

# Linux File System

## Operating System Lab Spring 2015

Roya Razmnoush

CE & IT Department, Amirkabir University of Technology



# File Management



# Introduction

- Filesystem
  - How are data stored in storage?
  - How do users access the data?
    - Data organization, files and directories
- Filesystem types
  - Disk FS: ext2, ext3, FAT, FAT32 & NTFS
  - Network FS: Samba & NFS
  - Flash FS: JFFS2
  - Special FS: proc FS

# Introduction (cont'd)

- You should understand Linux FS
  - Why?
- Everything in Linux **is file**, if it is NOT process
  - Easy to use
    - Open file, read/write and close the file
- Unlike Windows, Linux FS is standard FS
  - Everyone should learn standards

# What is File System?

It is responsible for storing information on disk and retrieving and updating this information.

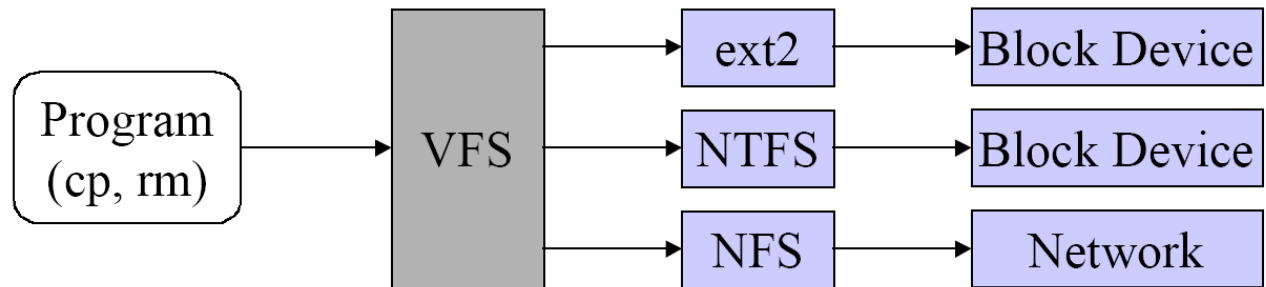
Example :

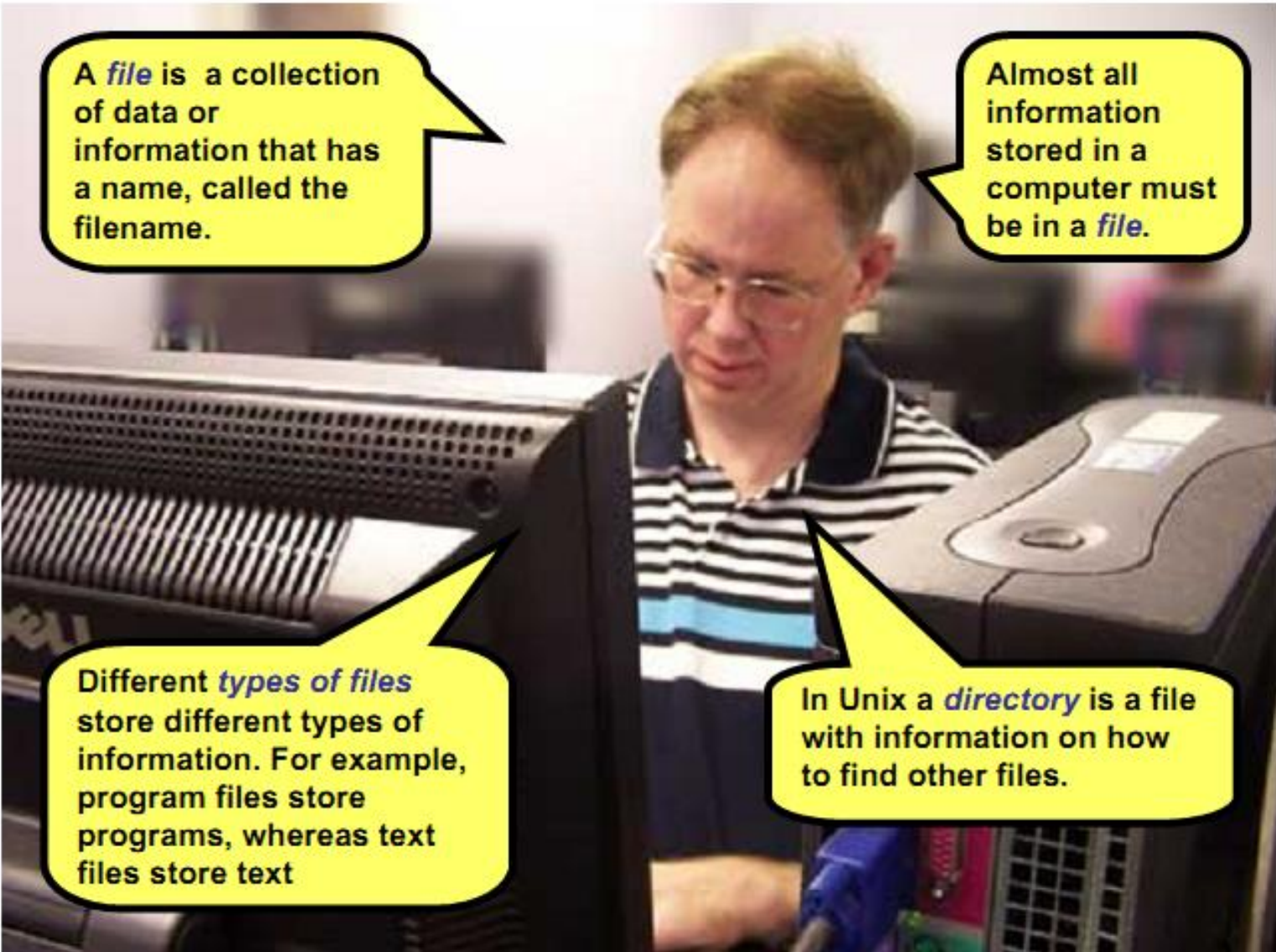
FAT16, FAT32, NTFS

ext2, ext3

...

In Linux everything is file.





A *file* is a collection of data or information that has a name, called the filename.

Almost all information stored in a computer must be in a *file*.

Different *types of files* store different types of information. For example, program files store programs, whereas text files store text

In Unix a *directory* is a file with information on how to find other files.



# Type of the File System

## Network File System

NFS

SMB

## Disk File System

ext2

ext3

FAT32

NTFS



# Network File System

Network File System are physically somewhere else, but appear as if they are mounted on one computer.

## NFS

It was developed by Sun.

## SMB

It was developed by Microsoft.





# Disk File System

Disk File System are what you will find on a physical device, such as hard drive in a computer.



# ext2 file system

It has been the standard File System for Linux.  
The original Extended File System was named ext.

The ext2 File System can accommodate:

- Files as large as 2GB

- Directories as large as 2TB

- Max. file name length of 255 characters.



# Hard Disk Partitions

Disk divided into partitions

Definition of partition: group of adjacent cylinders

1 file system may reside in a single partition

BIOS defines boot sector to be head 0, cylinder 0, sector 1

*Master Boot Record* (MBR) – used to boot computer

Partition table

Start and end addresses of each partition

One partition is marked as active

When computer boots up:

BIOS executes MBR

MBR locates active partition and executes boot block



# Hard Disk Partitions

MBR	Partition table	Partition	Partition	Partition
-----	-----------------	-----------	-----------	-----------

Boot block	Superblock	Free space management	Inodes	Root dir.	Files & directories
---------------	------------	--------------------------	--------	-----------	------------------------

Each partition:

Starts with boot block even if it does not contain bootable OS

*Superblock* – key parameters about file system in partition such as magic identifying file system type, number of blocks Bitmap or linked-list of

free blocks – free space management

Inodes – array of these (one per file)

Root directory of partition

Files and directories at the end



# ex2 Structure

A file in the ext2 File System begins with the inode.  
inode

Each file has an inode structure that is identified by an i-number.

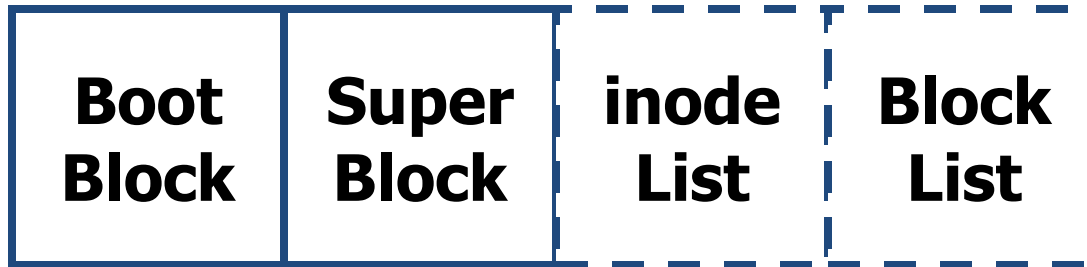
The inode contains the information required to access the file.

It doesn't contain file name.

Inodes store information on files, such as  
user and group ownership,  
access mode (read, write, execute permissions)  
and type of file.



# Physical Structure on the Disk



- Boot Block : information needs to boot the system
- Super Block : File System Specifications
  - Size
  - Max. number of files
  - Free blocks
  - Free inodes
- inode List
- Block List : The files data



# Symbolic Link

Because of the structure of the ex2 File System, several names can be associated with a single file.

In effect, you create another inode that reference already existing data.



# ex3 File System

It is as same as ext2.

It is a journaling File System for Linux.

In a journaling system, metadata is written to a journal on the disk before it is actually used to modify the file.



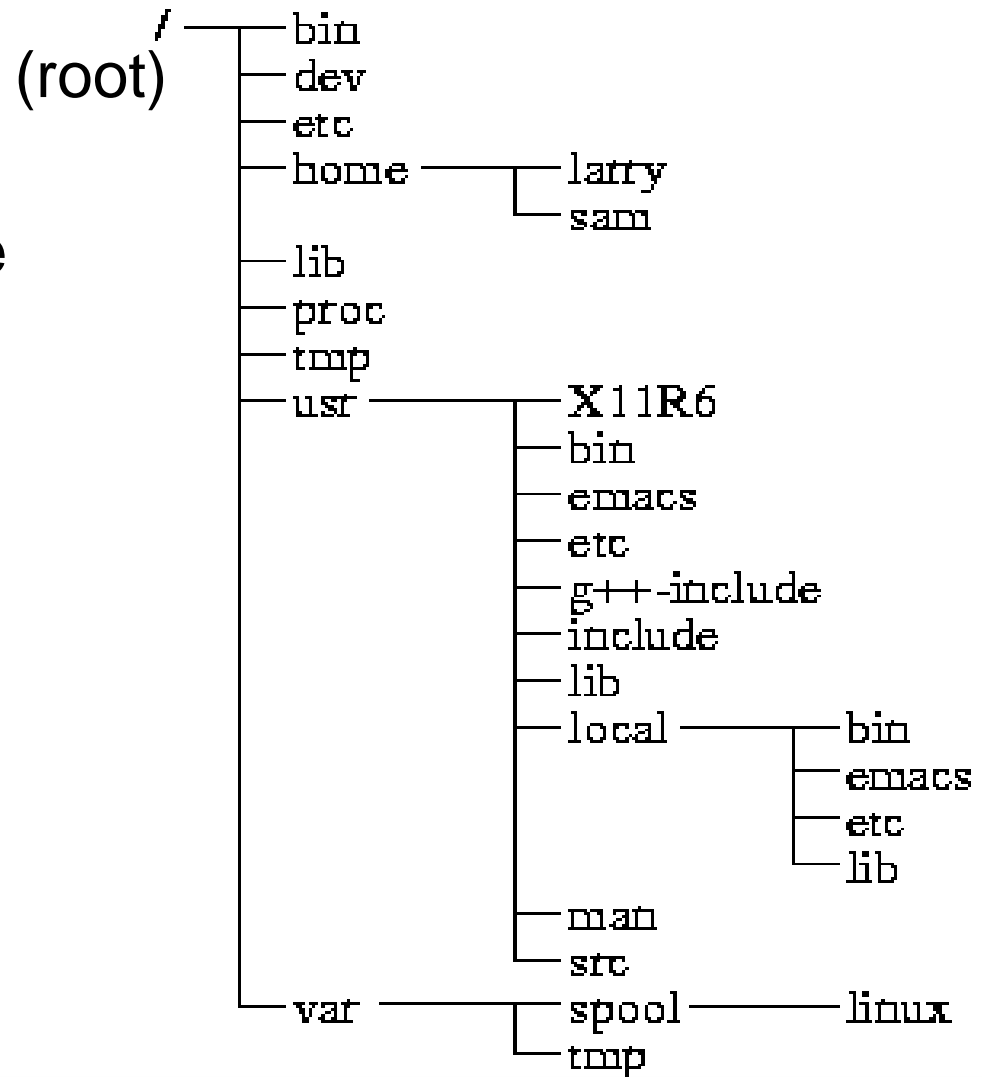


# FHS

- There is not any drive C:, D:, ...
- All directories are under “/”
  - “/” is the root directory
- It is possible
  - to have multiple partitions
  - to multiple filesystems

# Directory Tree

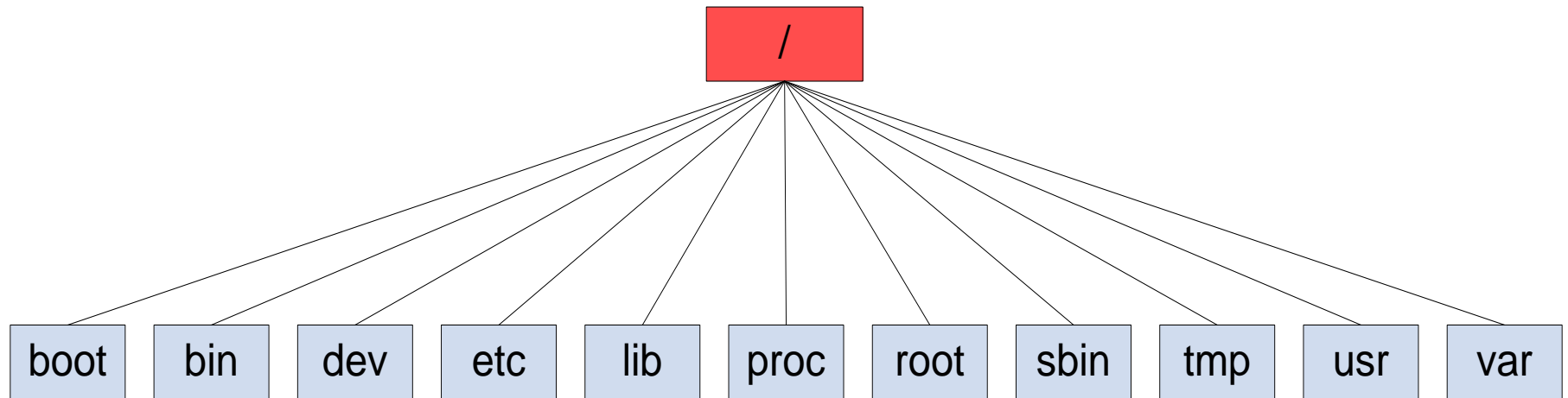
When you log on the the Linux OS using your username you are automatically located in your home directory.



# The “/”

- The primary hierarchy in FSH
  - The root of tree of filesystem
- All paths start form here
- There is only one “/” in filesystem

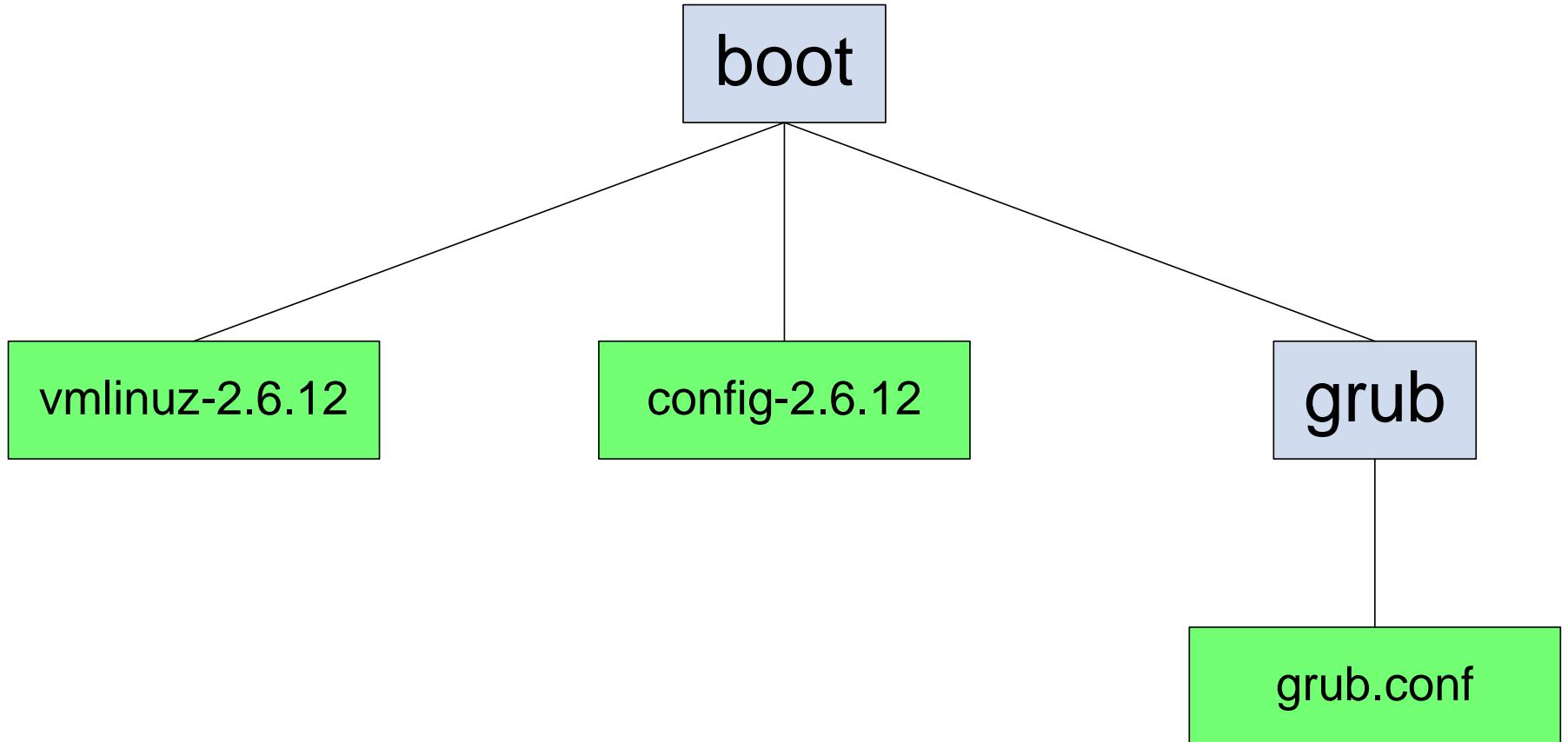
# The “/”



# boot

- **/boot** : The files necessary for the system to boot. Not all Linux distributions use this one. Fedora does.
- Linux kernel
- Boot loader configuration
- If you lost **boot**
  - You cannot boot your OS

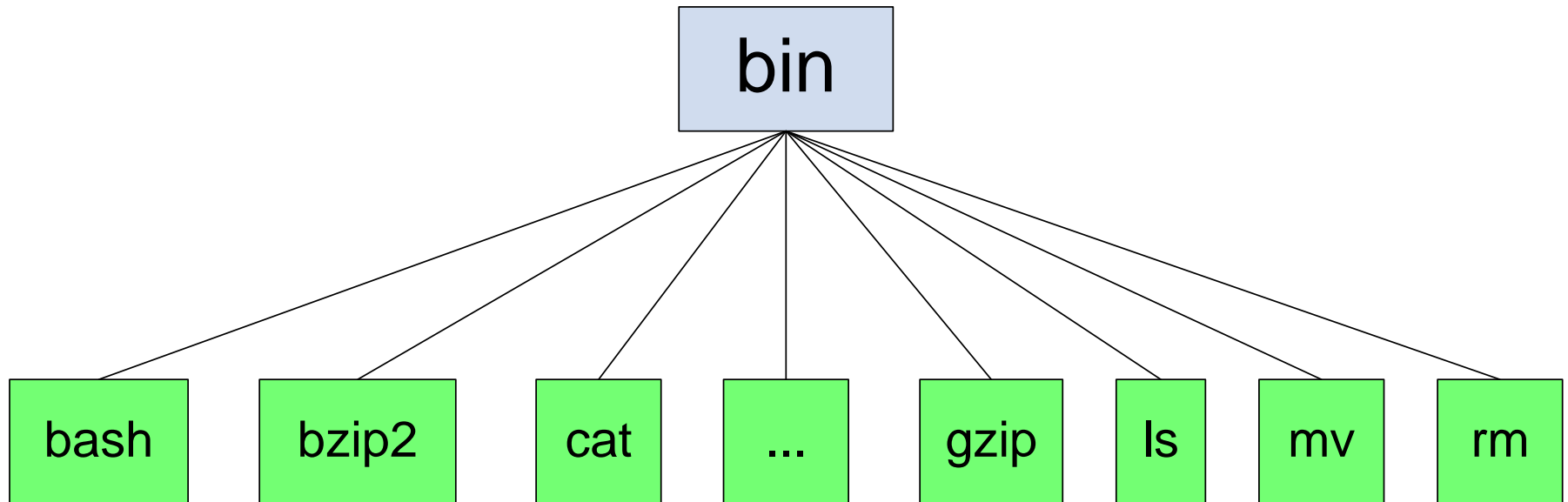
# boot



# bin

- **/bin** : Important Linux commands available to the average user.
- Essential programs
- Need for system startup
- Basic commands for
  - Navigating in filesystem
  - File management

# bin

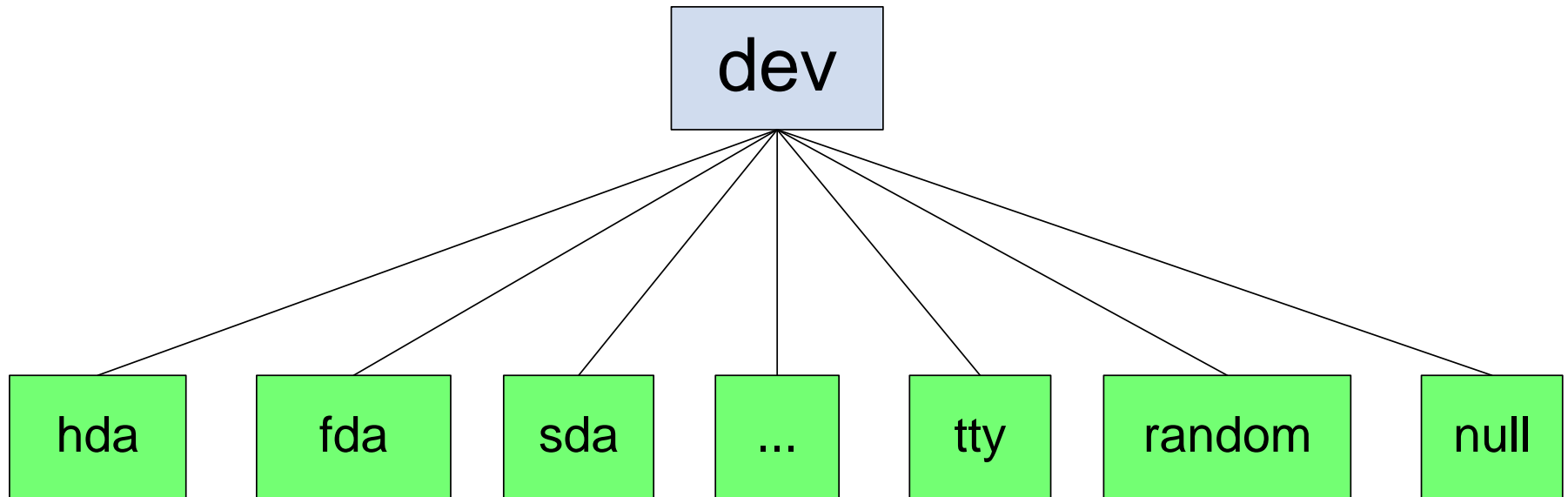




# dev

- **/dev** : All device drivers. Device drivers are the files that your Linux system uses to talk to your hardware. For example, there's a file in the /dev directory for your particular make and model of monitor, and all of your Linux computer's communications with the monitor go through that file.
- Everything is file
  - Hardware components (devices) are file
    - Hard disk
    - Key board
- All device files are here
- Direct interaction with device driver
  - Open the device file
  - Read & Write

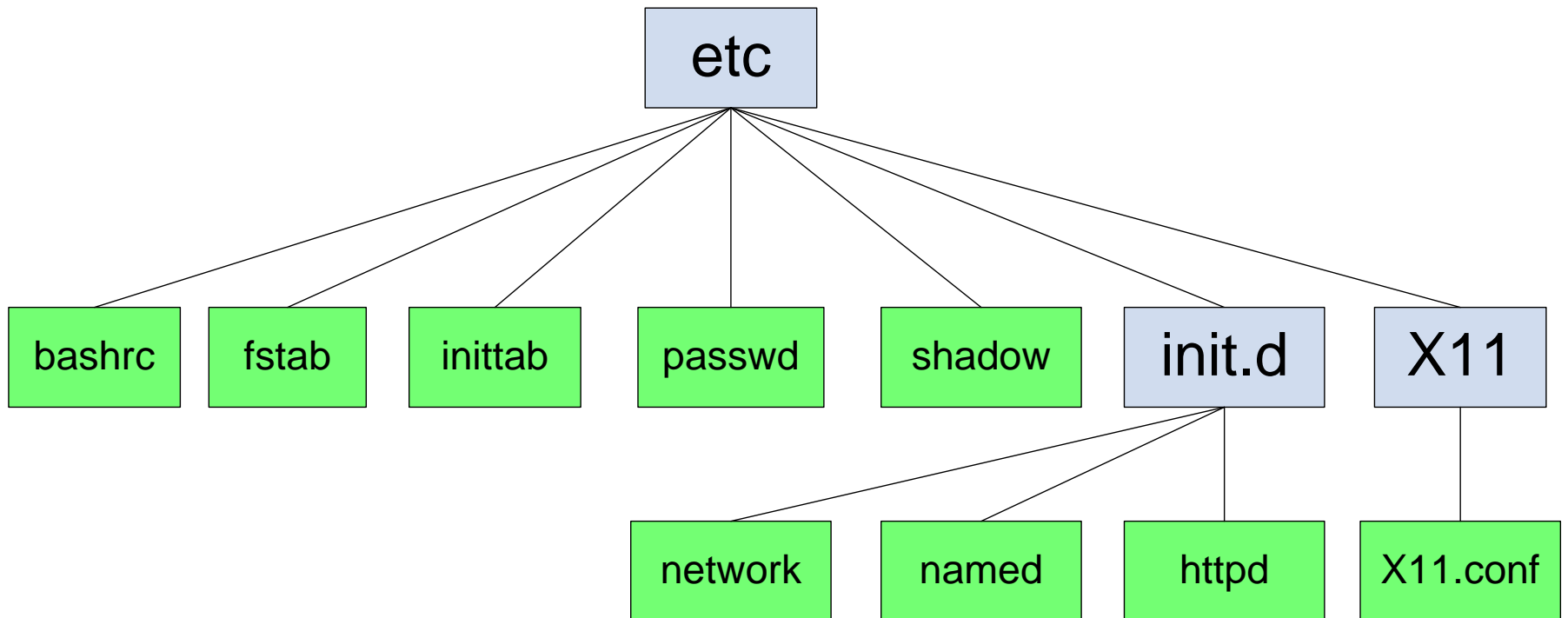
# dev



# etc

- **/etc** : System configuration files.
- System configuration directory
  - What is done by the registry in Windows
- All configuration file are text files
  - You can view and edit it manually

# etc



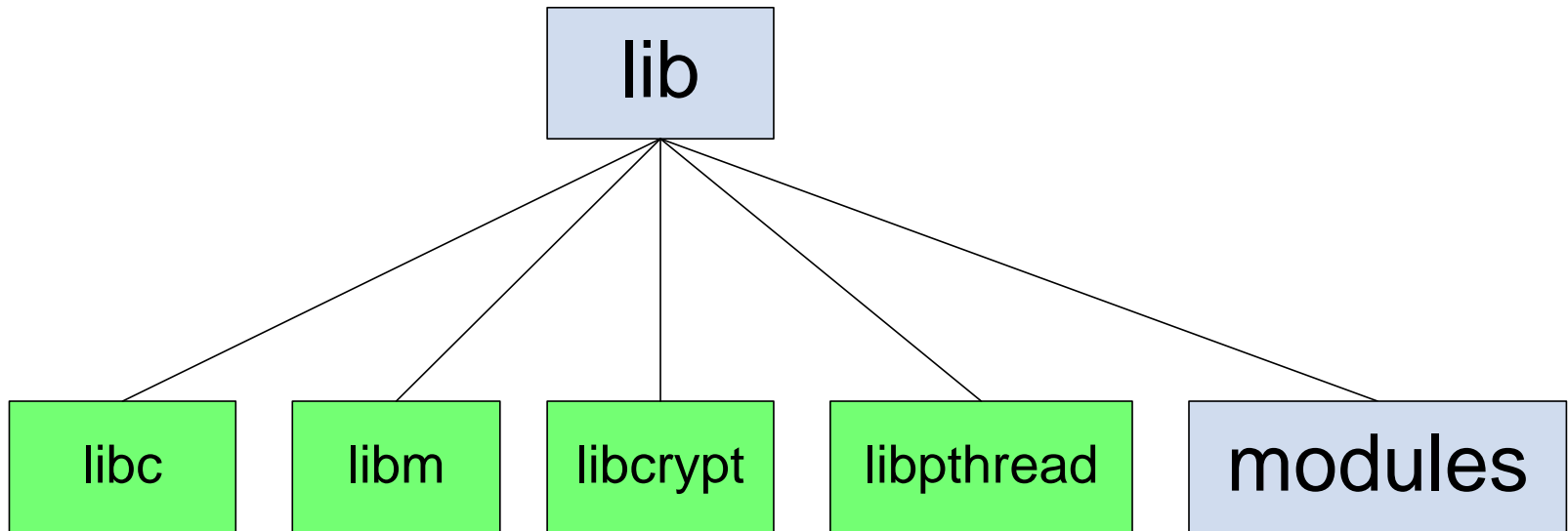
# home

- **/home** : Every user except root gets her own folder in here, named for her login account. So, the user who logs in with linda has the directory /home/linda, where all of her personal files are kept.
- Home directory of user
- Each user has a directory
  - /home/bahador
  - /home/hamed
- All files of user are stored here

# lib

- **/lib** : System libraries. Libraries are just bunches of programming code that the programs on your system use to get things done.
- Programs need libraries
  - Dynamically linked libraries
- Programmers need libraries
- All essential libraries are here
  - Needed for system startup

# lib

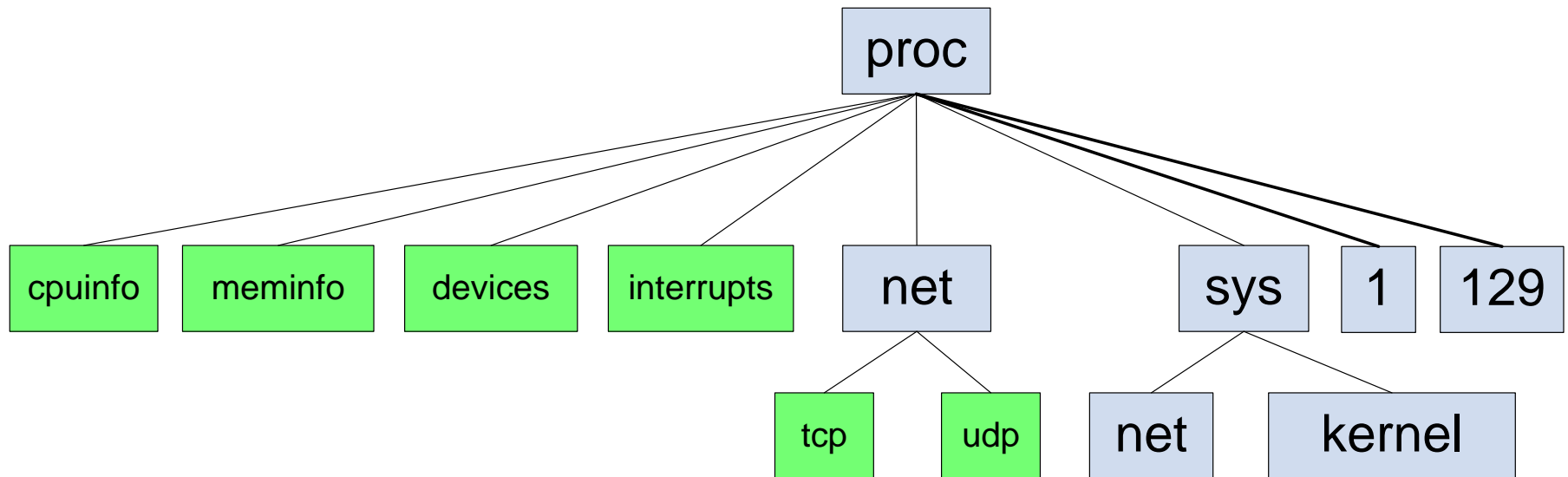


# proc

- Kernel's interface
  - Kernel pseudo-directory
- Special directory
  - It is **NOT** a directory on hard disk
- Kernel Configuration
- Kernel State monitoring



# proc



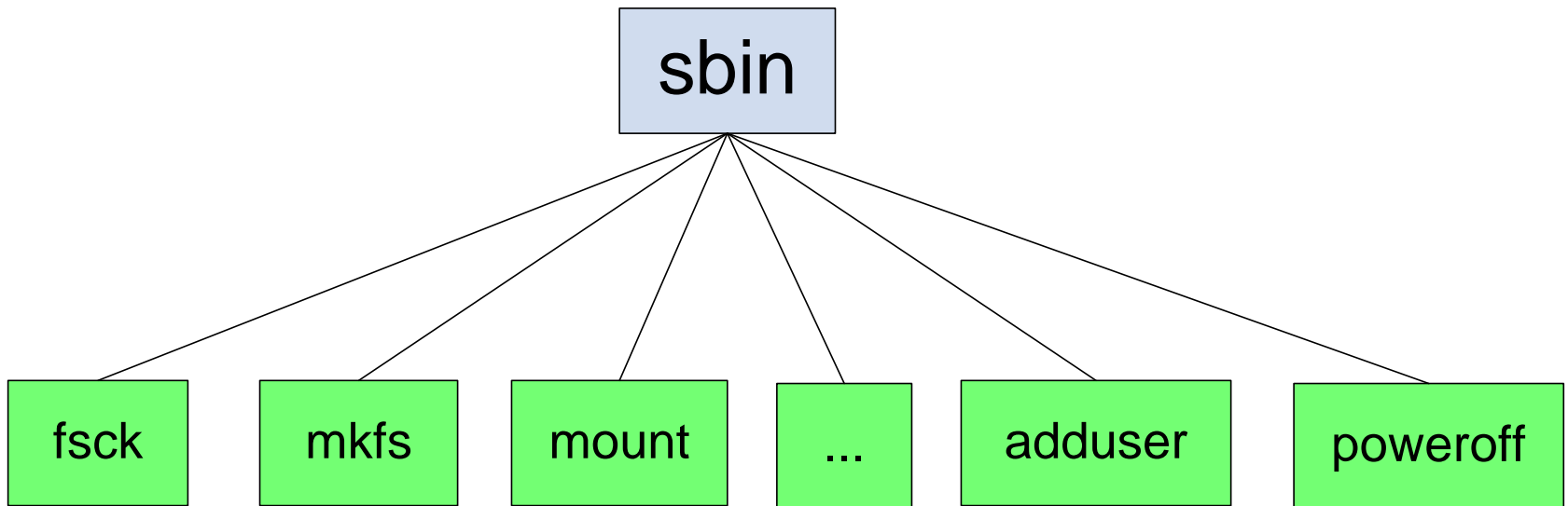
# root

- **/root** : The root user's home directory.
- Home directory of root
- Don't confuse
  - / is the “root of Filesystem”
  - root is the name of system admin
  - /root is the admin

# sbin

- **/sbin** : Essential commands that are only for the system administrator.
- System configuration programs
  - Format hard disk
  - Manage hardware
- Only “root” can run the programs

# sbin



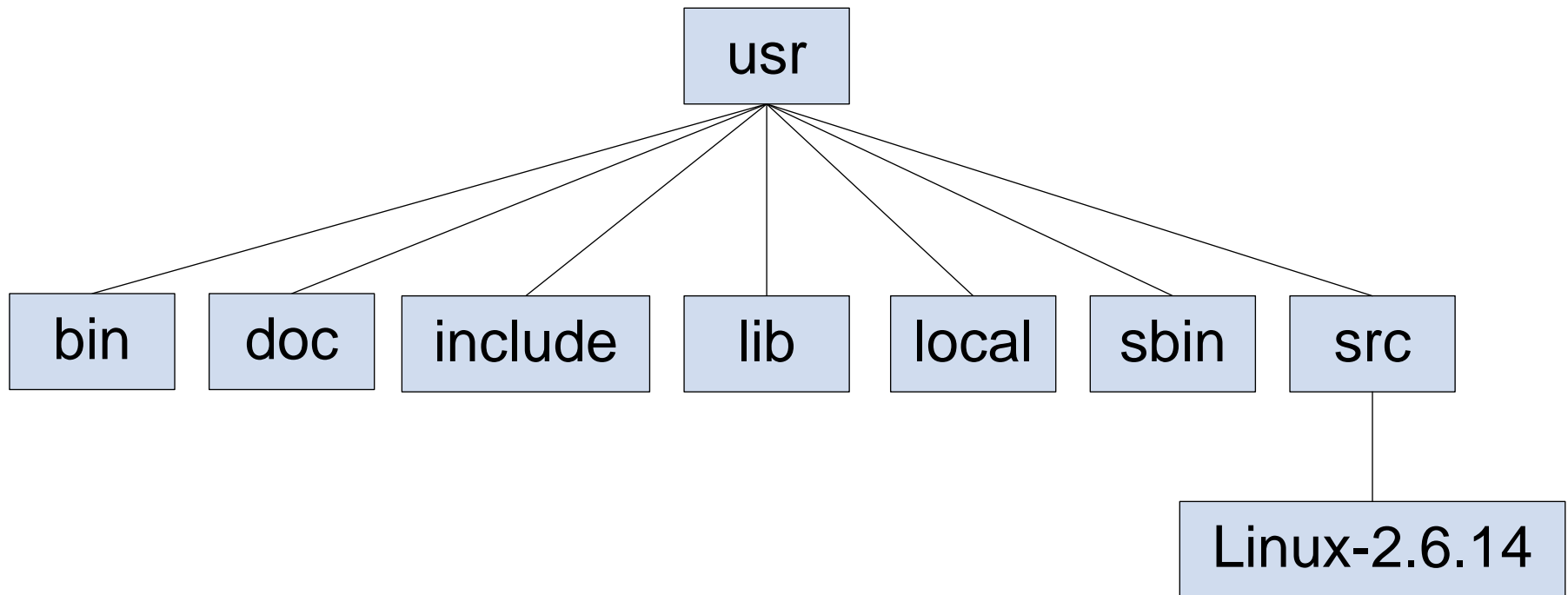
# tmp

- **/tmp** : Temporary files and storage space. Don't put anything in here that you want to keep. Most Linux distributions (including Fedora) are set up to delete any file that's been in this directory longer than three days.
- Temporary directory
- All temp files are created by programs
- Your temp files
- It is emptied regularly

# usr

- **/usr** : Programs and data that can be shared across many systems and don't need to be changed.
- Secondary hierarchy
- Very useful programs
  - We usually use them
    - compiler, tools
- Are not essential for system startup

# usr

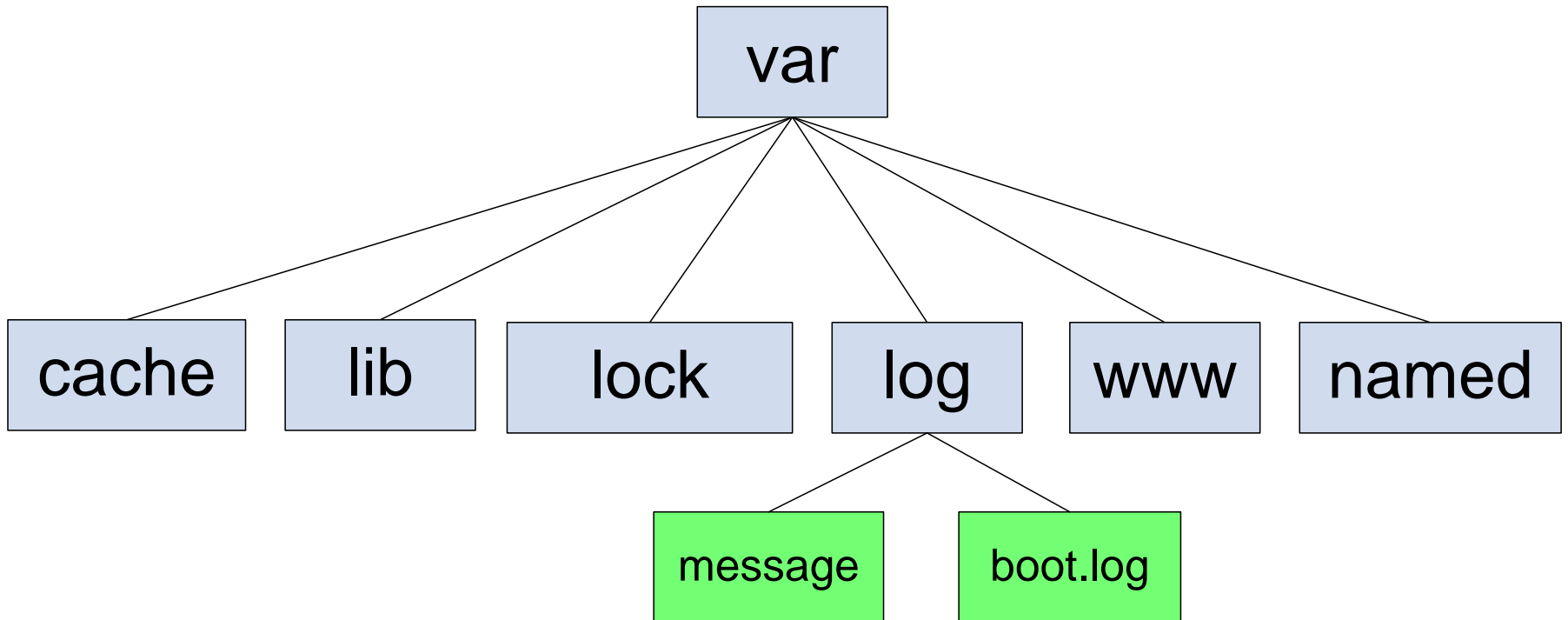


# var

- **/var** : Data that changes constantly (log files that contain information about what's happening on your system, data on its way to the printer, and so on).
- The variable directory
- All dynamic files
- User cannot change the files



# var



# /home and /root

- The /home directory is where all the home directories for all the users on a system are stored.
- The /root directory is where all the home directories for root user on a system are stored.

# Other directories

- /mnt
  - removable media such as CD-ROM, floppy and ... are mounted.
  - /mnt/floppy
  - /mnt/cdrom
  - **/mnt** : Mount points. When you temporarily load the contents of a CD-ROM or USB drive, you typically use a special name under /mnt. For example, many distributions (including Fedora) come, by default, with the directory /mnt/cdrom, which is where your CD-ROM drive's contents are made accessible.

# Basic Shell



# The shell

- The Shell executes programs.
  - User types command
  - Shell reads command (read from input) and translates it to the operating system.
  - Shell types: Bash, csh, ksh, sh
- usually `bash` is Linux default shell

# Different type of shells

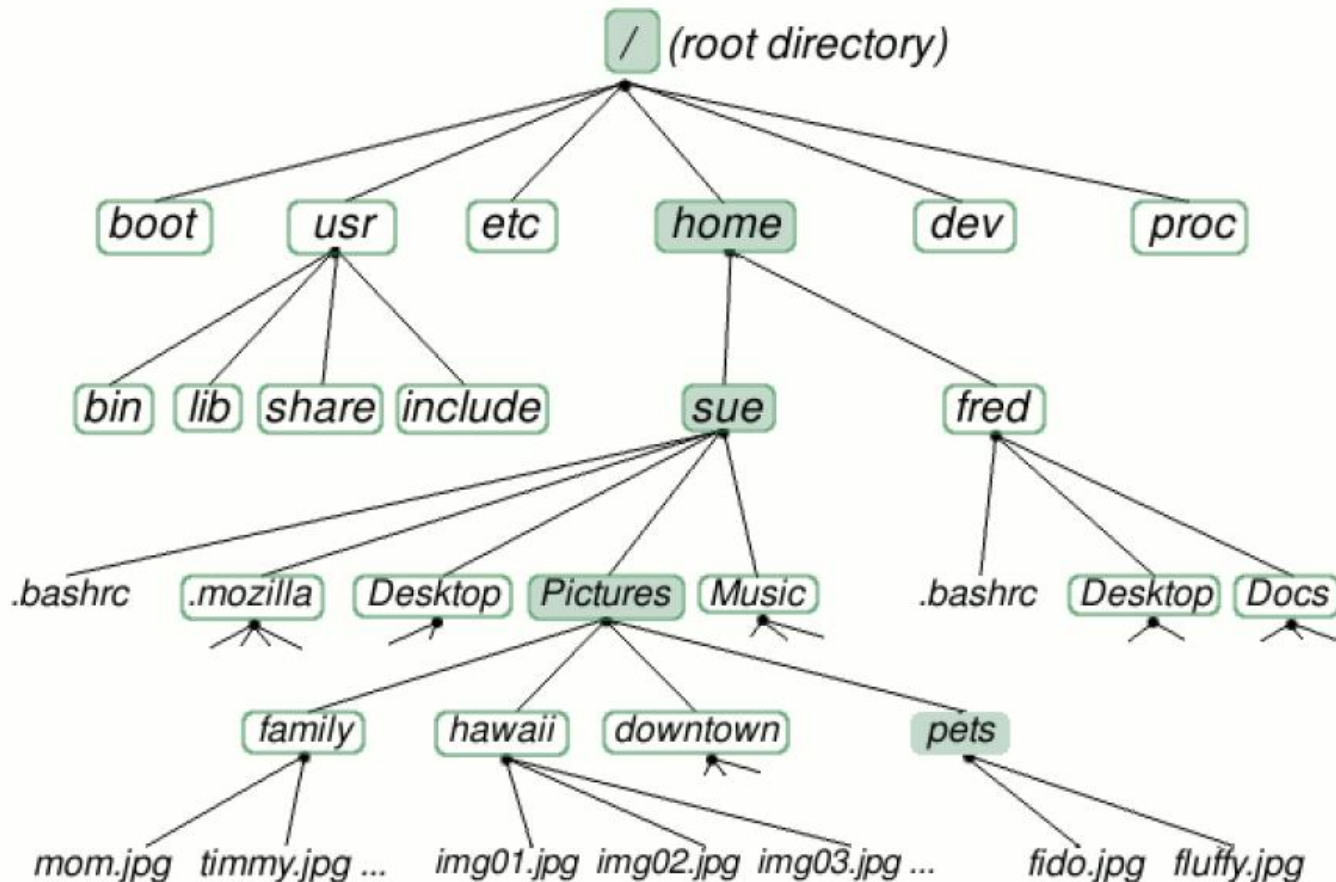
Shell Name	Developed by	Where	Remark
BASH ( Bourne-Again SHell )	Brian Fox and Chet Ramey	Free Software Foundation	Most common shell in Linux. It's Freeware shell.
CSH (C SHell)	Bill Joy	University of California (For BSD)	The C shell's syntax and usage are very similar to the C programming language.
KSH (Korn SHell)	David Korn	AT & T Bell Labs	--
TCSH	See the man page. Type \$ man tesh	--	TCSH is an enhanced but completely compatible version of the Berkeley UNIX C shell (CSH).

- To view shells
  - \$ cat /etc/shells
  - \$ echo \$SHELL

# The prompt

- The `[username@server current_directory]$` signifies that
  - this console is being used by user `username`,
  - and the host-name is `server`.
  - The second word is the current working directory

# Linux Directory Hierarchy





# Current Working Directory

- Every process has a location in the directory, termed its **current working directory**.
- When you log into a Linux system, your shell starts off in a particular directory called **home directory** (ex. `/home/rinaldi/`).
- Each file or directory has a unambiguously name specified by its **pathname** relative to `/` the **root directory**.
- A pathname relative the root directory is often termed an **absolute pathname** (ex. `/usr/src/linux/CREDITS`).
  - `$ cd /usr/src/linux/CREDITS`

# Absolute and relative pathname

- A file can be unambiguously specified by using a pathname relative to its current working directory.
  - Example: if `/usr/src/` is the current working directory, `/linux/CREDITS` is the relative pathname of `CREDITS`
- The file system provides the following special fields that may be used supplying a relative path:
  - `.`                      - current directory
  - `..`                     - parent directory
  - `~`                      - your home directory
  - Example:
    - `cd ..`
    - `cd game`
    - `./fortune`

# Commands

- General command syntax format:

`$ command -options arguments`

- Example:

– `$ clear`

– `$ cd /usr/src/linux`

– `$ wc -w file1` (number of words in file1)

– `$ cat file1 file2 file3`

– `$ ls -al`

- history: see command history

# Options

- An option is an argument that modifies the effects of a command.
- Most utilities require you to prefix options with a single hyphen. GNU program options are frequently preceded by two hyphens in a row.
- Multiple single-letter options can be combined into one argument that starts with a single hyphen
  - thus `ls -xr` produces the same results as `ls -rx` and `ls -x -r`
- a `—` argument (two consecutive hyphens) indicates the end of the options

# Elementary Troubleshooting

- If a command causes an error message like:
  - `-bash: <command>: Command not found`

then the first step is:

## 1. Did you type it all right?

- **REMEMBER:** LINUX is **CASE SENSITIVE**. Uppercase and lowercase are evaluated differently.

## 2. Check whether the executable for the command has been installed with the `whereis` command.

- As an example, let us apply `whereis` for `useradd`:

```
$ whereis useradd
```

```
bash: useradd: /usr/sbin/useradd
```

```
/usr/share/man/man8/useradd.8.bz2
```

- So the command is present, but in a different directory. You can execute this command by typing:

```
$ /usr/sbin/useradd
```

# Elementary Troubleshooting

3. If step 2 (i.e., `$ /usr/sbin/useradd`) does not work, then perhaps the command could requires you to be a super-user.

4. If step 2 results in no entries,

```
$ latex
```

```
-bash: latex: command not found
```

```
$ whereis latex
```

```
latex:
```

then you must install the package. Insert the Linux CD, and install the package with the **rpm** command.

- There are some several graphic tools for the installation of the packages (**rpmdrake**, **Kpackage**, **Gnome RPM**, ...)

# Control characters

- Interrupting
  - ^C interrupts.
    - Exits the program and returns you to the command-line prompt.
  - ^Z suspends.
    - Stops the program and puts it in the background. Type *fg* to restart it.
  - ^D end of file.
- Complete word : you can complete a word by using TAB

# Directories Operations





# ls command

- `ls [options]..[pathnames]`
- List information about files

Options	Means
<code>-a , --all</code>	Do not hide entries starting with .
<code>-d, --directory</code>	List directory entries instead of contents
<code>-l</code>	Long listing format
<code>-s, --size</code>	Print size of each file, in blocks
<code>-k, --kilobytes</code>	Use 1kbyte blocks
<code>-1</code>	list one file per line
<code>-R</code>	Includes the contents of subdirectories.
<code>-Z</code>	Selinux related information

# ls options

- ls options
  - -a all files including hidden files
  - C column list sorted down
  - F adds / for directory; \* for executable; @ for symbolic links
  - m across page; comma separators
  - p adds / for directory; \* for executable
  - r reverse alphabet order
  - R recursive; includes subdirectories
  - t list in time date last modified
  - u lists in time date last accessed
  - x column list sorted across page
  - i inode of each file

# ls command (cont.)

- The result of executing `ls -l`

Permissions	Directories	user	Group	Size	Date	Directory or File
drwxr-xr-x	4	amir	amir	4096	2008-12-26 01:22	Desktop
drwxr-xr-x	2	amir	amir	4096	2008-12-26 01:22	Documents
drwxr-xr-x	2	amir	amir	4096	2008-12-26 01:22	Download
drwxr-xr-x	2	amir	amir	4096	2008-12-26 01:22	Public

- **Permissions** - The permissions of the directory or file.
- **Directories** - The amount of links or directories within the directory. The default amount of directories is going to always be 2 because of the `.` and `..` directories.
- **Group** - The group assigned to the file or directory
- **Size** - Size of the file or directory.
- **Date** - Date of last modification.
- **Directory of file** - The name of the file or file.

# ls command (cont.)

- Here is the list of most common colors:
  - Normal file : normal
  - Directory : blue
  - Compressed files : red
  - Image files : Magenta
- The color setting can be changed in `/etc/DIR_COLORS`

# cd command

- `cd [directory name]`
  - `[oslab@localhost~] cd Desktop`
  - `[oslab@localhost Desktop ]`
- To come back the home directory use
  - `$ cd`
- `pwd` – print working directory
  - `Pwd => /home/oslab/Desktop`

# cp command

- `cp [options] source .. destination`
- Copy one or multiple sources to destination
- If destination does not exist, then `cp` creates it; otherwise `cp` overwrites it
- If destination is a directory, `cp` makes a copy of source in the directory

Options	Means
<code>-f, --force</code>	remove existing destinations, never prompt
<code>-R, --recursive</code>	copy directories recursively
<code>-v, --verbose</code>	explain what is being done

# rm command

- `rm [options] file`

Options	Means
<code>-f, --force</code>	remove existing destinations, never prompt
<code>-R, --recursive</code>	remove directories recursively
<code>-v, --verbose</code>	explain what is being done

- If the directory is not empty, to remove it we must use
  - `$ rm -r dir_name`
- If `-i` option is used, shell displays the name of the file and then waits for you to respond with **y (yes) before it deletes the file.**
- `rmdir [options] folder(s)`: this command will only work if the folders are empty.

# mv command

- `mv [options] source .. destination`

Options	Means
<code>-f, --force</code>	remove existing destinations, never prompt
<code>-v, --verbose</code>	explain what is being done

- Examples of mv command
  - `$ mv chap[1,3,7] book` (moves files chap1, chap3, and chap7 to directory book)
  - `$ mv chap[1-5] book` (moves files chap1 to chap5 to directory book)



# Mkdir command

- `mkdir [options] dir_name` (creates a directory)

# Files Operations



# finding-things utilities

- basic Linux finding-things utilities:
  - locate, find, and grep.
- Locate and find search for files, and grep searches for text in files.

# locate

- Locate is the easiest. It maintains its own database of files on your system, so it's blazing fast.
- `locate [options] name(s)`
- following command would display only 15 results in a search for files that have an *.html* extension:

```
locate -n 15 "*.html"
```

# find

- The find command searches a folder hierarchy for filename(s) that meet a desired criteria: Name, Size, File Type
- `find <path> <regular expression>`
  - name, size, time, type, permission, ...
  - `find /etc -name samba`
  - List all filenames ending in .mp3, searching in the current folder and all subfolders:  
`$ find . -name "*.mp3"`
  - List all filenames ending in .mp3, searching in the music folder and subfolders:  
`$ find ./music -name "*.mp3"`

# grep

- The grep utility searches through one or more files to see whether any contain a specified string of characters (pattern). Then it displays the lines that containing the pattern on the console.
- `grep [options] "pattern" [filename]`

```
$ cat memo
```

```
Helen:
```

```
In our meeting on June 6 we  
discussed the issue of credit.  
Have you had any further thoughts  
about it?
```

```
Alex
```

```
$ grep 'credit' memo  
discussed the issue of credit.
```

# which

- Shows the full path of (shell) commands.
- Which takes one or more arguments. For each of its arguments it prints to stdout the full path of the executables that would have been executed when this argument had been entered at the shell prompt.
- It only searches for an executable or script in the directories listed in the environment variable PATH
- `> which q2 ~/bin/q2`

# whereis

- whereis - locate the binary, source, and manual page files for a command.
- The **whereis** utility searches for files related to a utility by looking in standard locations.
  - Use whereis if you want to find a binary file, a man page, for a command
- If you wanted to find out if you have the 'gedit' editor and where it is, you would type
  - **whereis gedit**



# File manipulating Operations



# 'file' command

- file - determine file type.
- file file\_name ( Tests the Contents of a File without having to open and examine the file yourself. )

# Commands for showing file contents

- cat
- less
- more
- head
- tail

# Listing a file

- `$ cat filename` (displays the contents of `filename`)
- `$ more filename` (displays first screen of `filename`, use space bar to scroll up one screen, quits automatically after last screen)
- `$ less filename` (displays first screen of `filename`, use space bar to scroll up one screen, type b to scroll down one screen, use up and down arrows to move up or down one line, need to type `q` or `Q` to exit less command)
- `$ head -n filename` (displays the first `n` lines of `filename`. If `n` is not specified, it defaults to 10)
- `$ tail -n filename` (displays the last `n` line of `filename`. If `n` is not specified, it defaults to 10)

# Cat command

- Simple usage
  - cat [filename]
    - Concatenate source files and display them
  - cat > [filename]
    - Create or override filename
    - Ctrl+D: EOF
  - cat >> [filename]
    - append
  - cat -n [filename]
    - Add line number

# Regular Expressions

# Wild-cards

- `$ ls -l *.c`
- `$ ls [abc]*`
- `$ ls ?a*`
- **\*** means 'match any number of characters'.
  - For example, **chap\*** matches: *chap01*, *chapa*, *chap\_end*, and also *chap*.
  - If you just give **\*** (nothing else), it matches every file.
- **?** means 'match any single character'.
  - For example, **chap?** matches: *chapa* and *chap1*, but not *chap01* and *chap*.

# Wild-cards

- `[..]` means 'match any one characters between the brackets'. A range of characters may be specified by separating a pair of characters by a dash.
  - For example, `chap[abc]` matches: *chapa* and *chapc*, but not *chap1* and *chapab*.
  - The following command line prints 39 files, part 0 through part 38:  

```
$ lpr part[0-9] part[12][0-9] part3[0-8]
```
  - `[A-Za-z]*` matches with any word whose first element is a character



# Wild-cards

- optional When an exclamation point (!) or a caret (^) immediately follows the opening bracket ([) that defines a character class, the string enclosed by the brackets matches any character not between the brackets. Thus `[^ab]*` matches any filename that does not begin with a or b.

```
$ ls
```

```
aa ab ac ad ba bb bc bd cc dd
```

```
$ ls *[^ab]
```

```
ac ad bc bd cc ddcc dd
```

```
$ ls [b-d]*
```

```
ba bb bc bd cc dd
```

# File Archives and compression

# Compression

- Compression: process in which files are reduced in size by stripping out characters
- Compression algorithm: standard set of instructions used to compress a file
- Compression ratio: percentage by which the file size was decreased
- Common compression utilities include compress, gzip, and bzip2

# The gzip Utility

- ◎ GNU zip (gzip): used to compress files using the Lempel-Ziv compression algorithm (LZ77)
  - Varies slightly from algorithm used by compress
  - Average compression ratio of 60-70%
  - Uses .gz filename extension by default
  - Can control level of compression via numeric option
- ◎ `gunzip` command: used to decompress .gz files

# The gzip Utility (continued)

Option	Description
-#	Specifies how thorough the compression will be, where # can be any number between 1 and 9. The option -1 represents fast compression, which takes less time to compress but results in a lower compression ratio. The option -9 represents thorough compression, which takes more time but results in a higher compression ratio.
--best	Results in a higher compression ratio; same as the -9 option.
-c --stdout -to-stdout	Displays the contents of the compress file to SO (same function as the zcat command) when used with the gunzip command.
-d - -decompress - -uncompress	Decompresses the files specified (same as the gunzip command) when used with the gzip command.
-f --force	Compresses symbolic links when used with the gzip command. When used with the gunzip command, it overwrites any existing files without prompting the user.

Table 11-2: Common options used with the gzip utility

# The gzip Utility (continued)

Option	Description
<code>--fast</code>	Results in a lower compression ratio; same as the <code>-1</code> option.
<code>-h</code> <code>--help</code>	Displays the syntax and available options for the <code>gzip</code> and <code>gunzip</code> commands.
<code>-l</code> <code>--list</code>	Lists the compression ratio for files that have been compressed with <code>gzip</code> .
<code>-n</code> <code>--no-name</code>	Does not allow <code>gzip</code> and <code>gunzip</code> to preserve the original modification and access time for files.
<code>-q</code> <code>--quiet</code>	Suppresses all warning messages.

Table 11-2 (continued): Common options used with the gzip utility

# The gzip Utility (continued)

Option	Description
<code>-r</code> <code>--recursive</code>	Specifies to compress or decompress all files recursively within a specified directory.
<code>-S .suffix</code> <code>--suffix</code> <code>.suffix</code>	Specifies a file suffix other than <code>.gz</code> when compressing or decompressing files.
<code>-t</code> <code>--test</code>	Performs a test decompression such that a user can view any error messages before decompression, when used with the <code>gunzip</code> command; it does not decompress files.
<code>-v</code> <code>--verbose</code>	Displays verbose output (compression ratio and filenames) during compression and decompression.

Table 11-2 (continued): Common options used with the gzip utility

# The bzip2 Utility

- ◎ `bzip2` command: used to compress files using Burrows-Wheeler Block Sorting Huffman Coding compression algorithm
  - Cannot compress directory full of files
  - Cannot use `zcat` and `zmore` to view files
    - Must use `bzcat` command
  - Compression ratio is 50% to 75% on average
  - Uses `.bz2` filename extension by default
- ◎ `bunzip2` command: used to decompress files compressed via `bzip2`



# The bzip2 Utility (continued)

Option	Description
-#	Specifies the block size used during compression; -1 indicates a block size of 100K, whereas -9 indicates a block size of 900K.
-c --stdout	Displays the contents of the compressed file to SO when used with the bunzip2 command.
-d --decompress	Decompresses the files specified (same as the bunzip2 command) when used with the bzip2 command.
-f --force	Compresses symbolic links when used with the bzip2 command. When used with the bunzip2 command, it overwrites any existing files without prompting the user.
-k --keep	Keeps the original file during compression; a new file is created with the extension .bz2.

Table 11-3: Common options used with the bzip2 utility

# The bzip2 Utility (continued)

Option	Description
<code>-q</code> <code>--quiet</code>	Suppresses all warning messages.
<code>-s</code> <code>--small</code>	Minimizes memory usage during compression.
<code>-t</code> <code>--test</code>	Performs a test decompression such that a user can view any error messages before decompression, when used with the <code>bunzip2</code> command; it does not decompress files.
<code>-v</code> <code>--verbose</code>	Displays verbose output (compression ratio) during compression and decompression.

Table 11-3 (continued): Common options used with the bzip2 utility

# The tar Utility

- Tape archive (tar) utility: one of oldest and most common backup utilities
  - Can create archive in a file on a filesystem or directly on a device
- `tar` command: activates tar utility
  - Arguments list the files to place in the archive
  - Accepts options to determine location of archive and action to perform on archive

# The tar Utility (continued)

Option	Description
-A --catenate --concatenate	Appends whole archives to another archive
-c --create	Creates a new archive
--exclude <i>FILENAME</i>	Excludes <i>FILENAME</i> when creating an archive
-f <i>FILENAME</i> --file <i>FILENAME</i>	Specifies the location of the archive ( <i>FILENAME</i> ); it can be a file on a filesystem or a device file
-h --dereference	Prevents tar from backing up symbolic links; instead, <i>tar</i> backs up the target files of symbolic links
-j --bzip	Compresses/decompresses the archive using the bzip2 utility
-P --absolute-paths	Stores filenames in an archive using absolute pathnames
-r --append	Appends files to an existing archive
--remove-files	Removes files after adding them to an archive

Table 11-5: Common options used with the tar utility

# The tar Utility (continued)

<b>-x</b>	<b>Extract Archive</b>
<b>-c</b>	<b>Create New Archive</b>
<b>-r</b>	<b>Append to Archive</b>
<b>-v</b>	<b>verbose</b>
<b>-f</b>	<b>Write/Read File</b>
<b>-t</b>	<b>List Archive Contents</b>
<b>-z</b>	<b>Compress using gzip (c mode only)</b>

Table 11-5 (continued): Common options used with the tar utility

# The tar Utility (continued)

- ◎ tar utility does not compress files inside archive
  - Time needed to transfer archive across a network is high
  - Can compress archive
- ◎ Backing up files to compressed archive on a filesystem is useful when transferring data across a network
  - Use options of the `tar` command to compress an archive immediately after creation

# Summary

- Many compression utilities are available for Linux systems; each uses a different compression algorithm and produces a different compression ratio
- tar is the most common backup utility used today
  - Typically used to create compressed archives called tarballs

# Other useful commands





- id (print user identity)
- Whoami (print effective userid)
- hostname (displays the name of the system you are working on.)
- who (The who utility displays a list of users who are logged in on the system)
- echo (Copies anything you put on the command line after **echo to the** screen. Useful for learning about the shell and other Linux programs.)
- date (Displays the Time and Date)

# Redirecting Standard Input and Standard Output

# Standard Input and Standard Output

- Every program you run from the shell opens three files:
  - standard input ← 0
  - standard output ← 1
  - standard error ← 2
- The files provide the primary means of communications between the programs, and exist for as long as the process runs.
- **The standard input** file provides a way to send data to a process. As a default, the standard input is read from the *terminal keyboard*.
- **The standard output** provides a means for the program to output data. As a default, the standard output goes to the *terminal display screen*.
- **The standard error** is where the program reports any errors encountered during execution. By default, the standard error goes to the *terminal display*.

# Redirecting Input and Output

- It is possible to tell a program:
  - where to look for input
  - where to send output,using input/output redirection. UNIX uses the special characters **<** and **>** to signify input and output redirection, respectively.
- **Redirecting input:** Using **<** with a file name (i.e., **< file1**) in a shell command tells the shell to read input from a file called "file1" instead of from the keyboard.
  - ***command [arguments] < filename***
  - `$ more < /etc/passwd`
- **Redirecting output:** Using **>** with a file name (i.e., **> file 2**) causes the shell to place the output from the command in a file called "file2" instead of on the screen. If the file "file2" already exists, the old version will be overwritten.
  - ***command [arguments] > filename***
  - `$ sort pippo > pippo.ordinato`

# Redirecting Input and Output

- Use **>>** to append to an existing file (i.e., **>> file2**) causes the shell to append the output from a command to the end of a file called "file2".
  - If the file "file2" does not already exist, it will be created.

- **Example**

```
1. $ ls /bin > ~/bin; wc -l ~/bin  
   $ ls /usr/sbin > ~/bin ; wc -l ~/bin
```

```
2. $ ls /bin > ~/bin;  
   $ ls /usr/sbin >> ~/bin; wc -l ~/bin
```

# Redirecting Error

- Using **>&** with a file name (i.e., **>& file1**) causes the shell to place the **standard error** and the **standard output** from the command in a file called "file1".
  - If the file "file1" already exists, the old version will be overwritten.
- **Example**
  - `$ ls abcdef`
  - `$ ls abcdef >& lserror`
  - `cat lserror`
  
  - `$ abcdef >& command`
  - `cat command`
  
  - `$ mkdir /bin/miei >& ~/miei; cat ~/miei`
  - `$ rm /bin/perl >& ~/errperl; cat ~/errperl`

# Pipe & Filter

# Pipes

- UNIX allows you to connect processes, by letting the standard output of one process feed into the standard input of another process. That mechanism is called a **pipe (|)**.
- `$ command1 | command2` causes the standard output of `command1` to flow through to standard input of `command2`.
- A sequence of commands chained together in this way is called a **pipeline**. Connecting simple processes in a pipeline allows you to perform complex tasks without writing complex programs.
  - `$ cat /etc/passwd | sort > ~/pass_ord`
  - `$ sort < pippo | lpr`



# Examples

- Print the list of files

```
$ ls > temp
```

```
$ lpr temp
```

```
$ rm temp
```

*or*

```
$ ls | lpr
```

- Logins of 'osLab' account

```
$ who | grep 'root'
```

- Show the contents of a directory one page each time

```
$ ls | less
```

# Linux Editors

## vim



# What is vi ?

- The *visual editor* on the Unix.
- Before vi the primary editor used on Unix was the line editor
  - User was able to see/edit only one line of the text at a time
- The vi editor is not a text formatter (like MS Word, Word Perfect, etc.)
  - you cannot set margins
  - center headings
  - Etc...



# Editors

- emacs
  - Old and very user friendly
  - Menu based
- mcedit
  - A part of the midnight commander
  - Menu based, easy to use
- vi & vim (vi improved)
  - Difficult
  - Editor for programmers
  - Minimalist interface, Very little info displayed
  - Powerful shortcuts and commands



# 15 ubuntu text editors

- Gedit
- Cream
- Jedit
- Emacs
- Vim
- Nano
- sciTE
- Leafpad
- Geany
- Bluefish
- Xemacs
- Kwrite
- Scribes
- Lyx
- Pico

<http://www.addictivetips.com/ubuntu-linux-tips/15-ubuntu-text-editors-grab-your-favorite/>

sudo apt-get install



# Vim equals Vi

- The current iteration of **vi** for Linux is called **vim**
  - **Vi** **I**mproved
  - <http://www.vim.org>





# Starting vi

- Type **vi** <filename> at the shell prompt
- After pressing enter the command prompt disappears and you see tilde(~) characters on all the lines
- These tilde characters indicate that the line is blank



# Vi modes

- There are two modes in vi
  - Command mode
  - Input mode
- When you start vi by default it is in command mode
- You enter the input mode through various commands
- You exit the input mode by pressing the Esc key to get back to the command mode





# How to exit from vi

(comand mode)

- **:q** <enter> is to exit, if you have not made any changes to the file
- **:q!** <enter> is the forced quit, it will discard the changes and quit
- **:wq** <enter> is for save and Exit
- **:x** <enter> is same as above command
- **ZZ** is for save and Exit (Note this command is uppercase)
- The **!** Character forces over writes, etc. **:wq!**



# Moving Around

- ◆ You can move around only when you are in the command mode
- ◆ Arrow keys usually works (but may not)
- ◆ The standard keys for moving cursor are:
  - **h** - for left
  - **l** - for right
  - **j** - for down
  - **k** - for up



# Moving Around

- **w** - to move one word forward
- **b** - to move one word backward
- **\$** - takes you to the end of line
- **<enter>** takes the cursor to the beginning of next line



# Moving Around

- **f** - (find) is used to move cursor to a particular character on the current line
  - For example, **fa** moves the cursor from the current position to next occurrence of '**a**'
- **F** - finds in the reverse direction



# Moving Around

- `)` - moves cursor to the next sentence
- `}` - move the cursor to the beginning of next paragraph
- `(` - moves the cursor backward to the beginning of the current sentence
- `{` - moves the cursor backward to the beginning of the current paragraph



# Moving Around

- **Control-d** scrolls the screen down (half screen)
- **Control-u** scrolls the screen up (half screen)
- **Control-f** scrolls the screen forward (full screen)
- **Control-b** scrolls the screen backward (full screen).



# Entering text

- To enter the text in vi you should first switch to **input mode**
  - To switch to input mode there are several different commands
  - **a** - Append mode places the insertion point after the current character
  - **i** - Insert mode places the insertion point before the current character



# Entering text

- **I** - places the insertion point at the beginning of current line
- **o** - is for open mode and places the insertion point after the current line
- **O** - places the insertion point before the current line
- **R** - starts the replace(overwrite) mode





# Editing text

- **x** - deletes the current character
- **d** - is the delete command but pressing only d will not delete anything you need to press a second key
  - **dw** - deletes to end of word
  - **dd** - deletes the current line
  - **do** - deletes to beginning of line
- There are many more keys to be used with delete command



# Structure of vi command

- The vi commands can be used followed by a number such as  
**n<command key(s)>**
  - For example **dd** deletes a line **5dd** will delete five lines.
- This applies to almost all vi commands



# Undo and repeat command

- **u** - undo the changes made by editing commands
- **.** (dot or period) repeats the last edit command
- **^R**- Redo



# Copy, cut and paste in vi

- **yy** - (yank) copy current line to buffer
  - **nyy** - Where **n** is number of lines
  - **p** - Paste the yanked lines from buffer to the line below
  - **P** - Paste the yanked lines from buffer to the line above
- (the paste commands will also work after the **dd** or **ndd** command)



# User Management



# Managing Users

- Each system has two kinds of users:
  - Superuser (root)
  - Regular user
- Each user has his own username, password, and permissions that can only be assigned by the user.
- All users have a user ID (**UID**) and a group ID (**GID**).

# The Password File

- `/etc/passwd`
- It is the database file for all users on the system.
- Username:password:uid:gid:comment:homedir:shell
- \* in password means disable.

# Shadow Passwords

- `/etc/shadow`
- It is considered to use the encrypted passwords found in `/etc/passwd`.
- Only x or \* appears in the passwd field of `/etc/passwd`.



# Shadow Password Fields

- The user's login name
- The encrypted password
- The number of days since jan 1970
- The number of days before the the password can be changed
- The number of days before the password is to expire that the user is warned it will expire.
- The number of days after the password expires the account is disabled.
- The number of days since jan 1 1970 that account has been disabled.

# Groups

- User groups are a convenient way to **logically organize** sets of user accounts and allow users to **share files** within their group or groups.
- Each file on the system has both a user and a group **owner** associated with it.
  - *chmod g+...*
- Every user is assigned to at least one group
- User has "group access" to any files on the system with a group ID contained in his list of groups (default and additional groups)

# Groups

- [/etc/group](#)
- The custom is to use GIDs of 500 or more for regular users and less for administrations or special program.
- Groupname:password:gid:users

root:x:0:root

bin:x:1:root,bin,daemon

test:x:500:

- Special group
  - Allow these services to manage their own files with permissions that restrict other users from them.

# User Management Commands

- `useradd`
  - Create a new user
- `userdel`
  - Delete a user
- `usermod`
  - Modify a user account
- `passwd`
- `groupadd`
  - Create a new group
- `groupdel`
  - Delete a group
- `groupmod`
  - Modify a group
- `grpck`
  - Verify the integrity of the system authentication information.

# User's Home Directory

- When each user is created, a home directory is created for him (/home/<username>).
- The set of files that initially are used to populate this home directory are kept in [/etc/skel](#).

# Some Useful Command

- `chmod`
  - Change file access permission
- `chown`
  - Change file owner or group
- `chroot`
  - Run command with special root directory

# Some Useful Command

- setuid and setgid bit
  - **setuid** and **setgid** (short for **set user ID** upon execution and **set group ID** upon execution)
  - For executables
    - (note: they will have different meaning on directories)
  - chmod [ugo][+-]s
- Manage delete access
  - For directories
  - sticky bit
    - Chmod 1745
  - Example
    - Each user have write access to /usr/share
    - Each user could not delete others files

# GUI Administration Tool

- Creating users
  - KDE: K, System, Kuser
  - GNOME: Main Menu, KDE menus, System, Kuser
- Change Password
  - KDE: K, System, Change Password
  - GNOME: Main Menu, Programs, System, Change Password



# The su Command

- It is necessary for regular users to run a command as if they were root.
- The su means **substitute user**.
- This command changes the UID and GID of the existing user.
- The syntax for the su command is this:  
**su option username arguments**
- To return to the regular users' identity
  - exit

# The sudo Command

- It gives to the certain users only a few superuser permissions.
- The list of authorized users is kept in [/etc/sudoers](#)
- Sudo will prompt for a password and then check the /etc/sudoers.
- Sample:
  - Sudo fdisk /dev/hda1

# /etc/sudoers Format

- User\_Alias GURU=test  
Cmd\_Alias FILE\_TOOL=/bin/ls, /bin/cat

Syntax:

User\_Alias MACHINE=Cmd\_Alias

Example:

GURU ALL=FILE\_TOOLS

# Users and Groups Permissions

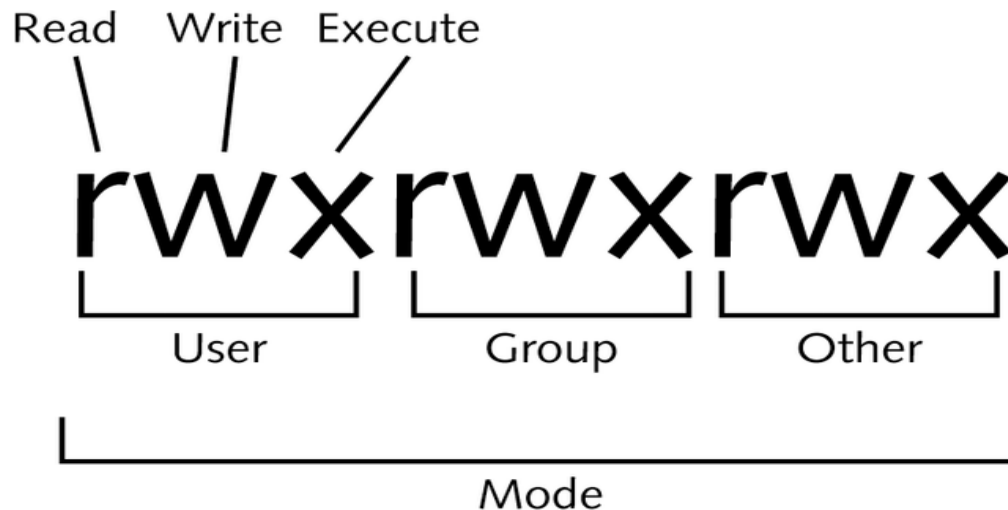


# Objectives

- Understand file permissions
- Understand directory permissions
- Understand groups
  - private
  - secondary
- Commands to change permissions
  - chmod, chgrp, chown
- Understand umask
  - Setting the default permissions with umask

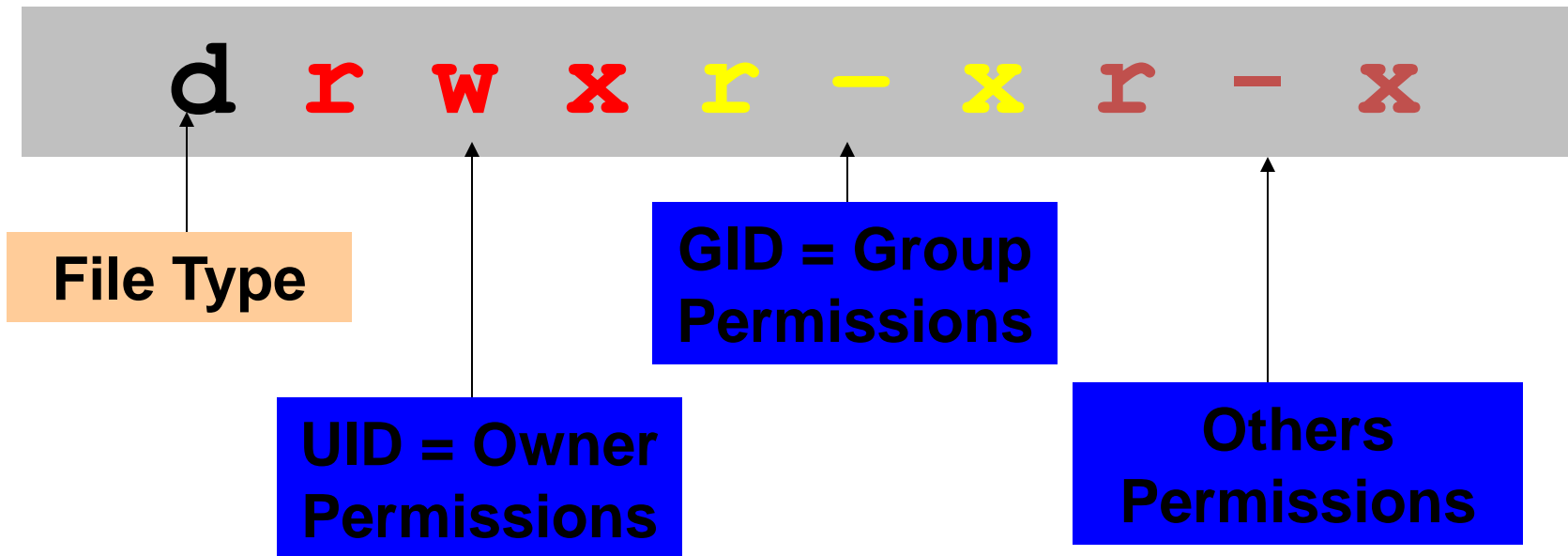
# Understanding How Permissions Work

- Each item has 3 sections of permissions.
  - User permissions: Owner
  - Group permissions: Associated Group
  - Other permissions: Everyone else on the system
- Three regular permissions may be assigned to each section:
- Each of the different sections can have different permissions.



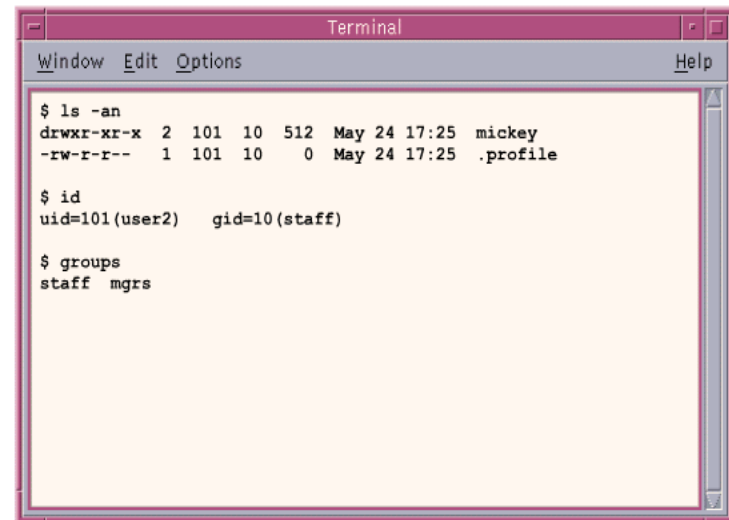
# All items have an owner /group.

- `ls -l` displays the items permissions.
- Each file / directory has an owner & a group linked to it.
  - These are determined using the uid & gid associated with it.
  - These are used to determine your access to all resources.
  - To access resources you must have the required permissions



# id command

- The user and group names displayed in a long listing actually correspond to User Identifier (**UID**) and Group Identifier (**GID**) numbers.
  - **ls -n** displays UID and GID instead of their text counterparts.
  - You can use the **id** command to displays the numeric UIDs and GIDs for any user.
  - **id userid**
  - The **groups** command displays all the groups the user is a member of.
  - **groups userid**



```
Terminal
Window Edit Options Help
$ ls -an
drwxr-xr-x  2 101 10 512 May 24 17:25 mickey
-rw-r-r--  1 101 10   0 May 24 17:25 .profile

$ id
uid=101(user2)  gid=10(staff)

$ groups
staff mgrs
```



# Files & Directories use permissions differently.

- Default file = `- r w - r w - r - -`
- Default directory = `d r w x r w x r - x`

Permission	Symbol	Plain File	Directory
Read	r	File can be displayed or copied.	Contents can be listed with the ls command. (To Display a long listing (ls -l) you must also have Execute permission)
Write	w	File contents can be modified.	Files can be added or deleted. (To add or delete files you must also have execute permission)
Execute	x	File can be executed (shell scripts or executables only).	Allows the find command to search thru directory.
No Permission	-	A dash (-) indicates permission is denied	A dash (-) indicates permission is denied

For Directories

r = listable

x = searchable

# Commands used to change permissions

- **chmod** (change mode) command:
  - Change mode (permissions) of files or directories
  - You must be able to use this.
- **chgrp** (change group) command:
  - Change group owner of a file or directory
  - Only superuser can use this on Linux.
  - **chgrp new-owner itemname**
  - **chgrp -R new-owner itemname**
- **chown** (change owner) command:
  - Change ownership of a file or directory
  - Only superuser can use this on Linux.
  - **chown new-owner itemname**
  - **chown -R new-owner itemname**
  - **chown new-owner.newgroup itemname**

# Changing Permissions

**\$ chmod mode filename**

- Permissions can be referred to either using Octal or Symbolic mode.
- Both use exactly the same permissions just differently.
- Both are used with the chmod command too change permissions.

- **Octal verse Symbolic Permissions**

- **Octal** grants the 3 characters within each of the 3 sections a value
- You must always refer to all 3 sections

**chmod 754 perm**

placement	=	1 2 3	1 2 3	1 2 3
permissions	=	r w x	r - x	r - -
value	=	4 2 1	4 2 1	4 2 1

- **Symbolic** uses a specified 'id' letter which refers to a specific mode section:

**chmod g-x perm**

# Values of Symbolic Permissions

**\$ chmod symbolic-mode filename**

In symbolic the mode portion is made up of three parts:

Who - Category you are working with

u = user

g = group

o = others

a = all

Op - Operator

set (=)

remove (-)

give (+)

Permission(s) to be assigned – Read (r), Write (w) or Execute (x)

Category	Operation	Permission
u (user)	+ (adds a permission)	r (read)
g (group)	- (removes a permission)	w (write)
o (other)	= (makes a permission equal to)	x (execute)
a (all categories)		

# Examples of Symbolic mode

- You can refer to a single or multiple sections, as long as they need the same action

```
chmod g-x perm
```

```
chmod u+r perm
```

```
chmod ug+x perm
```

```
chmod ugo-w perm
```

- You can not add and remove permissions in the same action
- They need to be separated by a comma.

```
chmod g-x,o+x perm    chmod u+rx,g-w perm
```

- **chmod a+rwx <filename>**  
would give read, write and executable permission to the filename specified to ALL users
- **chmod go+r, go-wx <filename>**  
would give the group and world read permission but remove write and execute permissions to the file

# Octal mode - Permissions

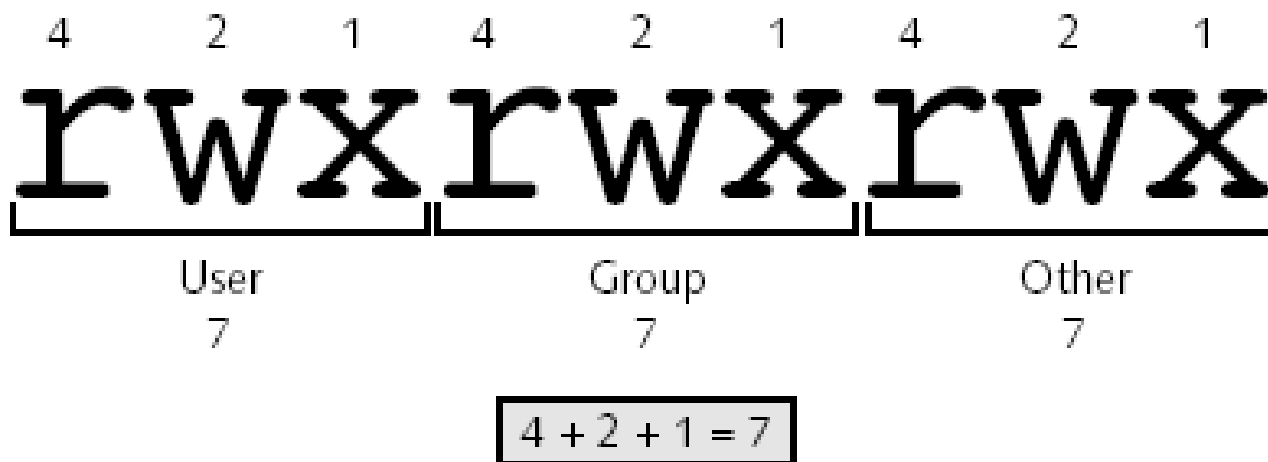


Figure 5-4: Numeric representation of the mode

Mode (one section only)	Corresponding Number
<b>rwx</b>	$4 + 2 + 1 = 7$
<b>rw-</b>	$4 + 2 = 6$
<b>r-x</b>	$4 + 1 = 5$
<b>r--</b>	4
<b>-wx</b>	$2 + 1 = 3$
<b>-w-</b>	2
<b>--x</b>	1
<b>---</b>	0

# Octal mode - examples

- You must always refer to all 3 sections

	Octal	Permission	Notes
<b>chmod 644 perm</b>	644	rw-r--r--	Default for files
<b>chmod 755 perm</b>	755	rw-r-xr-x	Default for Directories
<b>chmod 555 perm</b>	555	r-xr-xr-x	
<b>chmod 744 perm</b>	744	rw-r--r--	Typical for user scripts
<b>chmod 777 perm</b>	777	rw-rw-rw-x	All permissions to all

# ??? Quick Question ???

What is the octal value of the following symbolic permissions?

`r-xr-xr--`



# ??? Another Quick Question ???

What is the symbolic value of the following octal permissions?

536

# Umask is used to set the default permission.

- New files are given rw-rw-rw- permissions by default
- Used on it's own it will display how the the umask is currently set.
- Is used to reset the default permissions the system puts on new files & directories
- Changing the umask
- **umask 022**
  - Here numbers are used in reverse ie: 0 = all default permissions

	New Files	New Directories
Permissions assigned by system	rw-rw-rw-	rw-rw-rw-
- umask	0 2 2	0 2 2
= resulting permissions	rw-r--r--	rw-r--r--

Figure 5-5: Performing a umask 022 calculation

# Mount the file system



# Mounting

- Mount
  - To add a filesystem to other filesystem
    - Add you cool-disk FS to you laptop FS
- How?
  - `mount <options> <device> <mount point>`
  - `mount -t vfat /dev/sdb1 /mnt/flash`
- Don't forget the umount
  - `umount <mount point>`
  - `umount /mnt/flash`

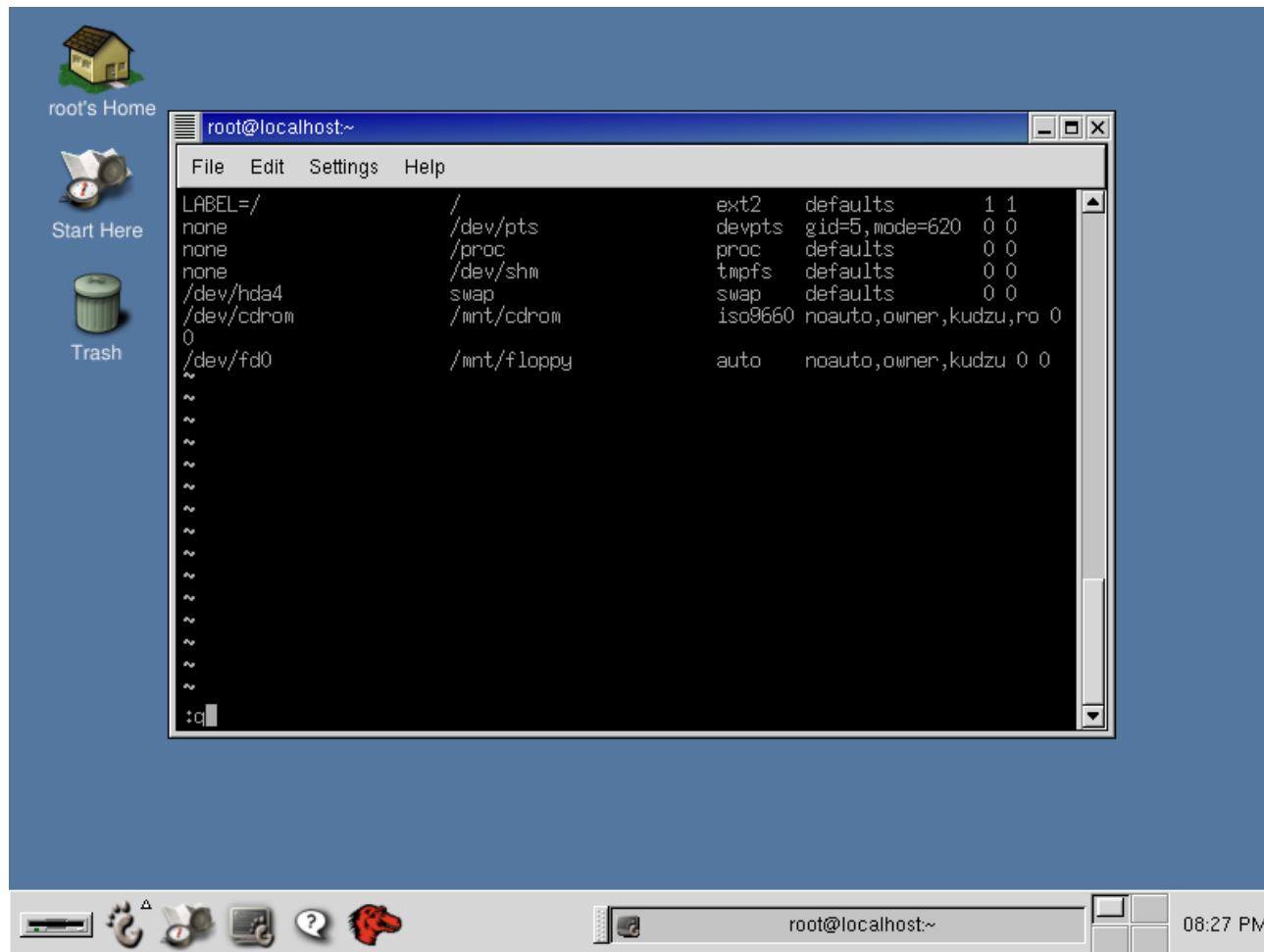
# Mounting File System

- The Linux File System makes it appear as if all the File System are local and mounted somewhere on the root File System.
- File System are mounted with the `mount` command.
  - `mount -t type source mount_point`
- To unmount a File System, the `umount` command is used.
  - `umount /dev/<device name> or mount_point`

# Mounting Automatically with fstab

- This file lists all the partitions that need to be mounted at boot time and the directory where they need to be mounted.
- Along with that information, you can pass parameters to the mount command.
- /etc/fstab
  - Which devices to be mounted
  - What kinds of File Systems they contain
  - At what point in the File System the mount takes place
  - ...

# fstab Example



# fstab Structure

- Each line has **six** fields:
  - **1'st field**: indicates the **block device or remote File System** that will be mounted.
  - **2'nd field**: identifies the **mount point** the local system where the File System will be mounted.
  - **3'rd field**: File System **type**
  - **4'th field**: list of mount options
  - **5'th field**: it is used by **dump** (a backup program) to determine whether the File System should be dumped (1:yes, 0:no).
  - **6'th field**: it is used by **fsck** (0:never run, 1:run on the drive at predetermined, 2:it is recommended for non root File System so that fsck isn't run on them as frequently).

```
LABEL=/          /          ext2      defaults      1 1
none             /dev/pts    devpts    gid=5,mode=620 0 0
none             /proc       proc      defaults      0 0
none             /dev/shm    tmpfs     defaults      0 0
/dev/hda4         swap        swap      defaults      0 0
/dev/cdrom        /mnt/cdrom  iso9660   noauto,owner,kudzu,no 0
0
```



# Linux FS vs. Windows FS

- There is not drive C:, D:
- Top hierarchy is /
- Path separator is / not \
- File extensions have NOT any meaning
- There is not hidden attribute, hidden files are started by .

# Shell help commands



# Helps

- Some documents are in /usr/share/doc
- Info pages are not complete help
  - **info** <command name>
- Man pages
  - /usr/share/man
  - man1: user commands, man8: System administration
  - **man** <command name>

# Information about commands

- man (displays the on-line manual pages)
  - man command
- info(read documentation in info format)
  - info command
- help
  - command --help

# man command

- `$ man -k keyword` displays a list of commands in which description there is the word “keyword”
  - `man -k cat`
  - `man -k manual`

# References

