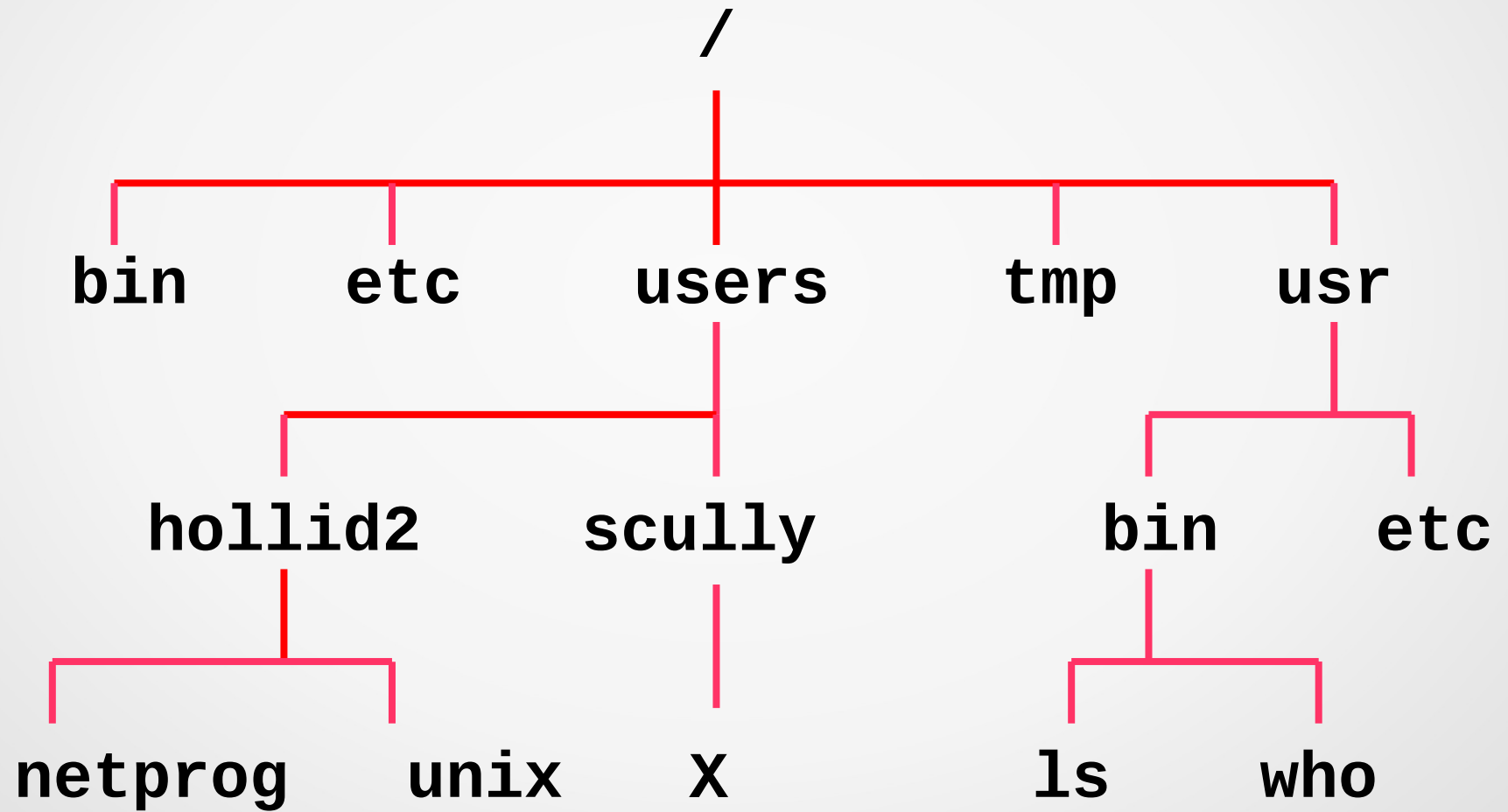


Unix File system

- The file system is a **hierarchical system of organizing files and directories.**
- The top level in the hierarchy is called the "root" and *holds* all files and directories.
- The name of the root directory is /

The File System



Directories

- A directory is a special kind of file -
- Unix uses a directory to hold information about other files.
- Directory acts as a container that holds other files (or directories).

File System

- A file system is consists of a sequence of logical blocks (512/1024 byte etc.)
- A file system has the following structure:

Boot Block	Super Block	Inode List	Data Blocks
------------	-------------	------------	-------------

File System: Inode List

- Inodes are used to access disk files.
- Inodes maps the disk files
- For each file there is an inode entry in the inode list block
- Inode list also keeps track of directory structure
- consists of
 - file owner identifier
 - file type
 - file access permissions
 - file access times
 - number of links to the file
 - table of contents for the disk address of data in a file
 - file size

UNIX File Types

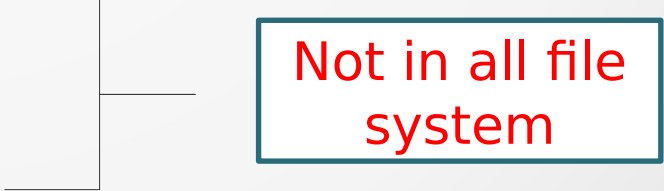
- **Regular**
 - Contains arbitrary data stored in zero or more data blocks
 - Treated as stream of bytes by the system
- **Directory**
 - Contains list of file names along with pointers to associated nodes (index nodes, or *inodes*)
 - Organized in hierarchies
- **Special**
 - Contains no data, but serves as a mapping to physical devices
 - Each I/O device is associated with a special file
- **Named pipe**
 - Implement inter-process communication facility in file system name space
- **Link**
 - Provides name aliasing mechanism for files
- **Symbolic link**
 - Data file containing the name of file it is linked to

File Types

- 1.Regular files
- 2.Directories
- 3.Character device files
- 4.Block device files
- 5.UNIX domain sockets
- 6.Named pipes (FIFOs)
- 7.Symbolic links

****ls -ld command gives type of file****

Q/p: drwx-----. 46 admin admin 4096 Jul 21 11:03 .



Not in all file system

File Types

File Type	Symbol used
Regular file	-
Directory	d
Character device file	c
Block device file	b
Local domain socket	s
Named pipe	p
Symbolic link	l

1. Regular Files

- A regular file is a big bag of bytes (so called bag o' bytes)
- Unix imposes no structure on its contents.
- Ex: Text files , data files, executable programs and shared libraries.
 - binary
 - GIF, JPEG, Executable etc.
 - text
 - scripts, program source code, documentation
- Supports sequential and random access

2. Directory

- Directory contains **named references of other files**.
Commands: **mkdir** and **rmdir**
- Can contain **ANY kind of files**.
 - what is **“.”** and **“..”**??
(**.** means directory itself and **..** means its parent directory)
 - **File's name** is actually stored in its **parent directory** not with the file itself.
 - **References to files** are called **Links**.
 - File system allow **more than one directory entry point to a particular file**.
 - **Attributes of files** like **ownership and permissions are shared among all links**.
 - **What is hard links and soft links?**

3. Device File

- Allows programs to communicate with hardware.
- The module for a particular device called device drivers to manage the device.
- Device drivers are like regular files.
- Kernel modules handles device management.
- Character Device
 - Accepts a stream of characters, without regard to any block structure.
 - It is not addressable, therefore no seek operation

4. Block Device

- Information stored in fixed-sized block
- It is addressable, therefore seek operation is possible.
- Device files are characterized by two numbers major and minor device numbers.
- Major device no. tells the kernel which driver the file refers to.
- Minor device no. tells the driver which physical unit to address.

Types of Files (cont.)

5. UNIX Domain Sockets (BSD)

- Sockets are connections between processes that allow them to **communicate**.
- Sockets that are local to a particular host and are referenced through a file system object rather than a network port.
- Used in X windows, syslog etc.

6. Named Pipe /FIFO files

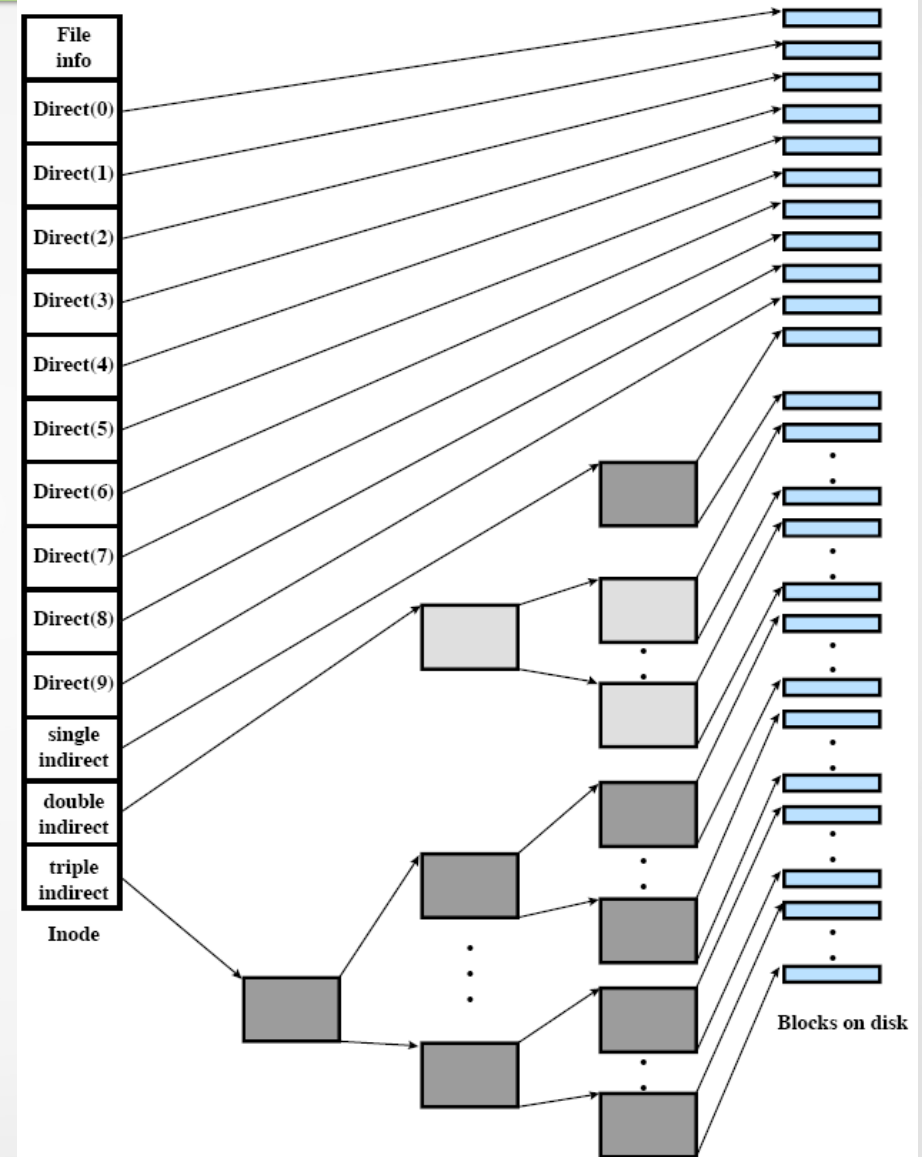
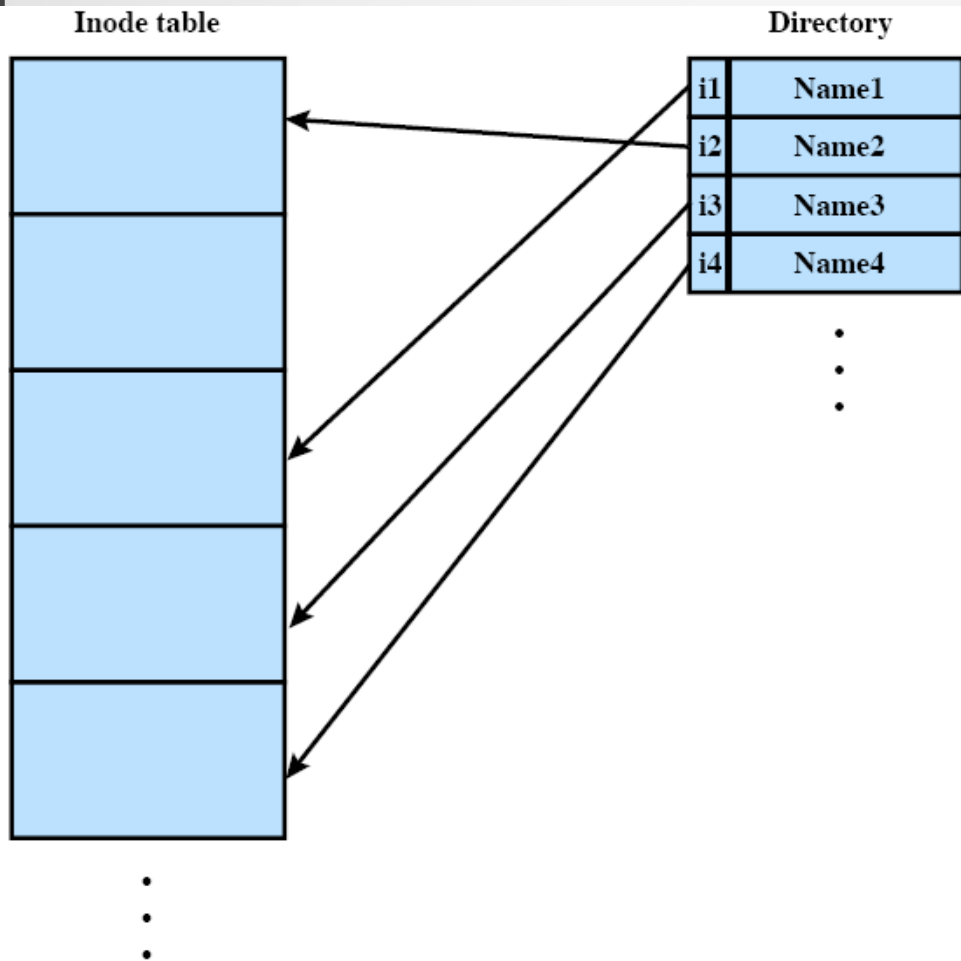
- Like sockets it **allow communication between two unrelated processes running on the same host.**

Types of Files (cont.)

7. Symbolic links

- symbolic links points to a file by its name (using its pathname).
- **Hard links**
 - Linking files by reference
 - System maintains a count of the number of links
 - Does not work across file systems.
- **Soft links**
 - Linking files by name
 - No counter is maintained
 - Work across file system

Directory Structure and File Layout



Organization of File Tree

- The UNIX file system has never been very well organized.”
 - incompatible naming convention
 - Different types of files are scattered in namespace.
e.g. long file naming
 - Due to different conventions, it is difficult to upgrade the Operating system.

Organizing of The File System (cont.)

/	The root directory
/bin or /sbin	Commands for basic system operation
/dev	Device entries
/etc	Critical startup and configuration files.
/lib	Library for the C compiler
/tmp	Temporary files
/var/adm or /var/log	Accounting file, log files
/proc	Information of all running process

Some more files ...

- /sys : kernel building work area, configuration files.
- /proc : images of all running processes
- /usr/bin : executable files
- /usr/man : online manual pages
- /usr/include: header files for C programs
- /usr/share/man : online manual pages
- /var/tmp: more temporary spaces

** /usr and /var directories are very important **

File Organization

- In `/usr`, most of the standard programs are kept along with many online `manual pages` and most of the libraries.
- While `/var`, maintains directories, log files, accounting information and various other entities that are needed on each host.
- Eg: `/usr/bin`, `/usr/tmp`, `/usr/lib`, `/usr/share` etc..
- Eg: `/var/tmp`, `/var/log`, `/var/adm` etc..

/usr	Contains most system commands and utilities—contains the following directories: /usr/bin—User binary commands /usr/games—Educational programs and games /usr/include—C program header files /usr/lib—Libraries /usr/local—Local programs /usr/sbin—System binary commands /usr/share—Files that are architecture independent /usr/src—Source code /usr/X11R6—The X Window system
------	---

/usr/local	Is the location for most additional programs
------------	--

/var	Contains log files and spools
------	-------------------------------

File Attributes

- Every file has a set of 9 permission bits that control who can **read, write and execute** the contents of the file.
- All users must login with a **username and password**
- Users identified by **username and group memberships**
 - **Access to resources** depends on username and group membership
 - To access resources it must have permissions

File Permissions

- Files and directories have
 - ✓ Owner
 - ✓ Group
- Linux determines who can access file or directory based on:
 - ✓ Who is owner
 - ✓ Which group is assigned to object/file
- File permissions define access granted to file or directory

File Permissions (continued)

- Access mode/access permissions/access control
- Permissions
 - ✓ Read permission (r) (4)
 - ✓ Write permission (w) (2)
 - ✓ Execute permission (x) (1)
- Permissions can be assigned by:
 - ✓ User permissions
 - ✓ Group permissions
 - ✓ Other permissions

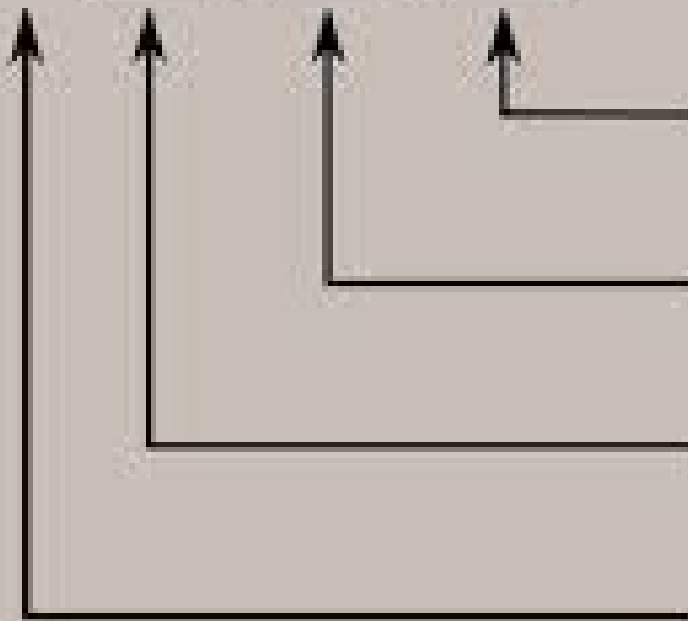
** use ls -l command to see long list of the file.**

Output is like :

```
-rw-rw-r-- 1 admin admin 1545 Jun 26 10:25 ass1.sh
-rw-rw-r-- 1 admin admin 1542 Jun 26 10:19 ass1.sh~
-rw-rw-r-- 1 admin admin 628 Oct 17 2013 b06.sh
-rw-rw-r-- 1 admin admin 731 Oct 17 2013 B06.sh
drwxrwxr-x 4 admin admin 4096 Mar 7 15:10 color-animation
```

File Permissions (continued)

- rwx rwx rwx



Read, write, and execute permissions for all other users.

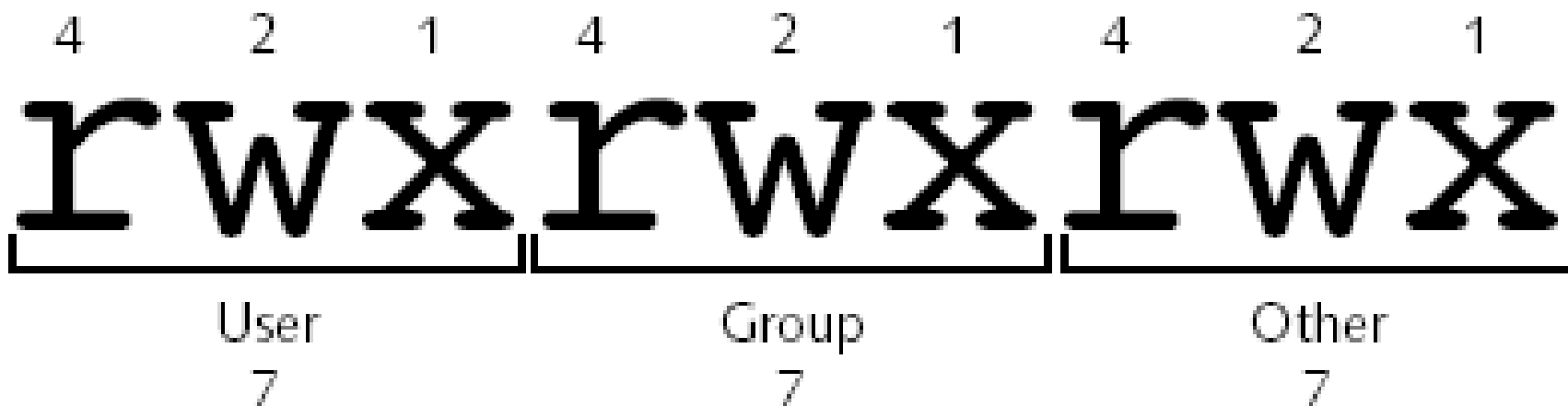
Read, write, and execute permissions for the group owner of the file.

Read, write, and execute permissions for the file owner.

File type:
- indicates regular file
d indicates directory

File Permissions (continued)

- On a regular file, the “r” bit allows the file to be opened and read.
- “w” bit allows the file contents to be modified or truncated.
- Delete or rename is now allowed directly as the file is controlled by the permissions on its parent directory.
- “x” bit allows the file to be executed.
- Executable files are of 2 types :
 1. Binary file: which the CPU runs directly.
 2. Script file: which must be interpreted by shell or some other program like compilers.



$$4 + 2 + 1 = 7$$

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

Permissions

Permission	Definition for Files	Definition for Directories
Read	Allows a user to open and read the contents of a file	Allows a user to list the contents of the directory (if she has also been given execute permission)
Write	Allows a user to open, read, and edit the contents of a file	Allows a user to add or remove files to and from the directory (if she has also been given execute permission)
Execute	Allows a user to execute the file in memory (if it is a program file or script)	Allows a user to enter the directory and work with directory contents

Check for the permissions

777	000	111	121	242
222	444	412	421	124
254	564	251	256	177



Changing Permissions

Changing Ownership

- **chown command**
 - Change user and group assigned to file or directory
 - Can only use when logged in as root
 - Example: chown jtaylorManagers report.doc**
- To change a file's group, you must either be the owner of the file and belong to the group you're changing to or be the superuser.
- **chgrp command**
 - Change group assigned to file or directory
 - **Example: chgrp managers report.doc**

Changing Ownership

- **chown (change owner) command**: change ownership of a file or directory
 - Two arguments:
 1. New owner
 2. File to change
 - Can use **-R** option for contents of directory
- **chgrp (change group) command**: change group owner of a file or directory
 - **Same arguments and options as for chown command**
 - Two arguments:
 1. New owner
 2. File to change

Changing Ownership

chown can change both the owner and group of a file at once with the syntax

chown user:group file ...

For example,

```
# chown -R matt : staff ~matt/restore
```


Changing Ownership

- **Primary group:** user's default group
- During file creation, file's owner and group owner set to **user's username** and **primary group** same for directory creation.
- **whoami** command: view current user name
- **groups** command: view group memberships and primary group
- **touch** command: create an empty file

Changing File Permissions

- **chmod (change mode) command**
 - Change file permissions
 - Regular users can alter permissions assigned to any file or directory that you own.
 - Only the **owner of the file and the superuser can change its permissions.**
 - The first argument to **chmod** is a specification of the permissions to be assigned, and the second and subsequent arguments are names of files on which permissions should be changed.
 - Examples:
 1. `chmod o+w report.doc`
 2. `chmod 711 myprog`
 - User (**u**), group (**g**) and others (**o**)

Changing File Permissions

- For the Alphabetical or mnemonic syntax, you combine a set of targets (u, g, or o for user, group, other) with an operator (+, -, = to add, remove, or set) and a set of permissions.
- Examples: u+w
O-x
- With the -R option, chmod recursively updates the file permissions within a directory.
- Example :\$ chmod -R g+w mydir

Default File Permissions

- `rw-rw-rw-`
- **umask command**
 - Defines mask to stop certain permissions from being granted by default when file is created
 - Executed automatically when you log in to Linux
 - Uses same three-digit permission codes as **chmod command**

umask: assign default permissions

- Every process has its own umask attribute.
- The umask is specified as a **three-digit octal value** that represents the permissions.
- When a file is created, its permissions are set to whatever the creating program requests **minus whatever the umask forbids.**

For example, umask 027

It allows all permissions for the owner but forbids write permission to the group and allows no permissions for anyone else.

Setting Special Permissions

- Special permissions require execute access.
- Mask the execute permission when displayed by the **ls -l command**
- May be set even if file or directory does not have execute permission
 - Indicating letter in the mode will be capitalized.
- Add special permissions via **chmod command**
 - Add an **extra digit at front of permissions argument**

Mounting and Un-mounting of file system

- Unix has a **tree file system**.
- The file system once created is **logically a separate entity and has a separate tree structure and root directory**.
- At the **time of booting**, these file system unites to become a single file system.
- The **root file system is the main file system and root directory is the directory for it**.

Mounting and Un-mounting of file system

- With mounting, the user sees a single file system and the file that seems moved from one directory to other.
- To **mount** means to attach a file systems to the root file system.
- **First an empty directory is created** in the main file system.
- Unix has **mount and umount** commands which are **used for device opening and closing for block devices**.
Eg: **mount** /dev/usb /home/ccoeu/pendrive1

Mounting

- **Mounting:** making a device accessible to users via the logical directory tree.
- **Mount point:** directory to which a device is attached
 - The mounted device temporarily covers up the contents of the mount point
 - Any existing directory can be a mount point
- In order to prevent making files inaccessible, create empty directories used specifically for mounting devices

Mounting (continued)

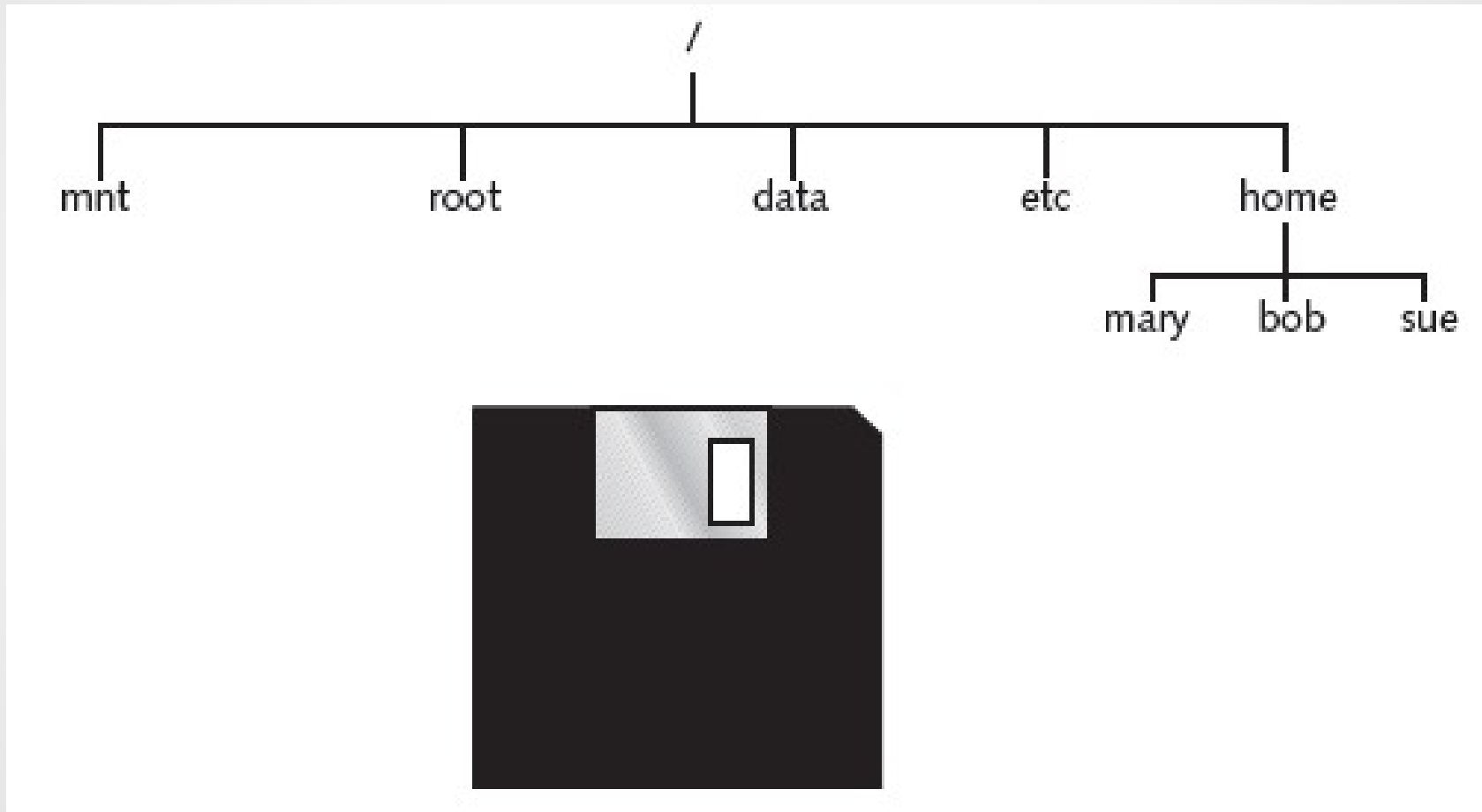
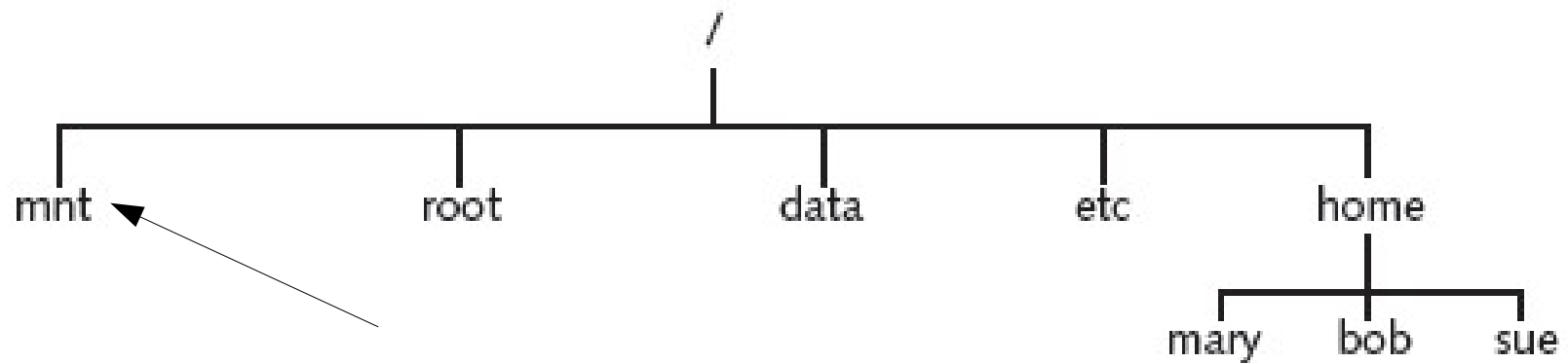


Figure 1: The directory structure prior to mounting

Mounting (continued)



Figure

Mounting (continued)

- **Root filesystem:** when Linux filesystem is first turned on, a filesystem on the hard drive is mounted to the "/" directory
 - Contains most OS files
- **mount command:** used to mount devices to mount point directories
 - When used with no options or arguments, lists currently mounted filesystems
- **umount command:** used to unmount devices from mount point directories

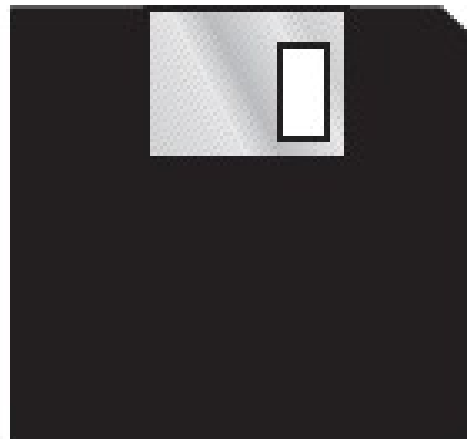
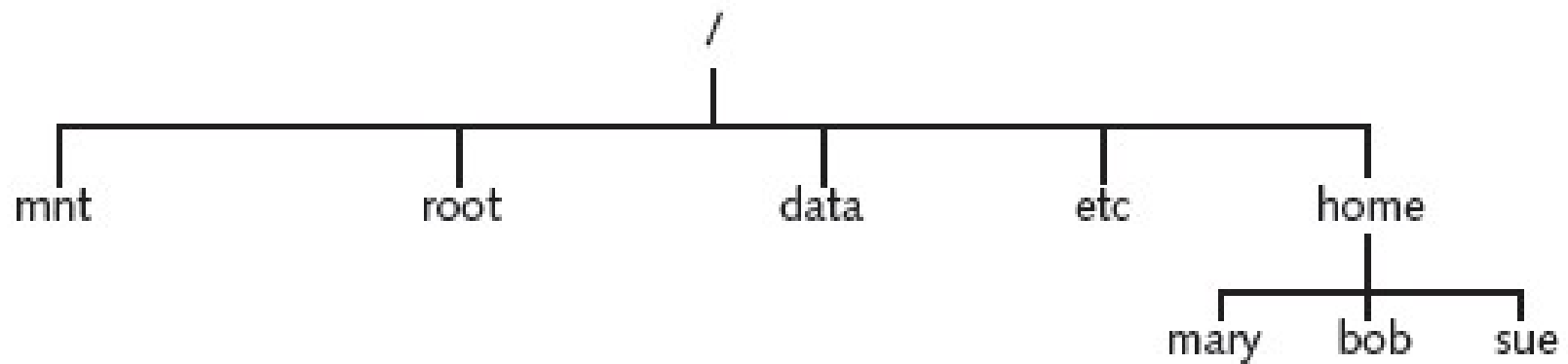
umount command

- It is **used to remove file systems.**

Eg: `umount /dev/usb/lp0`

- Unmounting of the file system is **not possible** if a **file is opened or not placed above it.**
- **Basic file systems are automatically mounted during start-up or booting** and **unmounted when the system is shut down.**

Unmounting (continued)



Figure

10