occurred in the last command (ls).

Conditional Execution

Command	Result
\$ ls \$ ls file1 && echo ok	file1 file1 ok Note: The echo command executed because the ls command returned with no error.
\$ ls \$ ls file2 && echo ok	ls: cannot access file2: No such file or directory file1 Note: The echo command did not execute because the ls command returned with an error.
\$ ls \$ ls file1 echo ok	file1 file1 Note: The echo command did not execute because the ls command returned with no errors
\$ ls \$ ls file2 echo ok	file1 ls: cannot access file2: No such file or directory ok Note: The echo command executed because the ls command returned with an error.

The BASH Shell

The command-line expansion

The command-line expansion

- Wildcards expansion.
- Characters expansion.
- Braces expansion.
- The tilde expansion (~).
- Command expansion.



Wildcards expansion

- * Match one or more characters
- ? Match a single character

Command	Result
\$ ls prog1/p*.c	Match the following files: prog1/p1.c prog1/polly.c prog1/pop-rock.c prog1/p.c
\$ ls *	Match all files and directories in the current directory except the hidden.
\$ ls .*	Match all hidden files and directories
\$ ls p?.c	Match the following files: p1.c pa.c p-.c p..c

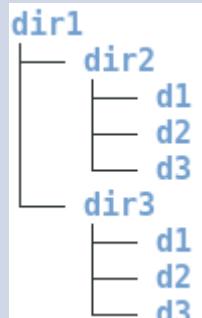


Characters expansion

[] Match exactly those characters that are enumerated within them

Command	Result
\$ ls file[124]	Only match: file1 file2 File3
\$ ls file[1-9]	Match the range from file1 to file9
\$ ls [A-Z]*.c	Match any file with the first character is capital letter and end with .c
\$ ls file[!124] \$ ls file[^124]	Match any file its name begin with the word file and end with one character not 1 nor 2 nor 4

Braces expansion

Command	Result
\$ ls {red,yellow,blue}.txt	Matches: red.txt yellow.txt blue.txt
\$ touch file{a,b,c}{1,2,3}	Create: filea1 filea2 filea3 fileb1 fileb2 fileb3 filec1 filec2 filec3
Note: You can have more than one brace expression in a word, which will result in the cartesian product (all possible combinations)	
\$ mkdir -p dir1/{dir2,dir3}/{d1,d2,d3} \$ tree dir1	

The tilde expansion (~)

The tilde character “~” in bash has two meanings:

1. Refer to your home directory.
2. Refer to the home directory for another user.

Command	Result
\$ cd ~ \$ pwd	/home/student
\$ cd ~user1 \$ pwd	/home/user1

Command expansion (Substitution)

Command substitution is a facility that allows a command to be run and its output to be pasted back on the command line as arguments to another command.

Command	Result
\$ echo "today is \$(date)" \$ echo "today is " `date`	today is Thu Nov 29 17:13:01 EET 2012 Note: Here we insert the date command between 2 back quotes (`date`)

Preventing Expansion

- Preventing expansion using the backslash: \
- Preventing expansion using the quoting characters: ' and "

Preventing expansion using the backslash

Backslash (\) before a special character cancels the special meaning of that character and interpret the next character as a literal

Command	Result
\$ mkdir -p dir1\{dir2\} \$ tree dir1	dir1 └─ {dir2}

Note: The special meaning of the characters { and } did not work, because we precede both letters by a backslash (\) which prevent the braces expansion, so the resulting directory name {dir2} includes { and } as parts of its name.

Preventing expansion using the quoting characters

The single forward quotation (')

Inhibit all expansion

The double quotation (")

Inhibit all expansion, except \$, ` , \ and !

Command	Result
\$ echo '\$af\;[]' \$ echo '\$SHELL'	\$af\;[] \$SHELL
\$ echo "current directory is \$(pwd)" \$ echo "\$SHELL" \$ echo	current directory is /home/student /bin/bash Note: SHELL is an environment variable

Shell variables

- Like most common shells, bash has features otherwise found in programming languages.

Command	Result
\$ x=linux Note: Take care not to insert spaces in front of or behind the equals sign	Create a variable called x and initialize it with the value "linux"
\$ echo \$x	linux
Note: Use \$ before the variable name when you need to access its value	
\$ today=`date` Note: Here we used the command substitution (`date`), otherwise the string "date" is used, not the date command	Store the output of the date command into a variable called "today"
\$ echo "today is \$today"	today is Tue Jun 5 11:58:29 EET 2018
\$ unset x \$ unset today	Remove the shell variable "x" Remove the shell variable "today"

Shell variables

- By default, shell variables are only visible in the shell in which they have been defined.

Command	Result
\$ x=linux	Create local variable x with value "linux"
\$ bash	Start a child bash shell
\$ echo \$x	Here the child shell know nothing about x
\$ exit	Return to the first shell
\$ echo \$x	Here echo print "linux" since x still defined

Environment variables

- Environment variables are shell variables control various aspects of the operation of the shell itself.
- Unlike shell variables, environment variables are inherited by child shells

Command	Result
\$ echo \$HOSTNAME	Display the value of the HOSTNAME environment variable
\$ bash	Display the value of the HOSTNAME environment variable
\$ echo \$HOSTNAME	Here the child shell know about HOSTNAME too

Environment variables

- All the environment variables of a shell are also shell variables but not vice versa.
- Using the export command, you can declare an existing shell variable as an environment variable

Command	Result
\$ x=one	Create local variable x with value "one".
\$ export x	Make x available to child shells
\$ bash \$ echo \$x	Here the child shell know about x and prints "one".
\$ export y="two"	Create and export the variable y at the same time.
\$ bash \$ echo \$y	Here the child shell know about y and prints "two"
\$ unset y	Remove y from the child shell only.
\$ exit \$ echo y	Return to the first shell. The first shell still know about y and prints "two".

Some important environment variables

Command	Result
\$ env \$ export	print all the current environment variables
PWD	Name of the current working directory
PS1	Shell command prompt appearance
PS2	Shell secondary prompt appearance
HOME	Current user's home directory
USER	Current user's user name
UID	Current user's user ID
PATH	List of directories containing executable programs that are eligible as external commands

Some important environment variables

(The PATH environment variable)

- The shell distinguishes internal and external commands.
- External commands correspond to executable programs, which the shell looks for in specified directories mentioned in the value of the **PATH** environment variable.
- \$echo \$PATH

/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/sbin:/home/student/bin

Some important environment (The PATH environment variable)

\$ echo \$PATH

/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/sbin:/home/student/bin

\$ ls (what happens ??)

“ls” is not an internal command, so search the following directories in order:

/usr/local/bin/ls	not found
/bin/ls	yes, it is there
/usr/bin/	don't check
/usr/local/sbin	don't check
/usr/sbin	don't check
/sbin	don't check
/home/student/bin	don't check

Login and non-login shells

- The login shell is any shell created at login including X login, like the normal graphical login or using the virtual consoles terminals or changing the user identity with the hyphen option.
- The non-login shell is a shell like, graphical terminals or when changing the user identity without the hyphen option or any other bash instance.

Login and non-login shells

Login SHELL	Non-Login SHELL
- Normal graphical login - Virtual Consoles	- Graphical terminals
\$ bash --login \$ bash -l	\$ bash \$ bash <script file>
\$ su - \$ su - <user>	\$ su \$ su <user>

Bash startup files

- When a user do a successful login to a bash shell, he will find environment variables and aliases already there. Where are these variable come from?!
- It comes from running “**bash startup files**” automatically, when a user starts a bash instance.

Bash startup profile files (Personal Initialization Files)

- Each time a new user was created, there are three files would be copied to his home directory from **/etc/skel**.

File name	Copy location	Purpose
/etc/skel/.bash_profile	~/.bash_profile	<ul style="list-style-type: none">• Run each time the user starts a login shell• Used for Setting <u>user specific</u> environment variables.
/etc/skel/.bashrc	~/.bashrc	<ul style="list-style-type: none">• Run for all shells (login/non-login)• Used for:<ul style="list-style-type: none">○ Setting user specific local variables○ Defining user specific aliases
/etc/skel/.bash_logout	~/.bash_logout	<ul style="list-style-type: none">• Run each time the user logout from a login shell

Bash startup system-wide files

- The shell startup files in the /etc directory generally provide global settings (Affect all users).

File name	Purpose
/etc/ profile	<ul style="list-style-type: none">• System-wide initialization file, executed for login shells.
/etc/ bashrc	<ul style="list-style-type: none">• System-wide functions and aliases.• This file is processed when subshells (non-login) are started.

Bash startup files execution order

- The order of execution varies according to the shell login status (login shell or non-login shell)

Execution for Login Shell	Execution for Non-Login Shell
1) First, reads and executes commands from /etc/profile , if that file exists.	1) Reads and executes commands from ~/.bashrc , if that file exists, which in turn at first read and execute commands from /etc/bashrc
2) Then, looks for ~/.bash_profile , ./bash_login , and ~/.profile , in that order, and reads and executes commands from the first one that exists and is readable, which in turn at first read and execute commands from ~/.bashrc	
3) Reads and executes commands from the file ~/.bash_logout , if it exists	

The **--noprofile** option may be used when the shell is started to inhibit this behavior

Bash startup files execution order

For login Shell

1. Reads and execute `/etc/profile`
2. Reads and execute `~/.bash_profile`
3. Reads and execute `~/.bashrc`
 4. Reads and execute `/etc/bashrc`
5. Execute specific `~/.bashrc` commands
6. Execute specific `~/.bash_profile` commands

For non login Shell

1. Reads and execute `~/.bashrc`
2. Reads and execute `/etc/bashrc`
3. Execute specific `~/.bashrc` commands

The commands “which” and “whereis”

- Which <command>
- Whereis <command>

LAB

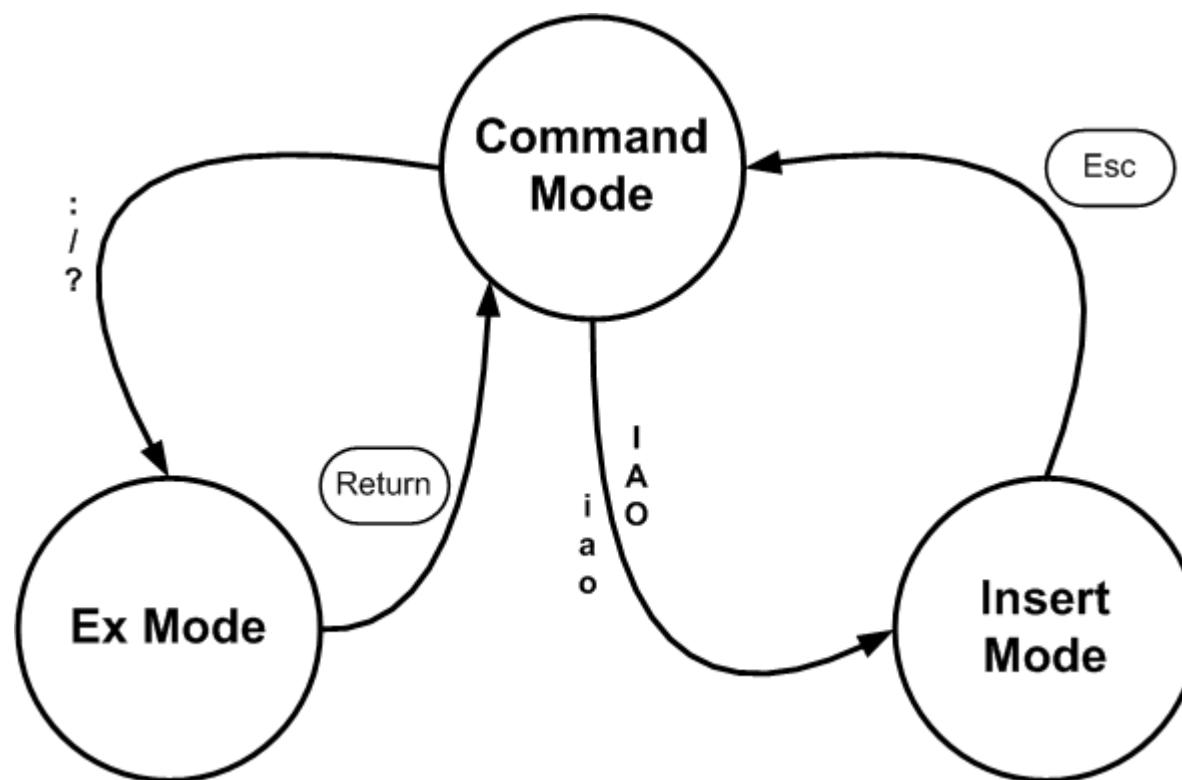
07- LAB (Using the Bash shell)

Chapter 8 | The VIM editor

- Using the VIM editor

VIM modes

- At any given moment, “vim” is running in one of three modes:



Saving & Exiting

- To save the file : Esc → :w
- To exit from vim : Esc → : → q
- To exit from vim and save : Esc → : → wq or ZZ or x

Note:

- Pressing <ESC> will place you in Normal mode or will cancel an unwanted and partially completed command.

Character navigation

Command



or



Function

Move up one line.



or



Move down one line.



or



or **Space**

Move right one character.



or



or **Backspace**

Move left one character.

Character navigation

Command

3



or
3



Function

Move up 3 lines.

5



or
5



Move down 5 lines.

10



or
or
10 Space

Move right 10 characters.

6



or
6



Move left 6 characters.

Word navigation

Command



Function

Move forward one word.



Move backward one word.



Move to the end of the next word.

Word navigation

Command

2



Function

Move forward 2 words.

3



Move backward 3 words.

4



Move to the end of the fourth word.

Line navigation

Command



or



Function

Go to the end of the current line.



or



or



Go to the beginning of the current line.



Go to the end of the file.

Note: #G = Go to the line number #.



#

Go to the line number #.

Window navigation

Command



+



or



Function

Scroll back to previous window of text.



+



or



Scroll forward to next window of text.



+



Scroll up half a window of text.



+



Scroll down half a window of text.

Screen navigation

Command



Function

Go to the last line on the screen.



Go to the middle line on the screen.



Go home (First line, first character)

Text selection

Command



+



Function

Line selection visual block.



+



Characters selection visual block.

Adding text

<u>Command</u>	<u>Function</u>
a	(append) after the current letter
A	(append) at the end of line
i	(insert) before the current letter
I	(insert) at the beginning of the line
o	(add new line) after current line
O	(add new line) before current line

Deleting text

<u>Command</u>	<u>Function</u>
x	Delete one letter.
D or d\$ or d+end	Delete to the end of line.
db	Delete to the start of current word.
de	Delete to the end of current word .
dd	Delete current line.
dw	Delete to the start of next word.
dh	Delete from the first line in screen.
u	Undo last changes - continues.
U	Undo in the current line.

Copying and moving text

<u>Command</u>	<u>Function</u>
d object (object = b,e,d,w,h)	Delete object into the cut buffer.
p (lowercase)	Paste contents of the cut buffer back into the text after the cursor.
P (uppercase)	Paste contents of the cut buffer back into the text before the cursor.
J (uppercase)	Join tow lines in text.

Changing text

<u>Command</u>	<u>Function</u>
r(character)	Replaces the character at the current cursor position with the named character.
R	Replaces all characters
c <i>object</i> (object=w,e,b)	Changes the named object.

Searching text

<u>Command</u>	<u>Function</u>
/text	Search for text from the current line towards the end of the file, with wrap around.
?text	Search for text from the current line towards the beginning of the file.
n	Find the next searched for text, in the same direction.
N	Find the previously searched for text, in the reverse direction.

VIM tutorial

- You can learn about vim through an interesting tutorial by executing the command “vimtutor”

```
$ vimtutor
```

Chapter 9 | Basic Tools

- Finding Files
- Using tar
- Working with text files

Finding Files

find – locate – whereis – which

The find command

find [Starting directory path(s) ...] [Expression]

One or more directory paths to begin the search from

The criteria used to match the files.
It may consist of:

1. Tests,
2. Operators,
3. Options,
4. Actions.

Tests

Test	Description
-name	Specifies a file name pattern.
\$ find . -name Desk*	Find all files in the current directory that its name begin with the letters "Desk"
-iname	Like (name) and also ignores case differences.
-type	Specifies a file type: <ol style="list-style-type: none">1. d directory2. f plain file3. l symbolic link
\$ find . -type d	Find all directories in the current directory
-user	Specifies a user that the file must belong to.
\$ find . -user student \$ find . -user 500	Find all files that the user "student" owns in the current directory

Tests

Test	Description
-group \$ find . -group student \$ find . -group 500	Specifies a group that the file must belong Find all files that the group "student" owns in the current directory
-size \$ find . -size +10k	Specifies a particular file size. Plain numbers signify 512-byte blocks; bytes or kilobytes can be given by appending c or k, respectively. A preceding plus or minus sign stands for a lower or upper limit; -size +10k, for example, matches all files bigger than 10 KB.

Operators

Operator	Description
!	(not) The following test must not match
a	(and) Both tests to the left and right of -a must match
o	(or) At least one of the tests to the left and right of -o must match

If multiple tests are given at the same time, they are implicitly ANDed together all of them must match.

Operators

Test	Description
\$ find . -name 'dir*' -type d \$ find . -name 'dir*' -and -type d \$ find . -name 'dir*' -a -type d	Find all directories in the current directory that the first three characters of its name is "dir"
\$ find . \(` -name 'dir*' -o -type d `)	Find all files in the current directory that the first three characters of its name is "dir" or its type is a directory
\$ find . \(` -name 'dir*' -or -type d `)	Note: In order to avoid mistakes when evaluating logical operators, the tests are best enclosed in parentheses
\$ find . ! -name 'dir*'	Find all files in the current directory that its name does not begin with the 3 letters 'dir'

Actions

Actions perform a specific action to each returned file by the file command

Action	Description
-print	The search results can be displayed on the screen. Note: this is the default action
-ls	list current file in “ls -dils” format on standard output
-exec	execute a command on the current returned file by the search process
-ok	Like (-exec) but it asks the user for confirmation before actually executing the command

Actions

Test	Description
\$ find . -user student -exec ls -l '{}' \\;	<p>searches for all files within the current directory (and below) belonging to user "student", and executes the "ls -l" command for each of the returned files</p> <p>Note:</p> <ol style="list-style-type: none">1. The command following -exec must be terminated with a semicolon (";").2. Since the semicolon is a special character in most shells, it must be escaped (\;)3. Two braces ("{}") within the command are replaced by the file name that was found.4. It is best to enclose the braces in quotes to avoid problems with spaces in file names

Actions

Test	Description
\$ find . -atime +13 -exec rm '{}' \\;	Deletes all files within the current directory (and below) that have not been accessed for two weeks
\$ find . -atime +13 -ok rm '{}' \\;	Deletes all files within the current directory (and below) that have not been accessed for two weeks and confirm before actual deletion
\$ find . -name "*.html" -type f -exec chmod 644 '{}' \\;	This command searches through the current directory for files that end with the extension .html. When these files are found, their permission is changed to mode 644 (rw-r--r--).
\$ find -name "*.conf" -exec cp '{}' '{}'.orig \\;	Backup all .conf files in the current directory by copying them to new files with the same name but with .orig extension

The locate command

- The find command needs to walk the complete directory tree below the starting directory (may take considerable time).
- The locate command locate is more fast because it does not walk the file system tree, but checks a “**database**” of file names that must have been previously created using the “updatedb” program run by administrator.

The locate command

Command	Result
\$ locate /passwd	Locates all file names that contain the pattern "/passwd" anywhere in its path name
\$ locate "*/passwd"	get all files contain only "/passwd" at the end of its full path name Note: A query with wildcard characters returns only those names which the pattern describes completely—from beginning to end. Therefore pattern queries to locate command usually start with "*"
\$ locate tc/pass	/etc/passwd /etc/passwd- Note: The slash ("/") is not handled specially



The whereis command

- The “whereis” utility searches for files in a restricted set of locations, such as standard binary file directories, library directories, and man page directories.
- This tool does not search user directories or many other locations that are easily searched by find or locate.
- The “whereis” utility is a quick way to find program executables and related files like documentation or configuration files.

The whereis & which commands

Command	Result
\$ whereis ls	ls: /bin/ls /usr/share/man/man1/ls.1.bz2 The result shows both the ls executable (/bin/ls) and the ls man page
\$ which date	/usr/bin/date The result shows the full path of the shell command “date”

LAB

10- LAB (Finding Files)

Using the “tar” tools

The tar tool

Command	Result
\$ tar cvf archive_name.tar dirname/	Create a new uncompressed archive archive_name.tar from “dirname” directory -c create a new archive -v verbosely list files which are processed. -f following is the archive file name
\$ tar cvzf archive_name.tar.gz dirname/	Creating a gzipped tar archive
\$ tar cvjf archive_name.tar.bz2 dirname/	Creating a bzipped tar archive
\$ tar xvf archive_name.tar \$ tar xvfz archive_name.tar.gz \$ tar xvzf archive_name.tar.bz2	Extract the archive -x extract files from archive
\$ tar tvf archive_name.tar \$ tar tvfz archive_name.tar.gz \$ tar tvzf archive_name.tar.bz2	View the tar archive file content without extracting



Working with text files

- Using common text tools
- Using "Regular Expressions"



Using common text tools

- less - cat - tac
- head - tail
- cut – sort - uniq
- wc
- grep



Using common text tools

Work on files	Work through Pipe	Notes
\$ less <file>	\$ <command> less	Pager for long text files /<text> Search forward ?<text> Search backward n find next N find previous m<letter> Mark position '<letter> Go to mark position " Go to previous mark
\$ cat <file> \$ cat -n <file> \$ cat -b <file> \$ cat -s <file> \$ cat <f1> <f2>	\$ <command> cat	dumps the contents of a file to the screen -n Number all output lines -b Number nonempty output lines -s suppress repeated empty output lines
\$ head <file> \$ head -n 5 <file>	\$ <command> head	output the first part of files (10 line by default)
\$ tail <file> \$ tail -n 5 <file> \$ tail -f <file>	\$ <command> tail \$ <command> tail -5	output the last part of files (10 line by default) -f output appended data as the file grows

Using common text tools

Work on files	Work through Pipe	Notes
\$ cut -d : -f 1 /etc/passwd	\$ <command> cut	Print selected parts of lines from file
\$ sort /etc/passwd	\$ <command> sort	Sort lines of text files -r Reverse sort -n Numeric sort -k <filed number> Sort by filed -t <sparator> Field separator
\$ uniq <file>	\$ <command> uniq \$ sort <file> uniq	report or omit repeated lines -c Count number of occurrences -d Only print duplicate lines -u Only print unique lines
\$ wc <filename>	\$ <command> wc	Count lines, words and characters of a file
\$ grep <pattern> <file>	\$ <command> grep <pattern>	Print lines matching a pattern -i Ignore case -v Invert match -w match a complete word



Using Regular Expressions (Regex or REs)

What is Regular Expressions (regex)

- **\$ man 7 regex**
- A regular expression is a special text strings that are used to search for and match patterns on text.
- “regex” is not user friendly, but they allow complex conditions to be expressed precisely.
- Many tools use “regex” (i.e. man, vim, less, grep)

Most Significant Basic Regular Expressions

Expression	Definition	Regex Example	Example match
d	Literal: The letter “d”	dog	dog dogma slimdogs
*	Modifier: Zero or more of the previous character	hel*o	hello Theophilus helllllllllo
.	Wildcard: “the dot” = any single character	test.txt	test.txt test0txt.mp3 mytest!txt
[]	Wildcard: any single character in the set	file[1348]	file1 file3 file4.txt somefile8

Most Significant Regular Expressions

Expression	Definition	Regex Example	Example match
[^]	Wildcard: any single character not in the set	file[^0123456789]	file a file A file b .txt some file %
^	Anchor: Line begin with	^Test	Test Test ing 1,2,3,4
\$	Anchor: Line ends with	test\$	test some text ends with test
.*	Combination of . (Any character) plus * (zero or more)	^Test.*123	Testing 123 4 Test 123 .txt Test this has 123 456
\	Treat the next character as literal	test\.txt\$	test.txt some test.txt
^\\$	Match blank lines		

Regex in man pages

- The man pages can be searched using regex

Try:

\$ man passwd

- /[^]passwd
- /exit
- /.^{*}pass

Using grep with regex

- The grep command “**General Regular Expression Parser**” uses Regex.

\$ grep <regex> <file>[<file>]...

\$ <command> | grep <regex>

Options:

- **-n** Print the line number of the matched line
- **-i** Ignore case
- **-v** Invert the selection
- **-e** Specify more than one regex (use multiple -e)
- **-w** Match the whole word only
- **--color**

Using grep with regex

Regex Example

```
$ grep 'root' /etc/passwd
```

```
$ grep '^root' /etc/passwd
```

```
$ grep ':/bin/bash' /etc/passwd
```

```
$ grep --color=auto ':/home/.*' /etc/passwd
```

```
$ grep -v 'nologin$' /etc/passwd
```

```
$ grep '^sS]tudent' /etc/passwd
```

- When using regex it is a good idea to use escaping (surrounding by quotes) to prevent the regex from being interpreted by the shell.
- In many cases, it is not really necessary to use escaping; in some cases, the regular expression fails without escaping.

Regex VS Filename globbing characters (wildcards)

- Regex and file name globbing are two very different things.
- Regex are used in commands for pattern matching in text.
- File name globbing is used by shells for matching file and directory names using wildcards.
- A common mistake is to forget this and get their functions mixed up.

Chapter 10 | Process Management

- Managing Shell Jobs
- Managing and Monitoring Processes

What is a Process?

- A program (a set of instructions loaded into memory) that is being executed is called a “process”.
- Example:
 1. When you log in and start the shell, you start a process.
 2. When you execute a command or when you open an application, you start a process.
 3. When a process starts another program, the new process is called its child process. The original process is the parent process.
 4. Child processes inherit characteristics from its parent.
 5. Child processes can have their own children. When a parent process exits, all of its children processes also exit.

The shell job control

- The shell offers a feature called job control which allows easy handling of multiple processes

The shell job control

Command	Effect
\$ sleep 400	Start a new foreground process that finish after 400 seconds
\$ CTRL+z	Suspend (stop, but not quit) a process running in the foreground and places it in the background as a stopped job [1]+ Stopped sleep 400
\$ jobs	display status of jobs in the current session [1]+ Stopped sleep 400
\$ bg %1	Reactivate a suspended program (with number=1) in the background
\$ jobs	[1]+ Running sleep 400 & Note: the & sign at the end of the command
\$ sleep 300 &	Run this command in the background (release the terminal)
\$ jobs	[1]- Running sleep 400 & [2]+ Running sleep 300 &
\$ fg %1	Reactivate a background program (with number=1) to work in foreground again and occupy the shell
\$ CTRL+c	Interrupt (terminate and quit) a process running in the foreground
\$ jobs	[2]+ Running sleep 300 &
\$ kill %2	End the job number 2
\$ jobs	[2]+ Terminated sleep 300

Process attributes

Attribute	Meaning
The process ID or PID	A unique identification number used to refer to the process
The parent process ID or PPID	the number of the process (PID) that started this process
Nice number	<ul style="list-style-type: none">▪ The degree of friendliness of this process toward other processes▪ The Nice number is not the process priority▪ The process priority calculated based on this nice number and recent CPU usage of the process.
Terminal or TTY	Terminal to which the process is connected.
User name	The owner of the process (The user issuing the command)
Group name	The group owner of the process (The primary group of the user who started the process)

Displaying process information

Command	Effect
\$ ps	Shows you all processes running on your current terminal
\$ ps -e	Show all processes
\$ ps aux	Display all processes including user name note: -a : includes processes on all terminals -x :includes processes not attached to terminals -u : prints process owner information
\$ ps -eo comm,pid,user,nice,%cpu, %mem	Display only the specified columns
\$ top	Display all processes and updating the output periodically
\$ pstree	Display all processes in a tree format showing the relations between processes
\$ pstree -p	Show process ID in parentheses after each process name



Finding processes

Command	Effect
\$ ps axo comm,pid,user grep student	search for the word "student" in the resulting output
\$ pgrep -U student	Get all processes for user "student"
\$ pidof firefox	Search processes for the exact program name

The process priority

- The priority or importance of a job is defined by its nice number.
- A program with a high nice number is friendly to other programs (It is not an important job).
- The lower the nice number, the more important a job is and the more resources it will take without sharing them.

The process priority

- The nice number is a value between (-20 to +19), and you can change this value to affect on the process priority.
- Processes with higher priority will run first in each time slice, and will run longer before it turns to run ends.

Setting the Nice value

Command	Effect
\$ nice -n 15 firefox	Start new instance of firefox with nice value = 15
\$ ps -eo pid,comm,nice grep firefox	<u>4189</u> firefox 15
\$ renice -n 17 4189	Renice an existing process (firefox) with new nice value = 17 4189: old priority 15, new priority 17
\$ ps -eo pid,comm,nice grep firefox	4189 firefox 17

Note: As a normal user you can only increase the nice value. It is not allowed to you to decrease it. Only the “root” user can decrease the nice value

Sending signals to processes

- The operating system communicates to processes through signals.
- These signals report events or errors situations to processes.

Signal Name	Signal number	Meaning
SIGTERM	15	Terminate the process in an orderly way (Normal shutdown)
SIGINT	2	Interrupt the process. A process can ignore this signal
SIGKILL	9	Interrupt the process. A process cannot ignore this signal
SIGHUP	1	For daemons: reread the configuration file

Sending signals to processes

Command	Effect
\$ firefox & \$ firefox & \$ pidof firefox	6442 4298
\$ kill 6442	Terminate process by sending signal 15
\$ kill -9 4298	Kill the firefox process by sending signal 9
\$ killall firefox	Terminate all instances from firefox by sending signal 15
\$ killall -9 firefox	Kill all firefox processes by sending signal 9

The Gnome System Monitor

- You can manage processes using the GUI tool
(gnome-system-monitor)

LAB

09 - LAB (Process Management)

Chapter 11 | Software Management

- The concept of software repositories
- Using “yum”
- Using “rpm”

Installing Software

- **Tar balls** (compile and install)
- **Packages**
 - Packages are archives containing a list of files.
 - .deb (Debian Package)
 - .rpm (Red hat Package)
- **Package Managers & Dependency hell**
 - Allowing to install, manage, and remove packages easily and quickly.
- **Software Repositories**



The .rpm package file

- Software on RHEL is provided in the RPM (Red Hat Package Manager) format.
- This is a specific format used to archive the package and provide package metadata as well.
- A typical RPM file name looks like:

openssl-libs-1.0.2k-12.el7.x86_64.rpm

<package name>-<version>-<sub version>.<OS specific>.<platform>.rpm

Using YUM

Yellowdog update manager

YUM (Yellowdog update manager)

- Yum is the default tool used to manage software packages in RHEL.
- Yum is designed to work with repositories.
- The yum command considers all package dependencies and tries to look them up in the currently available repositories.

Configuring YUM repositories

/etc/yum.repos.d/<repofile>.repo

- To tell your server which repository to use, you need to create a file with a name that ends in .repo.
- In that file you need the following contents:

[Label] #used as an identifier in the repository file

name=<the name of the repository>

baseurl=<URL to the location of the repository>

URL

http://...

ftp://...

file:///...

Using YUM

Command	Result
# yum repolist	Verifying Repository Availability
# yum search <package>	Search for a package
# provides */file_name	Perform a deep search in the package to look for specific files within the package
# yum info <package>	Provide more information about the package
# yum list # yum list installed # yum list <package>	<ul style="list-style-type: none">• List all packages• List installed packages• show which version of the package is actually installed and which version is available as the most recent version in the repositories.
# yum install <package>... # yum install <.rpm file>	Install the specified packages Install a package from a local .rpm file
# yum remove <package>...	Remove the specified packages
# yum check-update # yum update # yum update <package>...	<ul style="list-style-type: none">• Check for updates without running it.• Updating All Packages and Their Dependencies to latest version from available repositories.• Update packages specified.

Using YUM

Command	Result
# yum group list	List pacakge groups.
# yum group list hidden	List pacakge groups including hidden groups.
# yum groups info "group"	Get information about packages available in the group.
# yum group install "group "	Install all packages from a group.
# yum group remove "group"	Remove all packages from a group.
# yum clean all	Remove all stored metadata

Creating your own repository

- it can be useful to create your own repository. This allows you to put your own RPMs in a directory and publish that directory as a repository.
- You need to copy all RPM packages to a directory that you want to use as a repository, and after doing that, you need to use the **createrepo** command to generate the metadata that enables you to use that directory as a repository.

Sample Installation

- tree
- figlet

Using the rpm tool

The RPM tool

- The rpm tool lacks important functionality:
 - Downloading packages from the internet
 - Installing dependencies automatically.
- The rpm tool has a separate “rpm database” located at **/var/lib/rpm** which contains all meta information about packages that are installed (via the rpm tool).
- The “rpm database” keeps track of all files, which enables complete removes of software.

Using rpm

Command	Description
# rpm -qa # rpm -qa wc -l	To obtain a list of all installed software.
# rpm -q <package name>	To verify whether one package is installed.
# rpm -qi <package name>	To get a description of the package.
# rpm -qf <file path>	To find the name of the package that the specified file comes from installing it.
# rpm -ql <package name>	To provide a list of files in the RPM package.
# rpm -qd <package name>	To show all documentation that is available in the package.
rpm -qc <package name>	To show all configuration files that are available in the package.

Using rpm

Command	Description
# rpm -qp <.rpm package file>	<ul style="list-style-type: none">The -p option is used with query options to query individual RPM package files instead of the RPM package database.This helps you find out what is actually in the package before it is installed.
# rpm -ivh <package rpm file> # rpm -Uvh <package rpm file>	<p>To install or upgrade a package</p> <p>The -U switch is the same as -i for install, except that older versions of the software are removed.</p>
# rpm -e <package name>	<p>To remove a package</p> <p>Note: rpm -e verifies dependencies, and thus will prevent you from accidentally erasing packages that are needed by other packages</p>

RPM Database & YUM database

- On your system, two package databases are maintained: the **yum database** and the **rpm database**.
- When you are installing packages through yum, the yum database is updated first, after which the updated information is synchronized to the RPM database.
- If you install packages using the rpm command, the update is written to the rpm database only and will not be updated to the yum database, which is an important reason not to use the rpm command anymore to install software packages.

Chapter 12 | Disk Management

- Storage Standards
- Block Devices Naming Convention
- MBR vs GPT Partitioning Schemes
- Managing Disk Partitions
- Managing Swap Space

Storage Standards

- ATA (advanced technology attachment)
 - Parallel Interface
 - Sometimes called IDE
 - Allows two devices per bus, one master and one slave
- SCSI (small computer system interface)
 - Parallel interface
 - Allows more than two devices
- SAS (Serial Attached SCSI)
 - Serial interface
- Serial ATA (SATA)
 - Serial interface

Block device

- It is a kind of file which represents a device of some kind, with data that can be read or written to it in blocks.
 - Hard disk device usually uses 1 block = 512 byte
 - ISO 9660 standard for cdrom uses 1 block = 2048 byte
- A block device has the letter **b** to denote the file type in the output of **ls -l**.
- # **lsblk**

Device naming

- IDE

Controller	Connection	Device name
IDE0	Master	/dev/hda
	Slave	/dev/hdb
IDE1	Master	/dev/hdc
	Slave	/dev/hdd

Device naming

- SCSI
 - All scsi drives start with **/dev/sd**, Followed by letters a,b,c,...,z which represent a separate attached disk to the scsi controller.
 - When you run out of letters (after **/dev/sdz**), you can continue with **/dev/sdaa** and **/dev/sdab** and so on.
- **Note:**
 - A modern Linux system will use **/dev/sd*** for scsi and sata devices, and also for sd-cards,usb-sticks, (legacy) ATA/IDE devices and solid state drives.

Demo

Add a new virtual disk

Discovering disk devices

- # fdisk -l
- # lsscsi

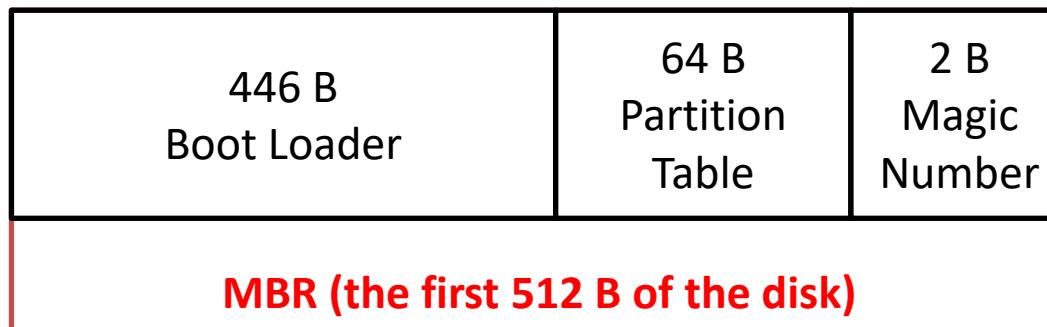
Disk partitions

- In order to use your hard disk, you should prepare it first for data.
- This can be achieved by creating disk partitions.
- You can create one or more partitions on your hard disk.
- A partition's **geometry** and size is usually defined by a starting and ending cylinder (sometimes by sector).

MBR partitioning scheme

MBR = Master Boot Record

- The MBR was defined as the first 512 bytes on a computer hard drive.



- The size that was used for the partition table was relatively small, just 64 bytes, so in the MBR no more than **4 partitions** could be created.

MBR partitioning scheme

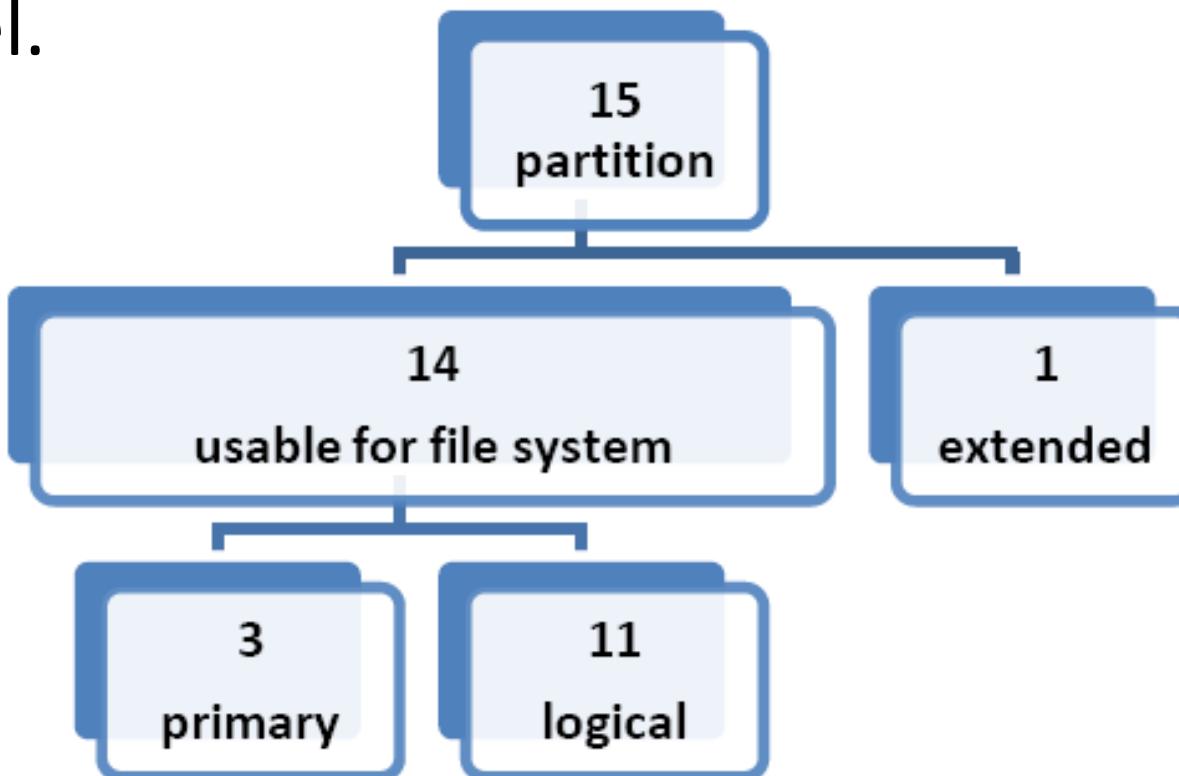
MBR = Master Boot Record

- In MBR, there are three types of partitions:
 1. **Primary** (maximum four).
 2. **Extended** (maximum one) (only used if you need more than 4 partitions).
 3. **Logical** (contained within the extended partition).
- The maximum size that could be used by a partition was limited to 2 TiB.
- Each partition has a type field that contains a code. This determines the computer's operating system or the partition's file system

MBR partitioning scheme

MBR = Master Boot Record

- Practically, in MBR a total number of 15 partitions only can be addressed by the Linux kernel.



GPT partitioning scheme

GPT = GUID (Global Unique ID)

- In GPT, up to a maximum number of 128 partitions can be created.
- The 2 TiB limit no longer exists.
- The maximum partition size is 8 zebibyte (ZiB).
- GPT uses a 128-bit global unique ID (GUID) to identify partitions.
- Because space that is available to store partitions is much bigger than 64 bytes (as used in MBR), there is no longer a need to distinguish between primary, extended, and logical partitions.
- A backup copy of the GUID partition table is created by default at the end of the disk, which eliminates the single point of failure that exists on MBR partition tables

Storage Measurement Units

Symbol	Name	Value	Symbol	Name	Value
KB	Kilobyte	1000^1	KiB	Kibibyte	1024^1
MB	Megabyte	1000^2	MiB	Mebibyte	1024^2
GB	Gigabyte	1000^3	GiB	Gibibyte	1024^3
TB	Terabyte	1000^4	TiB	Tebibyte	1024^4
PB	Petabyte	1000^5	PiB	Pebibyte	1024^5
EB	Exabyte	1000^6	EiB	Exbibyte	1024^6
ZB	Zettabyte	1000^7	ZiB	Zebibyte	1024^7
YB	Yottabyte	1000^8	YiB	Yobibyte	1024^8

Partition naming

- Partition names follow the same conventions like the disk names, and appending a number starting from 1

Partition name	Description
/dev/sda1	the 1 st primary partition on the disk /dev/sda
/dev/sda2	the 2 nd primary partition on the disk /dev/sda
/dev/sda3	the 3 rd primary partition on the disk /dev/sda
/dev/sda4	the 4 th primary partition on the disk /dev/sda

Partition naming

- Logical partition counting always starts at 5

Partition name	Description
/dev/sdb5	the 1 st logical partition on the disk /dev/sdb
/dev/sda6	the 2 nd logical partition on the disk /dev/sda

- Extended partition does not hold any data. Its function is only to hold the logical partitions

Partition naming example

MBR	P =/dev/hda1	P =/dev/hda2	Extended =/dev/hda3		Free space (unpartitioned)	
	L =/dev/hda5		Free space			

disk1 (IDE) = /dev/hda

MBR	P =/dev/hdb1	P =/dev/hdb2	P =/dev/hdb3	Extended= /dev/hdb4		
	L =/dev/hdb5	L =/dev/hdb6		Free space		

disk2 (IDE) = /dev/hdb

MBR	P =/dev/sda1	P =/dev/sda2	Extended =/dev/sda3			
	L =/dev/sda5	L =/dev/sda6		Free space		

disk3 (SCSI) = /dev/sda

MBR	P =/dev/sdb1	P =/dev/sdb2	Extended =/dev/sdb3			
	L =/dev/sdb5	L =/dev/sdb6		Free space		

disk4 (SATA) = /dev/sdb

Create partitions

1. Create (type = Linux 0x83)
2. Format (xfs, ext4 , ext3 , etx2,)

Note:

ext=Linux extended file system

man fs

3. mkdir <mount point>
4. mount

Partitioning new disks

- # fdisk -l
- # fdisk -l /dev/sdb
- # fdisk /dev/sdb
- # mkfs.ext4 <device file>
- # mkfs -t ext4 <device file>
- # mount <device file> <mount point>
- # df -h

Note:

- Use the **fdisk** tool to create MBR disks.
- Use the **gdisk** tool to create GPT disks.

Permanent mounts in /etc/fstab

- Adding a mount line to /etc/fstab cause to mount the partition after reboot.
- # man 5 fstab

<device file> <mount point> <fs> < mount options> <dump support> <Automatic check>
LABEL=<label> <mount point> <fs> <mount options> <dump support> <Automatic check>
UUID=<UUID> <mount point> <fs> <mount options> <dump support> <Automatic check>

Example:

/dev/sdb1 /part1 ext4 defaults 0 0

```
# mount -a  
# mount <mount point>
```

Permanent mounts in /etc/fstab

Filed	Description
Dump Support	<ul style="list-style-type: none">• Use 1 to enable support to backup using the dump utility. This may be necessary for some backup solutions.
Automatic Check	<ul style="list-style-type: none">• Specifies if the file system should be checked automatically when booting.<ul style="list-style-type: none"><input type="checkbox"/> Use 0 to disable automated check,<input type="checkbox"/> Use 1 if this is the root file system and it has to be checked automatically,<input type="checkbox"/> Use 2 for all other file systems that need automatic checking while booting. Network file systems should have this option set to 0.

Unmount partition

- # umount <device file>
- # umount <mount point>

Chapter 13 | Linux Installation

- Installation Methods
- Go Through The Process of Linux Installation

Installation Methods

- Installing from a CD-ROM/DVD-ROM
- Booting from a USB Key
- Installing through Network Booting

Demo

Installing CentOS 7

Chapter 14 | Basic Networking

- Verifying network connectivity
- Configuring Network Interfaces
- Configuring Name Resolution

Essential Network Concepts

- OSI Model.
- Ethernet.
- MAC Address.
- IP Address.
- Subnet.
- Unicast, multicast and broadcast.

Note:

Broadcast never passes through the router

Validating Network Configuration

Command	Description
# ip addr show [<nic>]	show current network settings
# ip link show [<nic>] # ip link set <nic> {up down}	Show the link state of the network interfaces Set the link state of the nic to up or down
# ifup <nic> # ifdown <nic>	Bring a network interface up Bring a network interface down
# ip route show	Show the default gateway

Configuring Network Configuration

- Networking on RHEL 7 is managed by the **NetworkManager** service.
 - # systemctl **status** NetworkManager (query the service status)
 - # systemctl **start** NetworkManager (start the service)
 - # systemctl **stop** NetworkManager (stop the service)
 - # systemctl **enable** NetworkManager (Run the service at startup)
 - # systemctl **disable** NetworkManager (Don't the service at startup)

Configuring Network Interface cards

- When NetworkManager comes up, it reads the network card configuration scripts, located in **/etc/sysconfig/network-scripts**.
- Network card configuration scripts are files with names that starts with **ifcfg** and is followed by the name of the network card.
- In RHEL 7, you can create multiple connections for a device (e.g. your laptop can connect your home network and the Corporate network at work).
- To manage the network connections that you want to assign to devices, you use the **nmtui** or the **nmcli** command

Demo

nmtui

Network Interface cards

- Every connection that you create is stored as a configuration file in the directory `/etc/sysconfig/network-scripts`
- Normally, there should be no need to modify these configuration files manually.
- If you want to edit these files manually , use the **nmcli con reload** command to activate the new configuration.

Configuring Hostname

- A hostname typically consists of different parts. These are the name of the host and the DNS domain in which the host resides.
- These two parts together make up for the fully qualified domain name (FQDN)

machine1.example.com

Configuring Hostname

- `# hostname` ‘ Get the current hostname
- `# hostname <new hostname>` ‘ Setting new hostname
(Not permanent)
- `# nmtui-hostname`
- `# vim /etc/hostname` ‘ Permanent , require reboot

Name Resolution

- The file `/etc/hosts`
- The DNS system
- `/etc/nsswitch.conf`

Configuring DNS Servers

/etc/resolve.conf

- The DNS servers to be used are indicated in the /etc/resolv.conf
 - nameserver 192.168.0.1**
 - nameserver 8.8.8.8**
- The NetworkManager configuration stores the DNS information in the interface configuration files inside /etc/sysconfig/network-scripts, and from there pushes the configuration to the /etc/resolv.conf file.

Verify host name resolution

```
# host www.google.com
```

(Query the DNS server)

```
# getent hosts <machine name>
```

(Query Both /etc/hosts, DNS)

```
# dnsdomainname
```

(Show the domain name)

Note:

- /etc/resolv.conf may be handled automatically (and overwritten) when the network is managed by NetworkManager or configured via DHCP.
- Do not specify the DNS servers directly in /etc/resolv.conf

Chapter 15 | Remote Login

- Using SSH (Secure Shell)

Remote Login

- telnet
 - Uses Telnet protocol.
 - All data, including usernames and passwords, is sent in clear text.
 - Use port 23 by default.
- ssh (Secure Shell)
 - Uses SSH protocol.
 - All data transmitted over a network is encrypted.
 - Relies on public key cryptography.
 - Use port 22 by default.

The public key cryptography (asymmetric cryptography)

- is an encryption scheme that uses two mathematically related, but not identical, keys
 1. public key
 2. private key
- You can not compute the private key from the public key
- Public keys can be freely shared.
- Private keys can be kept secret.

The public key cryptography (Business Applications)

- The main business applications for public-key cryptography are:
 - **Encryption** - content is encrypted using an individual's public key and can only be decrypted with the individual's private key
 - **Digital signatures** - content is digitally signed with an individual's private key and is verified by the individual's public key

SSH packages in Redhat

- OpenSSH is the most popular and most widely used.
 - openssh-server
 - Install sshd deamon
 - openssh-client
 - Install the ssh tool

Openssh client tools

- \$ ssh <machine>
 - \$ ssh <user>@<machine>
 - \$ ssh <user>@<machine> <command>
-
- \$ scp <Source files> <user>@<machine>:<path>
 - \$ scp <user>@<machine>:<path> <Destination>

Chapter 16 | Users & Groups Management

- Managing Local Users
- Managing Local Groups

Review on Chapter 5

- The concept of users & groups
- The users & groups configuration files
- Tools to query users & groups
 - \$ id [user]
 - \$ groups [user]...

Managing Local Users

Types of Users in Linux

- **Privileged users**
 - “root” is the default privileged user.
 - Use “root” only in tasks that require system administration privileges (i.e. installing software, managing users, and creating partitions on disk devices).
- **Unprivileged users**
 - Used for ordinary tasks (i.e. internet browsing, reading mails, writing office documents).

Types of Users Accounts

- **Normal Accounts**
 - Used by people who need to work on a server and who need limited access to the resources on that server.
 - These accounts typically have a password that is used for the authentication process.
- **System Accounts**
 - used by the services the server is offering.

Types of Users Accounts

- Both types of accounts (Normal account & System accounts) share common properties, which are kept in the files **/etc/passwd** and **/etc/shadow**.
- A part of the user properties is stored in **/etc/passwd**, while another part of the configuration of user properties is in **/etc/shadow**.

The structure of /etc/passwd

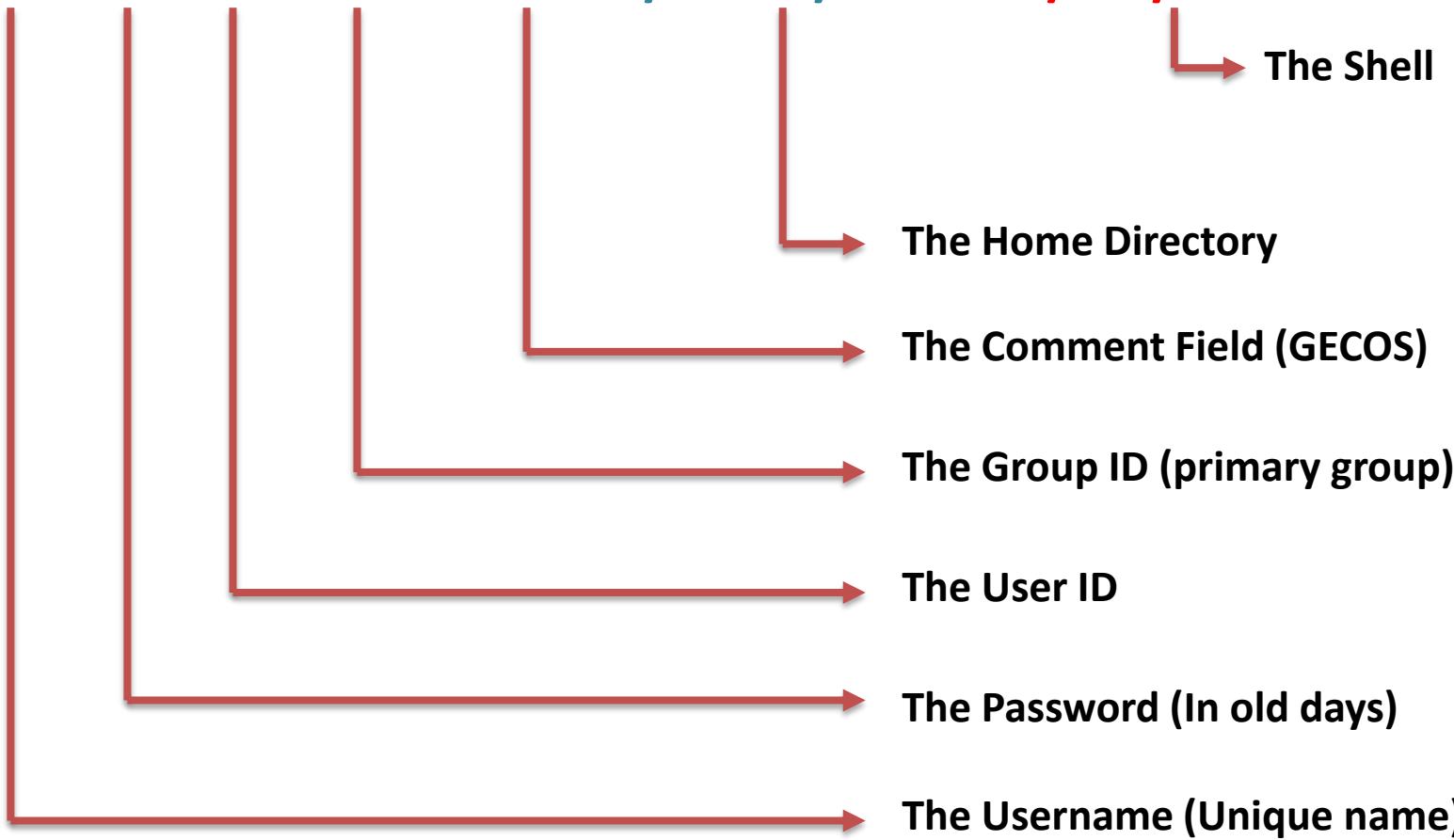
```
root:x:0:0:root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
.
.
.
student:x:1000:1000:student:/home/student:/bin/bash
```

- Each line represents only one user account.
- The fields are separated from each other by colons (“:”).
- **\$ man 5 passwd**

The structure of /etc/passwd

<name>:<password>:<UID>:<GID>:<GECOS>:<directory>:<shell>

student:x:1000:1000:student:/home/student:/bin/bash

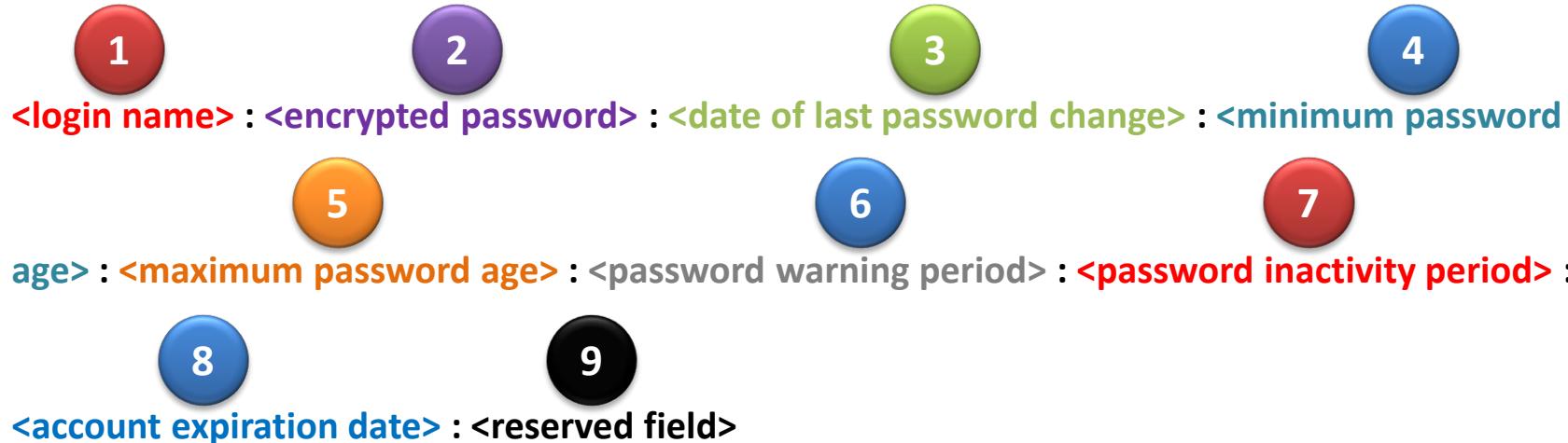


The structure of /etc/shadow

```
root:$6$A494MukQdcGSFukm$elxcipRIFIFzk9zQ7CITIJaqiBP3T1Su7blA8eRVFqcA3HGrghvtp3  
NveMbkl7igsIkMpWKQw16tNF.FPezXr.:0:99999:7:::  
bin:*:17632:0:99999:7:::  
daemon:*:17632:0:99999:7:::  
. . .  
student:$6$mSQH694BIKpgGx0x$y0RNl1H8Eri6F.UxliSKLJ9b56cWz8gBe/l487JcTp1vrwgvEcIR  
2Vzzls2SJ/u9/w37Sdpwyw20i5g3E4BNRX1::0:99999:7:::
```

- Each line represents only one user account.
- Each line contains 9 fields, separated by colons (“:”).
- \$ man 5 shadow

The structure of /etc/shadow



3 Date of last password change

- The date of the last password change, expressed as the number of days since Jan 1, 1970

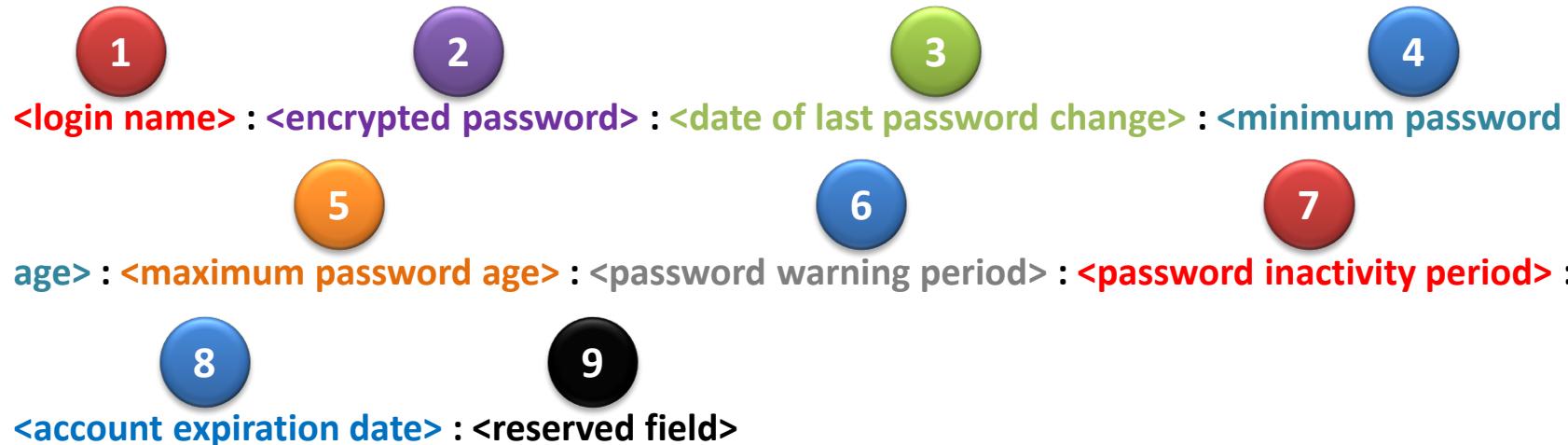
4 Minimum password age

- The number of **days** the user will have to wait before he/she will be allowed to change her password again (**0 or empty = No minimum age requirement**)

5 Maximum password age

- The number of **days** after which the user will have to change his/her password

The structure of /etc/shadow



6 Password warning period

- The number of days before a password is going to expire (**0 = No warning given**)

7 Password inactivity period

- The number of days after a password has expired

8 Account expiration date

- The date of expiration of the account, expressed as the number of **days** since Jan 1, 1970 (This is differ than password expiration).

9 Reserved field (For future use)

Creating Local Users

useradd [options] <login>

Options	Action
-m	<ul style="list-style-type: none">• Create Home directory
-d <path>	<ul style="list-style-type: none">• Specify the path of an existing directory to be the home directory
-c <comment>	<ul style="list-style-type: none">• Add a comment
-b <base dir path>	<ul style="list-style-type: none">• Specify the base directory
-s <shell>	<ul style="list-style-type: none">• Specify the shell
-u <uid>	<ul style="list-style-type: none">• User ID of the new account
-g <gid>	<ul style="list-style-type: none">• Specify the primary group
-G <group1>[,<group2>,.....]	<ul style="list-style-type: none">• Specify supplementary group(s)

Creating Local Users

- “`useradd`” depends on some default values .
- The default values are set in two configuration files:
 1. `/etc/login.defs`
 2. `/etc/default/useradd`

Creating Local Users

- \$ useradd -D Print defaults in /etc/default/useradd

GROUP=100

HOME=/home

INACTIVE=-1

EXPIRE=

SHELL=/bin/bash

SKEL=/etc/skel

CREATE_MAIL_SPOOL=yes

- \$ man useradd



Creating Local Users

\$ man 5 login.defs

Configuration parameter	Default value
PASS_MAX_DAYS 99999	<ul style="list-style-type: none">• Maximum number of days a password may be used.
PASS_MIN_DAYS 0	<ul style="list-style-type: none">• Minimum number of days allowed between password changes.
PASS_MIN_LEN 5	<ul style="list-style-type: none">• Minimum acceptable password length.
PASS_WARN_AGE 7	<ul style="list-style-type: none">• Number of days warning given before a password expires.
CREATE_HOME yes	<ul style="list-style-type: none">• “useradd” should create home directories for users by default.

Creating Local Users

\$ man 5 login.defs

Configuration parameter	Default value
USERGROUPS_ENAB yes	<ul style="list-style-type: none">• If Yes :<ul style="list-style-type: none">❑ Create by default a group with the name of the user (UPG) “User private Group”.❑ When removing the user, remove the user's private group too if it contains no more members.• If No :<ul style="list-style-type: none">❑ All users are made a member of the group “users”

Creating Local Users

\$ man 5 login.defs

Configuration parameter	Default value
MAIL_DIR /var/spool/mail	<ul style="list-style-type: none">• Directory where mailboxes reside
UMASK 077	<ul style="list-style-type: none">• “useradd” use this mask value to set the mode of the home directory for new users.• If not specified, it will be initialized to 022
ENCRYPT_METHOD SHA512	<ul style="list-style-type: none">• This defines the system default encryption algorithm for encrypting passwords.• It can take one of these values:<ul style="list-style-type: none"><input type="checkbox"/> DES (default, not recommended)<input type="checkbox"/> MD5<input type="checkbox"/> SHA256<input type="checkbox"/> SHA512.



Modifying Local Users

\$ usermod [options] <login>

Options	Action
-s <new shell>	<ul style="list-style-type: none">• Change the shell property
-g <group>	<ul style="list-style-type: none">• Change the primary group property
-aG <group>[,<group>]...	<ul style="list-style-type: none">• Add the specified supplementary groups <p>Note: If you don't use the “-a” option, all supplementary groups will be overridden.</p>
-c “new comment”	<ul style="list-style-type: none">• Change the user GECOS field in /etc/passwd (the comment field)
-L	<ul style="list-style-type: none">• Lock user account
-U	<ul style="list-style-type: none">• Unlock user account

Deleting Local Users

userdel [options] <login>

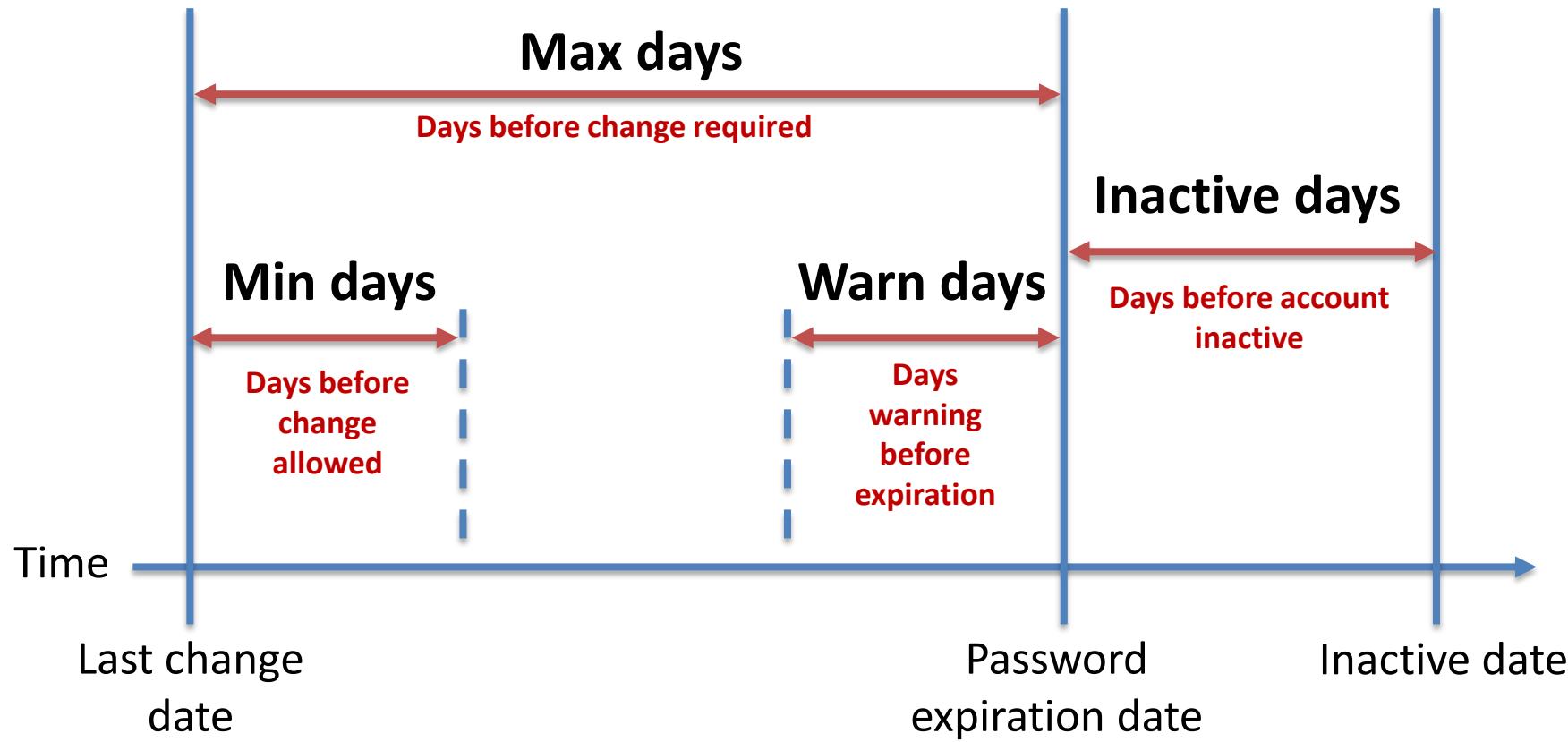
Options	Action
Without options	<ul style="list-style-type: none">• Delete a user without deleting user's files <p><u>Warning:</u> If when creating new user, the <u>new assigned uid</u> is as the same as the <u>deleted uid</u>, the new user will own the deleted user remaining files</p>
-r --remove	<ul style="list-style-type: none">• Delete the user and the following files:<ul style="list-style-type: none"><input type="checkbox"/> All files in the user's home directory<input type="checkbox"/> The home directory itself<input type="checkbox"/> The user's mail spool. <p>Note: Files located in other file systems will have to be searched for and deleted manually.</p>



Password Aging



Password Aging





Password Aging

chage [options] <user>

Options	Action
-l <user>	<ul style="list-style-type: none">list user's current aging information
-d <last day>	<ul style="list-style-type: none">Set the date of the last password change.<last day> can be expressed as:<ul style="list-style-type: none">The number of days since January 1st, 1970The format YYYY-MM-DD
-d 0	<ul style="list-style-type: none">force password update on next login
-E <date> -E 2019-03-26	<ul style="list-style-type: none">Set the account expiration on the specified date <p>Note:</p> <ul style="list-style-type: none">Account expiration and password expiration are different two things.Use the command "date" to calculate the account expiration date as follow: <code># date -d "+30 days"</code>
-E -1	<ul style="list-style-type: none">Cancel the account expiration



Password Aging

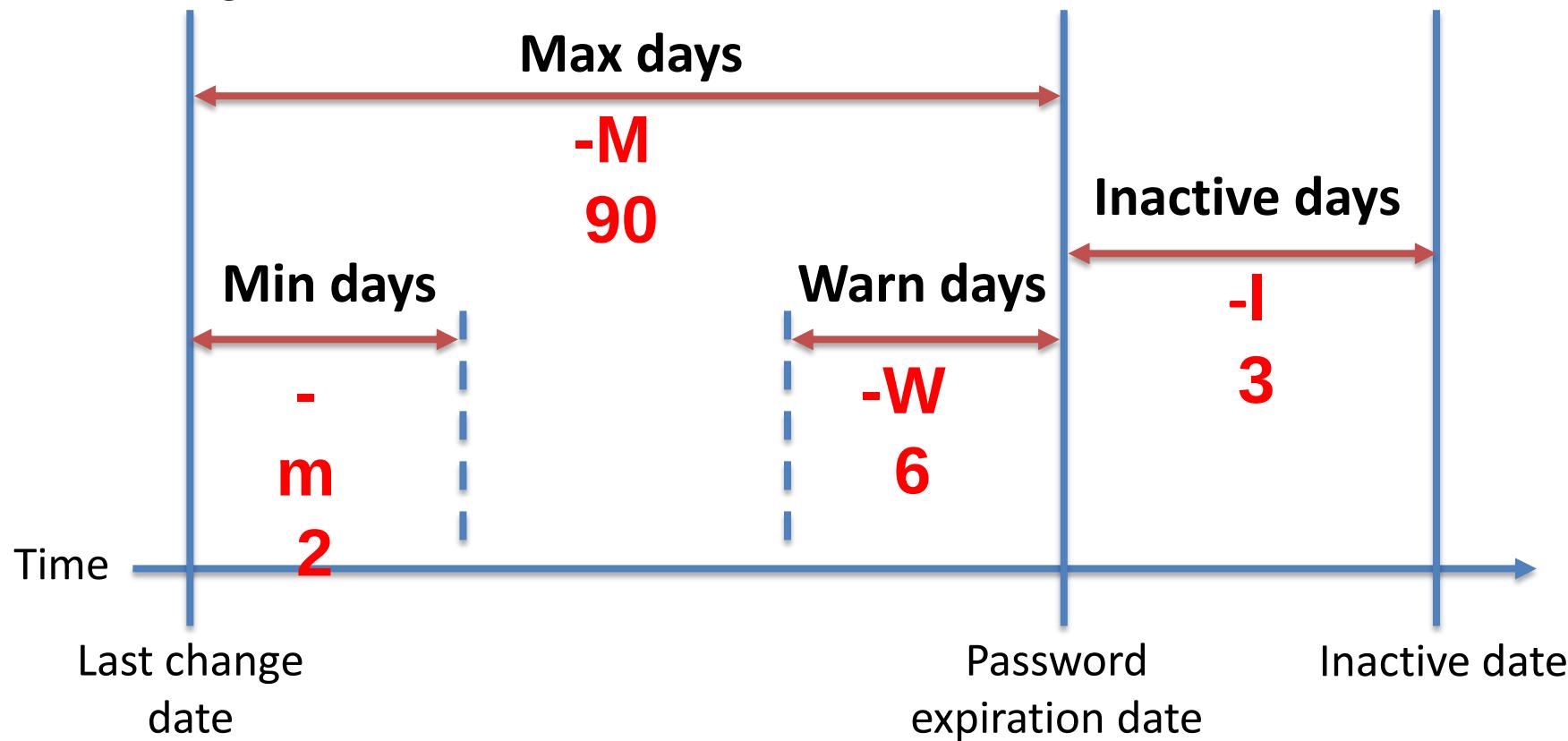
- **# chage [options] <user>**

Options	Action
-m <number in days>	<ul style="list-style-type: none">• Set the minimum number of days between password changes.
-M <number in days>	<ul style="list-style-type: none">• Set the maximum number of days during which a password is valid.
-W <number in days>	<ul style="list-style-type: none">• Number of days of warning before password expires
-I <number in days>	<ul style="list-style-type: none">• Set the number of days of inactivity after a password has expired before the account is locked
Without any option	<ul style="list-style-type: none">• Run in the interactive mode

Password Aging

```
# chage -m <days> -M<days> -W<days> -I<days> <login>
```

```
# chage -m 2 -M 90 -W 6 -I 3 user12
```



Managing Local Groups

Types of Groups

- Primary group
 - Every user has exactly one primary group defined by GID in /etc/passwd.
 - Normally, the primary group owns new files created by the user.
 - By default, the primary group of a newly created user is a newly created group with the same name as the user, and the user is the only member of this **User Private Group (UPG)**

Types of Groups

- Supplementary group(s)
 - Users may be a member of zero or more supplementary groups.
 - All groups including supplementary groups saved in /etc/group.
 - Each group in /etc/group keep a list of users that use it as a supplementary group.

The structure of /etc/group

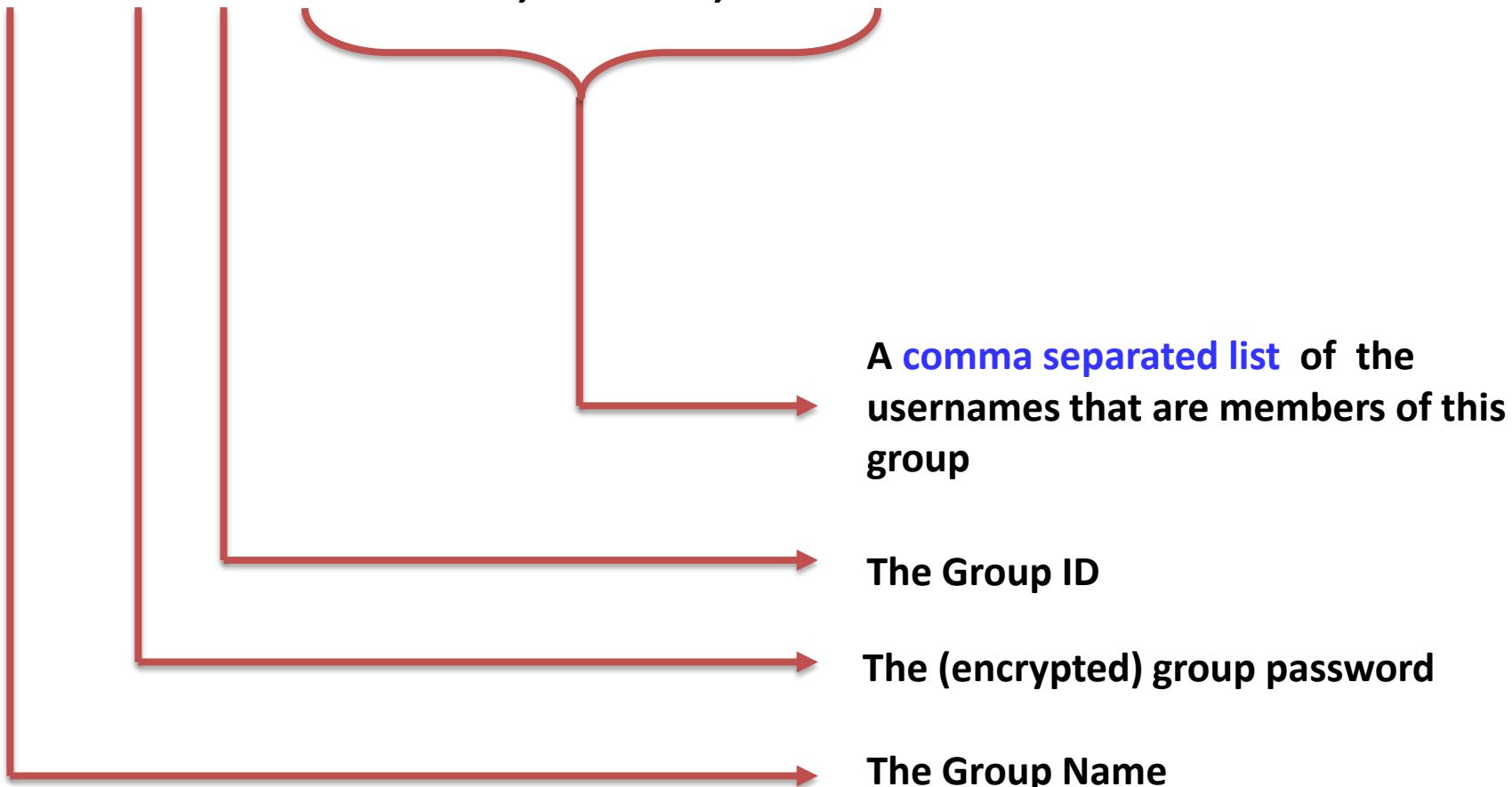
```
root:x:0:  
bin:x:1:  
kmem:x:9:  
wheel:x:10:instructor,student,visitor  
. . .  
instructor:x:1000:
```

- Each line represents only one group.
- The fields are separated from each other by colons (“:”).
- **\$ man 5 group**

The structure of /etc/group

<group_name>:<password>:<GID>:<list,of,users,in>this,group>

wheel : x : 10 : instructor,student,visitor



Creating Local Groups

- **\$ groupadd [options] <group>**

Options	Action
Without any options	<ul style="list-style-type: none">• Create the new group with the first free available GID
-g <GID>	<ul style="list-style-type: none">• Create the new group with the specified GID

The Group password

- Every group can have administrators, members and a password.
- A group password can be used by users that want to join the group on a temporary basis, so that access to files the group has access to is allowed.
- A group administrator can set the group password, Add/remove members using the “gpasswd” tool.

The Group password

- If a password is set, the members can use “**newgrp**” without a password to login to the group, but non-members must supply the password.
- Groups names are saved in **/etc/group**
- Groups passwords are saved in **/etc/gshadow**

The structure of /etc/gshadow

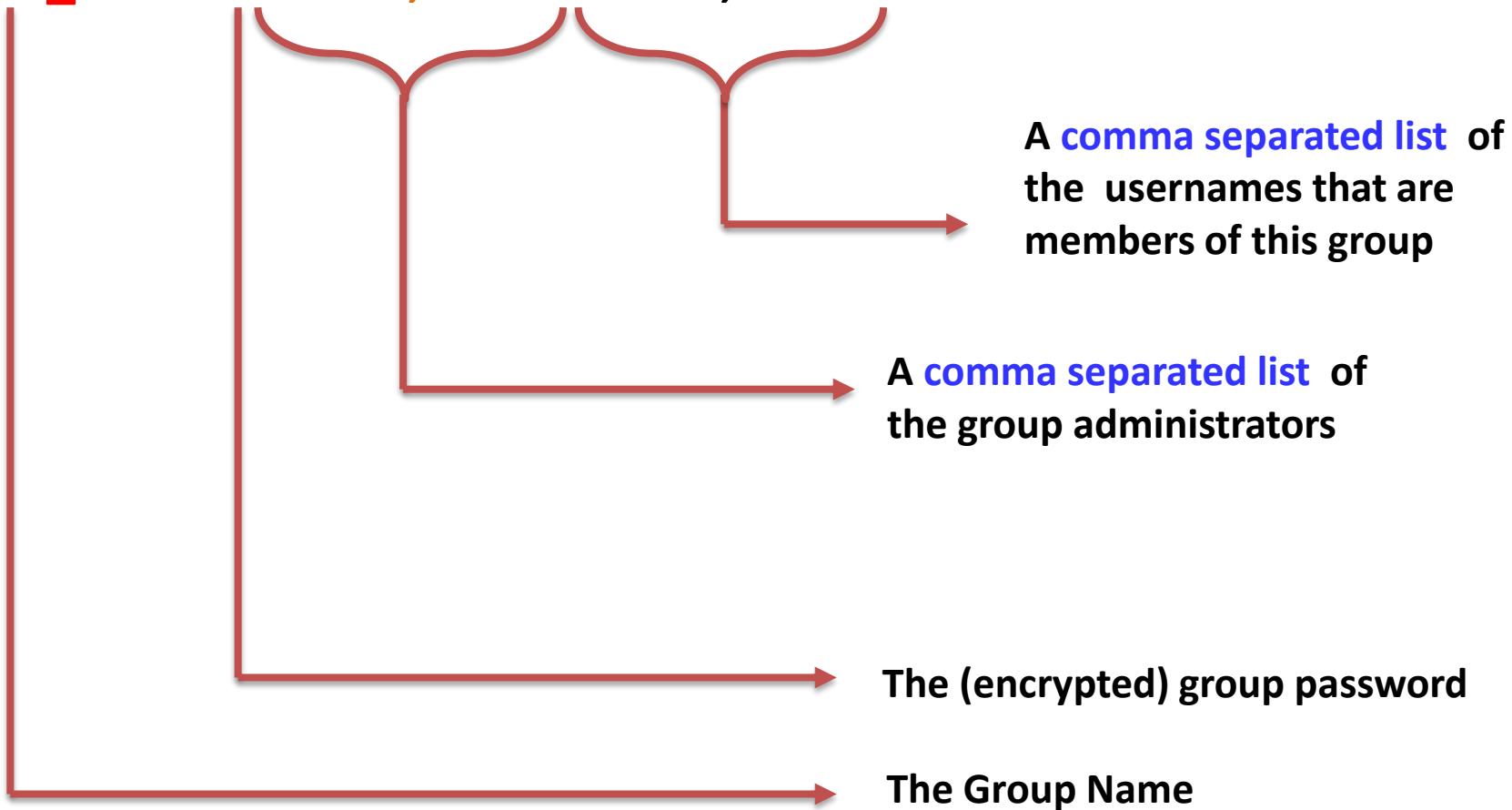
```
root:::  
bin:::  
daemon:::  
. . .  
staff_users::user1,user2:user4,user3
```

- Each line represents only one group.
- The fields are separated from each other by colons (“:”).
- **\$ man 5 gshadow**

The structure of /etc/gshadow

<group_name>:<password>:<administrators>:<members>

staff_users: : user1,user2: user5,user6



Modifying Local Groups

\$ groupmod [options] <group>

Options	Action
-g <new GID>	<ul style="list-style-type: none">• Change the named group “GID” to a new GID <p><u>Note:</u></p> <ul style="list-style-type: none"><input type="checkbox"/> Users who use <group> as primary group will be updated to keep the group as their primary group.<input type="checkbox"/> Any files that have the old group ID and must continue to belong to <group>, must have their group ID changed manually.
-n <new name>	<ul style="list-style-type: none">• Change the named group “name” to a new name

Note:

“groupmod” allow you to change the name or group ID of the group, but it **does not** allow you to add group members



Modifying Local Groups

\$ gpasswd [options] <group>

Options	Action
Without Any options	<ul style="list-style-type: none">Set a password for the named group, and saved in /etc/gshadow
-r	<ul style="list-style-type: none">Remove the password from the named group
-A <user>[,<user>]...	<ul style="list-style-type: none">Set the list of administrative users to the named group. Note:<ul style="list-style-type: none">This will appear in the named group line in /etc/gshadow in the third column.Administrative users need not to be members of the group.Used only by “root”, not the group admin userDelete all administrative users from the named group.
-A "	
-M <user>[,<user>]...	<ul style="list-style-type: none">Set the list of group members.<ul style="list-style-type: none">This will appear in the named group line in /etc/gshadow in the fourth column.Used only by “root”, not the group admin user

Modifying Local Groups

- **\$ gpasswd [options] <group>**

Options	Action
-a <user>	<ul style="list-style-type: none">• Add the user to the named group
-d <user>	<ul style="list-style-type: none">• Remove the user from the named group.
-R	<ul style="list-style-type: none">• Restrict the access to the named group<ul style="list-style-type: none"><input type="checkbox"/> Only member can login to the group<input type="checkbox"/> The group password is set to "!" in /etc/gshadow

(Remember) Changing The Default Group Ownership

- If the user is a member of more than one group, he can change the effective primary group.

\$newgrp <Other Group>

#change the effective primary group

- All new files that the user creates will get this **<Other Group>** as their group owner.
- This behavior will be continued until the user uses the exit command or logs out.
- If a user uses the “newgrp” but is not a member of the target group, the shell prompts for the group password. And he enter the correct password, then he will be considered a member of the group in the current session.
- A group password can be set for the group using the “gpasswd” command.

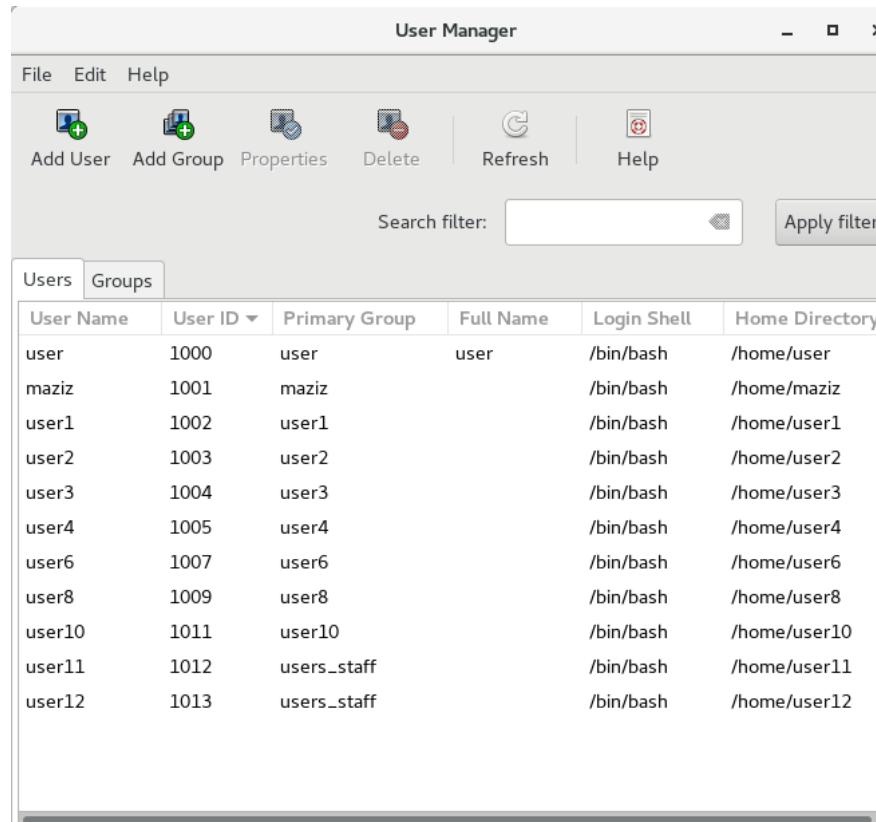


Deleting Local Groups

groupdel <group>

The GUI tool (system-config-users)

yum install -y system-config-users



Final Word