

# =Content

- ✓ What Is "The Shell"
- ✓ Understanding the File System Tree & its Navigation
- ✓ Exploring the System
- ✓ Manipulating Files and Directories
- ✓ I/O Redirection
- ✓ Expansion
- ✓ Permissions
- ✓ Job Control
- ✓ Configuration and The Environment
- ✓ Introduction To vi
- ✓ Customizing the Prompt

# What is The “Shell”?

The shell is a program that takes commands  
= from the keyboard and gives them to the  
operating system to perform.

# What is The “Shell”?

- ✓ Almost all Linux distributions supply a shell program from the GNU Project called bash.

=

- ✓ The name bash is an acronym for **Bourne Again Shell**, a reference to the fact that bash is an enhanced replacement for sh, the original Unix shell program written by **Steve Bourne**.

# Terminal Emulators

- ✓ When using a graphical user interface, we need another program called a terminal emulator to interact with the shell.
- ✓ KDE uses konsole and GNOME uses gnome-terminal, though it's likely called simply "terminal" on our menu.  
KDE = k desktop environment & GNOME = Network object model environment
- ✓ A number of other terminal emulators are available for Linux, but they all do basically the same thing: give us access to the shell.

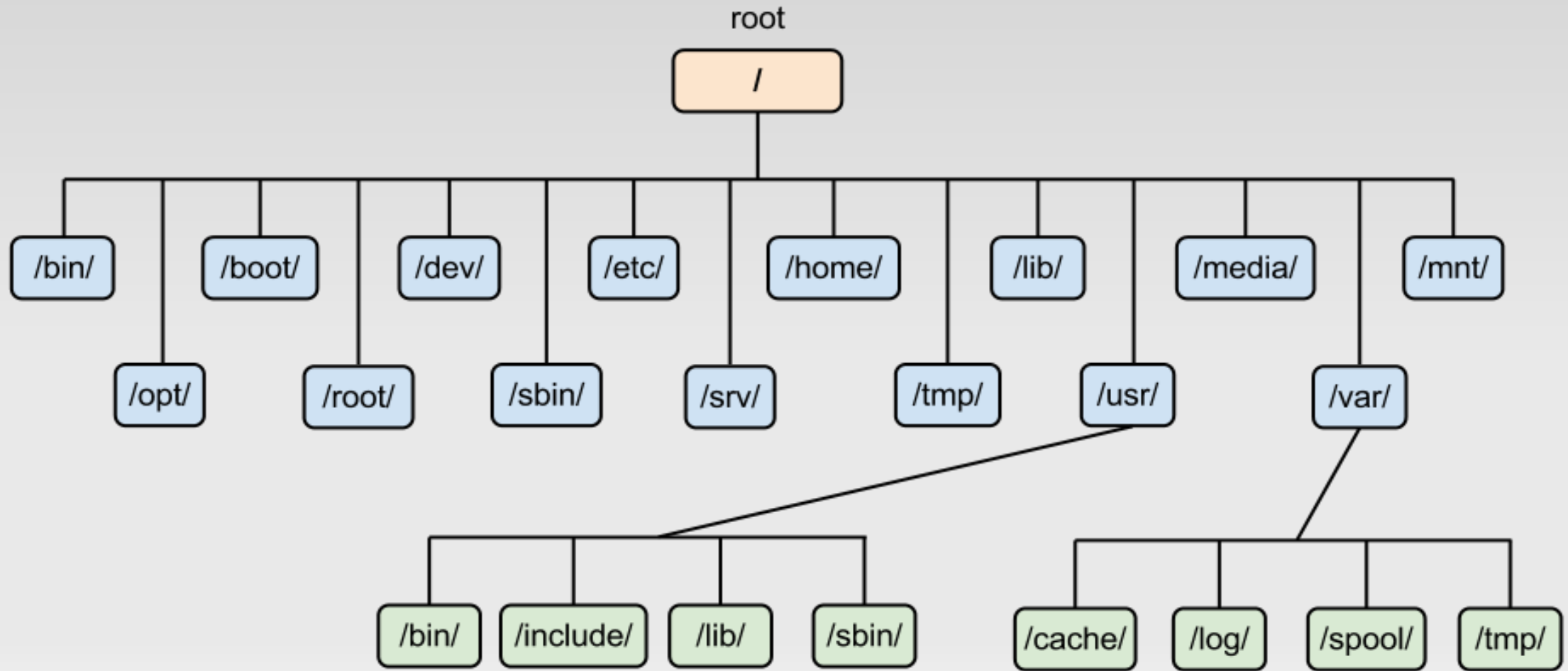
# Some Simple Commands

- ✓ **date** - This command displays the current time and date.
- ✓ **Cal** - displays a calendar of the current month.
- ✓ **df** - To see the current amount of free space on your disk drives
- ✓ **free** - to display the amount of free memory
- ✓ **exit** - We can end a terminal session by either closing the terminal emulator window or entering the command at the shell prompt

# Navigation

- **pwd** — Print name of current working directory.
- 
- **cd** — Change directory.
  - **ls** — List directory contents

# Filesystem Tree



# Filesystem Tree

- ✓ Unix-like operating system such as Linux organizes its files in what is called a hierarchical directory structure.
- ✓ This means that they are organized in a tree-like pattern of directories (sometimes called folders in other systems), which may contain files and other directories.
- ✓ The first directory in the filesystem is called the root directory.
- ✓ The root directory contains files and subdirectories, which contain more files and subdirectories, and so on.



# Absolute Pathnames

=

An absolute pathname begins with the root directory and follows the tree branch by branch until the path to the desired directory or file is completed.

# Relative Pathnames

- ✓ Where an absolute pathname starts from the root directory and leads to its destination, a relative pathname starts from the working directory.
- ✓ To do this, it uses a couple of special symbols to represent relative positions in the filesystem tree.
- ✓ These special symbols are . (dot) and .. (dot dot).
- ✓ The . symbol refers to the working directory and the .. symbol refers to the working directory's parent directory.

# Some Simple Commands

Shortcut	Result
<code>cd</code>	Changes the working directory to your home directory.
<code>cd -</code>	Changes the working directory to the previous working directory.
<code>cd ~<i>username</i></code>	Changes the working directory to the home directory of <i>username</i> . For example, <code>cd ~bob</code> changes the directory to the home directory of user <i>bob</i> .

# ls Command

- ✓ ls – List directory Content
- ✓ ls commands variations

=

# Options and Arguments

✓ command -options arguments

=

# Determining a File's Type with file

We can invoke the file command this way:

→ file filename

show the file is txt or what

# Viewing File Contents with less

- ✓ The less command is a program to view text files.
- ✓ Throughout Linux system, there are many files that contain human-readable text.
- ✓ The less program provides a convenient way to examine them.

The less command is used like this:

- ✓ `less filename`

# Types of files in the Linux system

✓ **General Files** - It is also called ordinary files. It may be an image, video, program, or simple text files. These types of files can be in ASCII or Binary format. It is the most commonly used file in the Linux system.

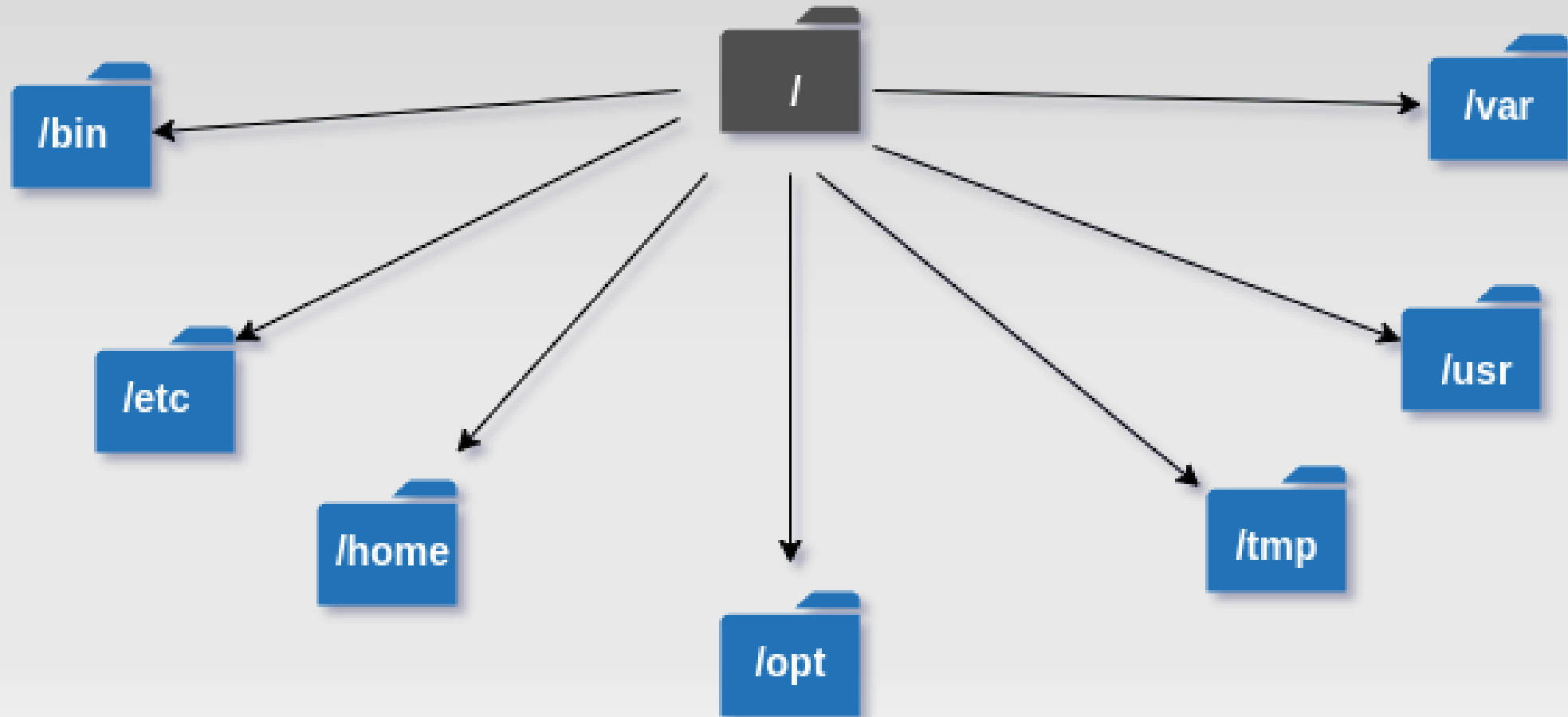
=

✓ **Directory Files** - These types of files are a warehouse for other file types. It may be a directory file within a directory (subdirectory).

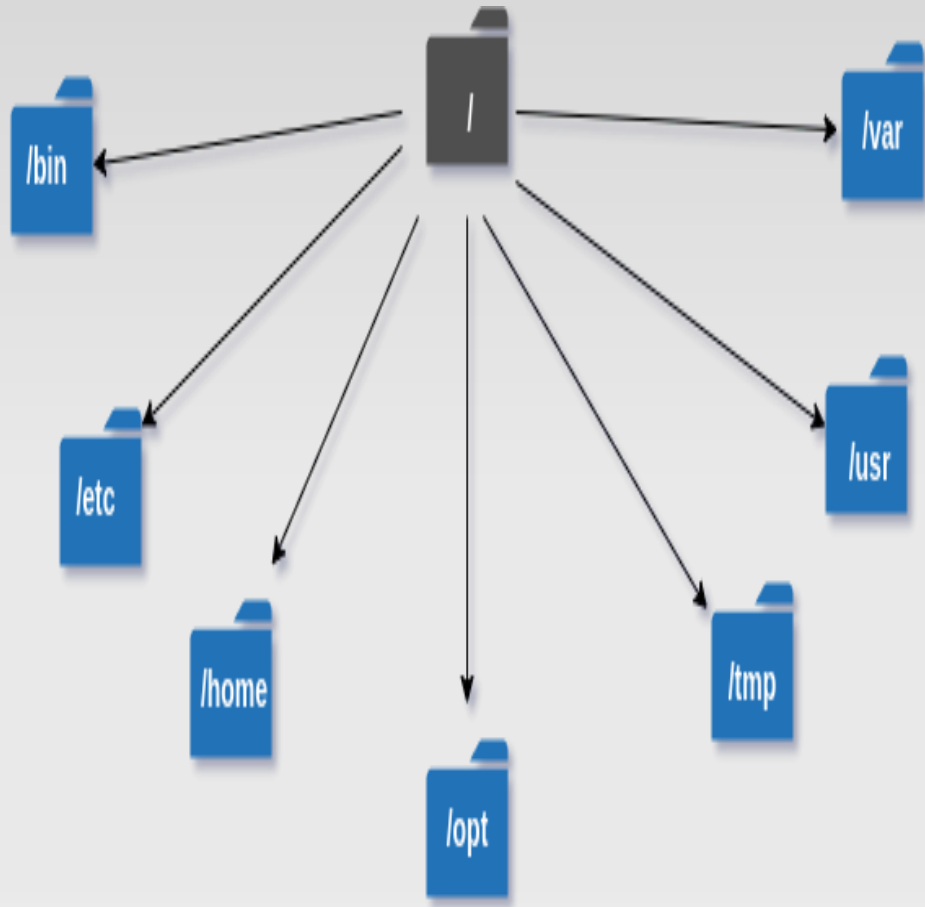
✓ **Device Files** - In a Windows-like operating system, devices like CD-ROM, and hard drives are represented as drive letters like F: G: H whereas in the Linux system device are represented as files. As for example, /dev/sda1, /dev/sda2 and so on.



# Types of files in the Linux system



# Types of files in the Linux system



✓ /bin – binary or executable programs.

✓ /etc – system configuration files.

✓ /home – home directory. It is the default current directory.

✓ /opt – optional or third-party software.

✓ /tmp – temporary space, typically cleared on reboot.

✓ /usr – User related programs.

✓ /var – log files.

# Types of files in the Linux system

- ✓ /boot- It contains all the boot-related information files and folders such as conf, grub, etc.
- ✓ /dev – It is the location of the device files such as dev/sda1, dev/sda2, etc.
- ✓ /lib – It contains kernel modules and a shared library.
- ✓ /lost+found – It is used to find recovered bits of corrupted files.
- ✓ /media – It contains subdirectories where removal media devices inserted.
- ✓ /mnt – It contains temporary mount directories for mounting the file system.

# Types of files in the Linux system

- ✓ /proc – It is a virtual and pseudo-file system to contains info about the running processes with a specific process ID or PID.
- ✓ /run – It stores volatile runtime data.
- ✓ /sbin – binary executable programs for an administrator.
- ✓ /srv – It contains server-specific and server-related files.
- ✓ /sys – It is a virtual filesystem for modern Linux distributions to store and allows modification of the devices connected to the system.

# Manipulating Files and Directories

✓ `cp` – Copy files and directories.

✓ `mv` – Move/rename files and directories.

✓ `mkdir` – Create directories.

✓ `rm` – Remove files and directories.

✓ `ln` – Create hard and symbolic links.

# Manipulating Files and Directories

## mkdir – Create Directories

✓ The mkdir command is used to create directories. It works like this:

✓ mkdir directory...

✓ mkdir dir1

would create a single directory named dir1, while

✓ mkdir dir1 dir2 dir3

would create three directories named dir1, dir2, and dir3.

# Manipulating Files and Directories

## `cp` – Copy Files and Directories

✓ The `cp` command copies files or directories. It can be used two different ways:

`cp item1 item2`

to copy the single file or directory `item1` to file or directory `item2` and:

`cp item... directory`

# Manipulating Files and Directories

The mv command performs both file moving and file renaming, depending on how it is used. In either case, the original filename no longer exists after the operation. mv is used in much the same way as cp:

```
mv item1 item2
```

to move or rename file or directory item1 to item2 or

```
mv item... directory
```



# Manipulating Files and Directories

**rm — Remove Files and Directories**

The rm command is used to remove (delete) files and directories, like this:

**rm item..**

# Manipulating Files and Directories

## **ln – Create Links**

The **ln** command is used to create either hard or symbolic links. It is used in one of two ways:

### **ln file link**

to create a hard link and

### **ln -s item link**

to create a symbolic link where item is either a file or a directory.

# Hard Links –

- Each hard linked file is assigned the same Inode value as the original, therefore they reference the same physical file location. Hard links more flexible and remain linked even if the original or linked files are moved throughout the file system, although hard links are unable to cross different file systems.
- `ls -l` command shows all the links with the link column shows number of links.
- Links have actual file contents
- Removing any link, just reduces the link count, but doesn't affect other links.
- Even if we change the filename of the original file then also the hard links properly work.
- We cannot create a hard link for a directory to avoid recursive loops.
- If original file is removed then the link will still show the content of the file.
- The size of any of the hard link file is same as the original file and if we change the content in any of the hard links then size of all hard link files are updated.
- The disadvantage of hard links is that it cannot be created for files on different file systems and it cannot be created for special files or directories.

# Soft Links –

- A soft link is similar to the file shortcut feature which is used in Windows Operating systems. Each soft linked file contains a separate Inode value that points to the original file. As similar to hard links, any changes to the data in either file is reflected in the other. Soft links can be linked across different file systems, although if the original file is deleted or moved, the soft linked file will not work correctly (called hanging link).
- `ls -l` command shows all links with first column value `l?` and the link points to original file.
- Soft Link contains the path for original file and not the contents.
- Removing soft link doesn't affect anything but removing original file, the link becomes "dangling" link which points to nonexistent file.
- A soft link can link to a directory.
- The size of the soft link is equal to the length of the path of the original file we gave. E.g if we link a file like `ln -s /tmp/hello.txt /tmp/link.txt` then the size of the file will be 14bytes which is equal to the length of the `"/tmp/hello.txt"`.
- If we change the name of the original file then all the soft links for that file become dangling i.e. they are worthless now.
- Link across file systems: If you want to link files across the file systems, you can only use `symlinks/soft links`.

# Basic Commands

**type** — Indicate how a command name is interpreted.

**which** — Display which executable program will be executed.

**man** — Display a command's manual page.

**apropos** — Display a list of appropriate commands.

**info** — Display a command's info entry.

**whatis** — Display a very brief description of a command.

**alias** — Create an alias for a command.

# I/O redirection

- ✓ **cat** – Concatenate files.
- ✓ **sort** – Sort lines of text.
- ✓ **uniq** – Report or omit repeated lines.
- ✓ **wc** – Print newline, word, and byte counts for each file.
- ✓ **grep** – Print lines matching a pattern.
- ✓ **head** – Output the first part of a file.
- ✓ **tail** – Output the last part of a file.
- ✓ **tee** – Read from standard input and write to standard output and files.

# I/O redirection

## Redirecting Standard Output

✓ I/O redirection allows us to redefine where standard output goes.

✓ To redirect standard output to another file instead of the screen, we use the `>` redirection operator followed by the name of the file.

✓ To append redirected output to a file instead of overwriting the file from the beginning? For that, we use the `>>` redirection operator

# I/O redirection

## Redirecting Standard Error

- ✓ Redirecting standard error lacks the ease of using a dedicated redirection operator.
- ✓ To redirect standard error we must refer to its file descriptor. A program can produce output on any of several numbered file streams.
- ✓ While we have referred to the first three of these file streams as standard input, output, and error, the shell references them internally as file descriptors 0, 1, and 2, respectively.
- ✓ The shell provides a notation for redirecting files using the file descriptor number. Since standard error is the same as file descriptor 2
- ✓ Ex., Command 2> filename



# I/O redirection

## Redirecting Standard Error

- ✓ Redirecting standard error lacks the ease of using a dedicated redirection operator.
- ✓ To redirect standard error we must refer to its file descriptor. A program can produce output on any of several numbered file streams.

==

- ✓ While we have referred to the first three of these file streams as standard input, output, and error, the shell references them internally as file descriptors 0, 1, and 2, respectively.
- ✓ The shell provides a notation for redirecting files using the file descriptor number. Since standard error is the same as file descriptor 2
- ✓ Ex.,    `Command 2> filename`

# I/O redirection

## Redirecting Standard Output and Standard Error

✓ Ex., `Command &> filename`

==

# I/O redirection

## Disposing of Unwanted Output

- ✓ If we don't want output from a command—we just want to throw it away.
- ✓ This applies particularly to error and status messages. The system provides a way to do this by redirecting output to a special file called `/dev/null`.
- ✓ This file is a system device called a bit bucket, which accepts input and does nothing with it.
- ✓ To suppress error messages from a command, we do this:

**Ex.,** `ls -l /bin/usr 2> /dev/null`

# I/O redirection

## Cat – Concatenate files

- ✓ The cat command reads one or more files and copies them to standard output like so: `cat [file...]`
- == ✓ You can use it to display files without paging
- ✓ Cat can accept more than one file as an argument, it can also be used to join files together.
- ✓ Using the `<` redirection operator, we change the source of standard input from the keyboard to the file

# I/O redirection

## Pipeline

- ✓ The ability of commands to read data from standard input and send to standard output is utilized by a shell feature called pipelines.
- ✓ Using the pipe operator | (vertical bar), the standard output of one command can be piped into the standard input of another.
- ✓ `command1 | command2`

# I/O redirection

## Filter

- ✓ Pipelines are often used to perform complex operations on data.
- == ✓ It is possible to put several commands together into a pipeline.
- ✓ Frequently, the commands used this way are referred to as filters. Filters take input, change it somehow, and then output it.

# I/O redirection

## uniq—Report or Omit Repeated Lines

- ✓ The `uniq` command is often used in conjunction with `sort`. `uniq` accepts a sorted list of data from either standard input or a single filename argument and, by default, removes any duplicates from the list.

- ✓ Ex., `ls /bin /usr/bin | sort | uniq | less`

# I/O redirection

## **wc—Print Line, Word, and Byte Counts**

- ✓ The **wc** (word count) command is used to display the number of lines, words, and bytes contained in files.

For example: **wc ls-output.txt**

**7902 64566 503634 ls-output.txt**

- ✓ In this case it prints out three numbers: lines, words, and bytes contained in **ls-output.txt**. Like our previous commands, if executed without command line arguments, **wc** accepts standard input.
- ✓ The **-l** option limits its output to only report lines. Adding it to a pipeline is a handy way to count things.



# I/O redirection

## grep—Print Lines Matching a Pattern

- ✓ grep is a powerful program used to find text patterns within files, like this: `grep pattern [file...]`
- ✓ When grep encounters a “pattern” in the file, it prints out the lines containing it.

==

# I/O redirection

## head/tail—Print First/Last Part of Files

- ✓ Sometimes you don't want all the output from a command.
- ✓ You may want only the first few lines or the last few lines.
- ✓ The head command prints the first 10 lines of a file, and the tail command prints the last 10 lines.
- ✓ By default, both commands print 10 lines of text, but this can be adjusted with the -n option

# I/O redirection

## **tee—Read from Stdin and Output to Stdout and Files**

- ✓ The tee program reads standard input and copies it to both standard output (allowing the data to continue down the pipeline) and to one or more files.
- ✓ This is useful for capturing a pipeline's contents at an intermediate stage of processing.

# Expansions

✓ Pathname Expansion

✓ Tilde Expansion

✓ Arithmetic Expansion

✓ Brace Expansion

✓ Parameter Expansion

✓ Quoting – Double Quoting

# Advanced Keyboard Tricks

✓ clear – Clear the screen.

✓ history – Display the contents of the history list.

# Permissions

- ✓ The computer would support many users at the same time.
- ✓ In order to make this practical, a method had to be devised to protect the users from each other.
- ✓ After all, the actions of one user could not be allowed to crash the computer, nor could one user interfere with the files belonging to another user.

# Permissions

- ✓ `id` – Display user identity.
- ✓ `chmod` – Change a file's mode.
- ✓ `umask` – Set the default file permissions.
- ✓ `su` – Run a shell as another user.
- ✓ `sudo` – Execute a command as another user.
- ✓ `chown` – Change a file's owner.
- ✓ `chgrp` – Change a file's group ownership.
- ✓ `passwd` – Change a user's password.

# Permissions

**id** – Display user identity.

- ✓ To find out information about your identity, use the **id** command:
- ✓ When user accounts are created, users are assigned a number called a user ID, or **uid**.
- ✓ This is then, for the sake of the humans, mapped to a username.
- ✓ The user is assigned a primary group ID, or **gid**, and may belong to additional groups.



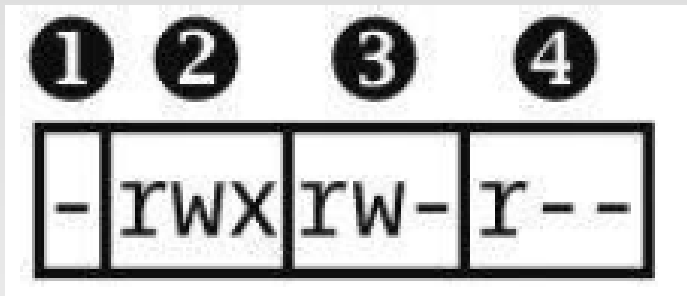
# Permissions

**id – Display user identity.**

- ✓ To find out information about your identity, use the **id** command:
- ✓ When user accounts are created, users are assigned a number called a user ID, or **uid**.
- ✓ This is then, for the sake of the humans, mapped to a username.
- ✓ The user is assigned a primary group ID, or **gid**, and may belong to additional groups.

# Permissions

Reading, Writing, and Executing



1	2	3	4
-	rwx	rW-	r--

1. File Type
2. Owner Permissions
3. Group Permissions
4. World Permissions

# Permissions

Attribute	File Type
-	A regular file.
d	A directory.
l	A symbolic link. Notice that with symbolic links, the remaining file attributes are always <code>rw-rw-rw-</code> and are dummy values. The real file attributes are those of the file the symbolic link points to.
c	A <i>character special file</i> . This file type refers to a device that handles data as a stream of bytes, such as a terminal or modem.
b	A <i>block special file</i> . This file type refers to a device that handles data in blocks, such as a hard drive or CD-ROM drive.

# Permissions

Attribute	Files	Directories
r	Allows a file to be opened and read.	Allows a directory's contents to be listed if the execute attribute is also set.
w	Allows a file to be written to or truncated; however, this attribute does not allow files to be renamed or deleted. The ability to delete or rename files is determined by directory attributes.	Allows files within a directory to be created, deleted, and renamed if the execute attribute is also set.
x	Allows a file to be treated as a pro-gram and executed. Program files written in scripting languages must also be set as readable to be executed.	Allows a directory to be entered; e.g., <code>cd directory</code> .

# Permissions

Attribute	Files	Directories
r	Allows a file to be opened and read.	Allows a directory's contents to be listed if the execute attribute is also set.
w	Allows a file to be written to or truncated; however, this attribute does not allow files to be renamed or deleted. The ability to delete or rename files is determined by directory attributes.	Allows files within a directory to be created, deleted, and renamed if the execute attribute is also set.
x	Allows a file to be treated as a pro-gram and executed. Program files written in scripting languages must also be set as readable to be executed.	Allows a directory to be entered; e.g., <code>cd directory</code> .

# Permissions

File Attributes	Meaning
-rwx-----	A regular file that is readable, writable, and executable by the file's owner. No one else has any access.
-rw-----	A regular file that is readable and writable by the file's owner. No one else has any access.
-rw-r--r--	A regular file that is readable and writable by the file's owner. Members of the file's owner group may read the file. The file is world readable.
-rwxr-xr-x	A regular file that is readable, writable, and executable by the file's owner. The file may be read and executed by everybody else.
-rw-rw----	A regular file that is readable and writable by the file's owner and members of the file's owner group only.
lrwxrwxrwx	A symbolic link. All symbolic links have "dummy" permissions. The real permissions are kept with the actual file pointed to by the symbolic link.
drwxrwx---	A directory. The owner and the members of the owner group may enter the directory and create, rename, and remove files within the directory.
drwxr-x---	A directory. The owner may enter the directory and create, rename, and delete files within the directory. Members of the owner group may enter the directory but cannot create, delete, or rename files.

# Permissions

## **chmod – Change File Mode**

- ✓ To change the mode (permissions) of a file or directory, the **chmod** command is used.
- ✓ Only the file's owner or the superuser can change the mode of a file or directory.
- ✓ **chmod** supports two distinct ways of specifying mode changes: octal number representation and symbolic representation.

# Permissions

## Octal Representation

Octal	Binary	File Mode
0	000	---
1	001	--x
2	010	-w-
3	011	-wx
4	100	r--
5	101	r-x
6	110	rw-
7	111	rwx



# Permissions

## Symbolic Representation

Symbol	Meaning
u	Short for <i>user</i> but means the file or directory owner.
g	Group owner.
o	Short for <i>others</i> but means world.
a	Short for <i>all</i> ; the combination of <i>u</i> , <i>g</i> , and <i>o</i> .

Notation	Meaning
u+x	Add execute permission for the owner.
u-x	Remove execute permission from the owner.
+x	Add execute permission for the owner, group, and world. Equivalent to a+x.
o-rw	Remove the read and write permissions from anyone besides the owner and group owner.
go=rw	Set the group owner and anyone besides the owner to have read and write permission. If either the group owner or world previously had execute permissions, remove them.
u+x, go=rx	Add execute permission for the owner and set the permissions for the group and others to read and execute. Multiple specifications may be separated by commas.

# Permissions

## Umask - Set Default Permissions

- ✓ A umask of 022 allows only you to write data, but anyone can read data.
- ✓ A umask of 077 is good for a completely private system. No other user can read or write your data if umask is set to 077.
- ✓ A umask of 002 is good when you share data with other users in the same group. Members of your group can create and modify data files; those outside your group can read data file, but cannot modify it. Set your umask to 007 to completely exclude users who are not group members.

# Permissions

## Changing Identities

There are three ways to take on an alternate identity:

- ✓ Log out and log back in as the alternate user.
- ✓ Use the `su` command.
- ✓ Use the `sudo` command.

# Permissions

## Changing Identities

- ✓ From within your own shell session, the `su` command allows you to assume the identity of another user and either start a new shell session with that user's ID or issue a single command as that user.
- ✓ The `sudo` command allows an administrator to set up a configuration file called `/etc/sudoers` and define specific commands that particular users are permitted to execute under an assumed identity.
- ✓ The choice of which command to use is largely determined by which Linux distribution you use.
- ✓ Your distribution probably includes both commands, but its configuration will favor either one or the other.

# Permissions

## **chown – Change File Owner and Group**

- ✓ The **chown** command is used to change the owner and group owner of a file or directory. Superuser privileges are required to use this command.

==

- ✓ The syntax of **chown** looks like this:

**chown [owner][:group] file...**

- ✓ **chown** can change the file owner and/or the file group owner depending on the first argument of the command

# Permissions

## chown – Change File Owner and Group

Argument	Results
<code>bob</code>	Changes the ownership of the file from its current owner to user <i>bob</i> .
<code>bob:users</code>	Changes the ownership of the file from its current owner to user <i>bob</i> and changes the file group owner to group <i>users</i> .
<code>:admins</code>	Changes the group owner to the group <i>admins</i> . The file owner is unchanged.
<code>bob:</code>	Change the file owner from the current owner to user <i>bob</i> and changes the group owner to the login group of user <i>bob</i> .

# Processes

✓ Modern operating systems are usually multitasking, meaning that they create the illusion of doing more than one thing at once by rapidly switching from one executing program to another.

✓ The Linux kernel manages this through the use of processes. Processes are how Linux organizes the different programs waiting for their turn at the CPU.

# Processes

- ✓ `ps` – Report a snapshot of current processes.
- ✓ `top` – Display tasks.
- ✓ `jobs` – List active jobs.
- ✓ `bg` – Place a job in the background.
- ✓ `fg` – Place a job in the foreground.
- ✓ `kill` – Send a signal to a process.
- ✓ `killall` – Kill processes by name.
- ✓ `shutdown` – Shut down or reboot the system.



# Processes

- ✓ When a system starts up, the kernel initiates a few of its own activities as processes and launches a program called **init**. **init**, in turn, runs a series of shell scripts (located in /etc) called **init scripts**, which start all the system services.
- ✓ Many of these services are implemented as **daemon programs**, programs that just sit in the background and do their thing without having any user interface.
- ✓ So even if we are not logged in, the system is at least a little busy performing routine stuff.
- ✓ The fact that a program can launch other programs is expressed in the process scheme as a **parent process** producing a **child process**.

# Processes

- ✓ The kernel maintains information about each process to help keep things organized. For example, each process is assigned a number called a process ID (PID).
- ✓ PIDs are assigned in ascending order, with init always getting PID 1.
- ✓ The kernel also keeps track of the memory assigned to each process, as well as the processes' readiness to resume execution. Like files, processes also have **owners** and **user IDs, effective user IDs**, and so on.

# Viewing Processes with ps

✓ command to view processes (there are several) is ps.

=

# Viewing Processes Dynamically with top

- ✓ The top program displays a continuously updating (by default, every 3 seconds) display of the system processes listed in order of process activity.

# Controlling Processes

- ✓ Interrupting Process – `ctrl-c`
- ✓ Putting a Process in the Background – `bg %[process no], &`
- ✓ Stopping ( Pausing a process ) – `ctrl-z`
- ✓ Returning Process to foreground – `fg %[process no]`

# Controlling Processes

## ✓ Signals

Number	Name	Meaning
1	HUP	<p>Hang up. This is a vestige of the good old days when terminals were attached to remote computers with phone lines and modems. The signal is used to indicate to programs that the controlling terminal has “hung up.” The effect of this signal can be demonstrated by closing a terminal session. The foreground program running on the terminal will be sent the signal and will terminate.</p> <p>This signal is also used by many daemon programs to cause a reinitialization. This means that when a daemon is sent this signal, it will restart and reread its configuration file. The Apache web server is an example of a daemon that uses the HUP signal in this way.</p>
2	INT	Interrupt. Performs the same function as the ctrl-C key sent from the terminal. It will usually terminate a program.
9	KILL	Kill. This signal is special. Whereas programs may choose to handle signals sent to them in different ways, including by ignoring them altogether, the KILL signal is never actually sent to the target program. Rather, the kernel immediately terminates the process. When a process is terminated in this manner, it is given no opportunity to “clean up” after itself or save its work. For this reason, the KILL signal should be used only as a last resort when other termination signals fail.
15	TERM	Terminate. This is the default signal sent by the kill command. If a program is still “alive” enough to receive signals, it will terminate.
18	CONT	Continue. This will restore a process after a STOP signal.
19	STOP	Stop. This signal causes a process to pause without terminating. Like the KILL signal, it is not sent to the target process, and thus it cannot be ignored.

# Sending Signals to Multiple Processes with killall

- ✓ It's also possible to send signals to multiple processes matching a specified program or username by using the killall command.
- ✓ Here is the syntax:
- ✓ `killall [-u user] [-signal] name...`

# Process

Command	Description
<code>ps tree</code>	Outputs a process list arranged in a tree-like pattern showing the parent/child relationships between processes.
<code>vmstat</code>	Outputs a snapshot of system resource usage including memory, swap, and disk I/O. To see a continuous display, follow the command with a time delay (in seconds) for updates (e.g., <code>vmstat 5</code> ). Terminate the output with <code>ctrl-C</code> .
<code>xload</code>	A graphical program that draws a graph showing system load over time.
<code>tload</code>	Similar to the <code>xload</code> program, but draws the graph in the terminal. Terminate the output with <code>ctrl-C</code> .



# Environment

- ✓ The shell maintains a body of information during our shell session called the environment.
- ✓ Data stored in the environment is used by programs to determine facts about our configuration.
- ✓ While most programs use configuration files to store program settings, some programs will also look for values stored in the environment to adjust their behavior.

# Environment

- ✓ `printenv` – Print part or all of the environment.
- ✓ `set` – Set shell options.
- ✓ `export` – Export environment to subsequently executed programs.
- ✓ `alias` – Create an alias for a command.

# Environment

- ✓ The shell stores two basic types of data in the environment, although, with bash, the types are largely indistinguishable.
- ✓ They are environment variables and shell variables.
- ✓ Shell variables are bits of data placed there by bash, and environment variables are basically everything else.
- ✓ In addition to variables, the shell also stores some programmatic data, namely aliases and shell functions