



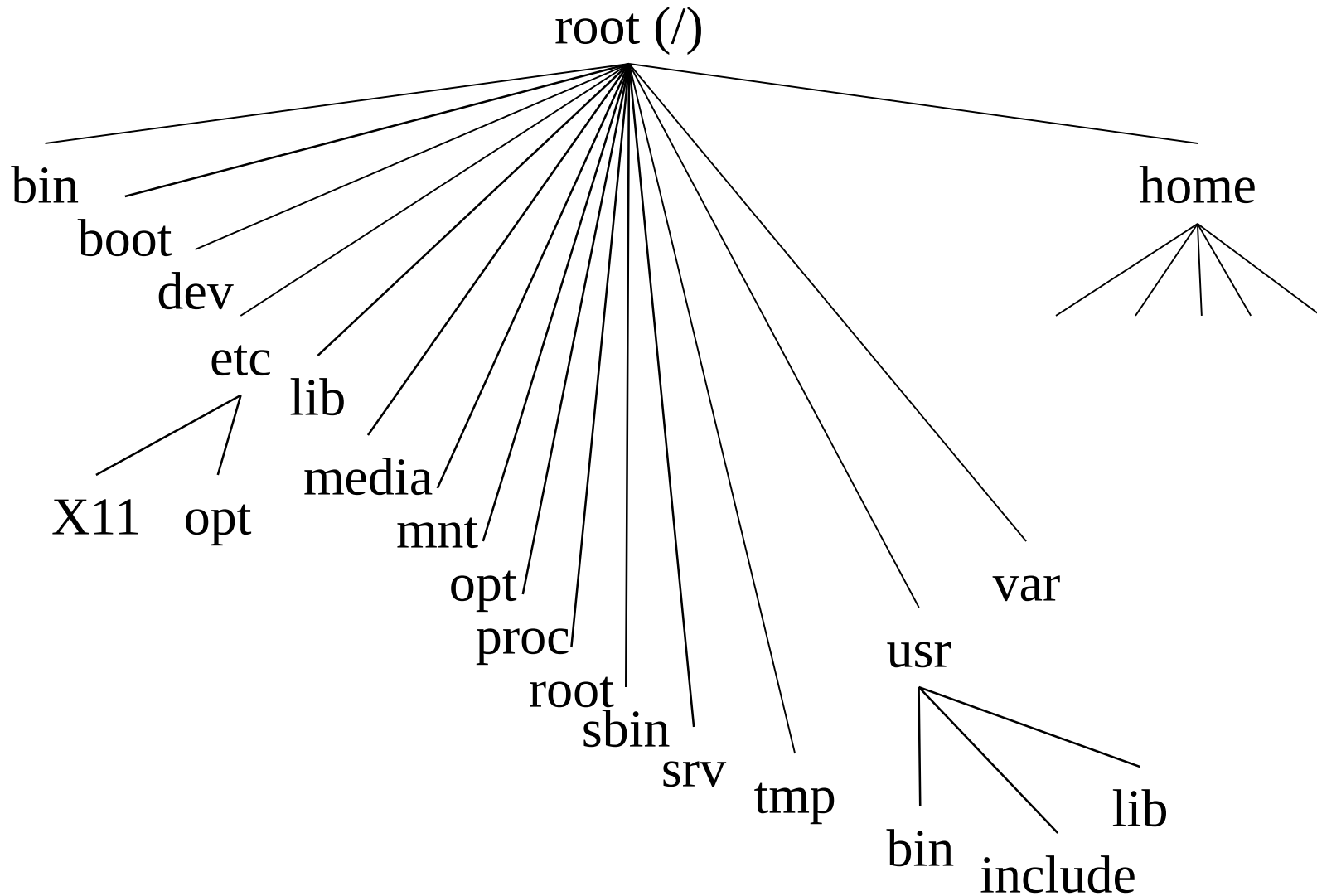
CHAPTER 3:

The file system

THE UNIX FILE SYSTEM

- hierarchical organization of files
 - contains directories and files
- basic commands to list and manipulate files
 - independent of physical file system organization
 - always single tree
- typical Unix file system types
 - ext3 (formerly ext2)
 - reiserfs
 - vfat
 - ntfs (now read & write)

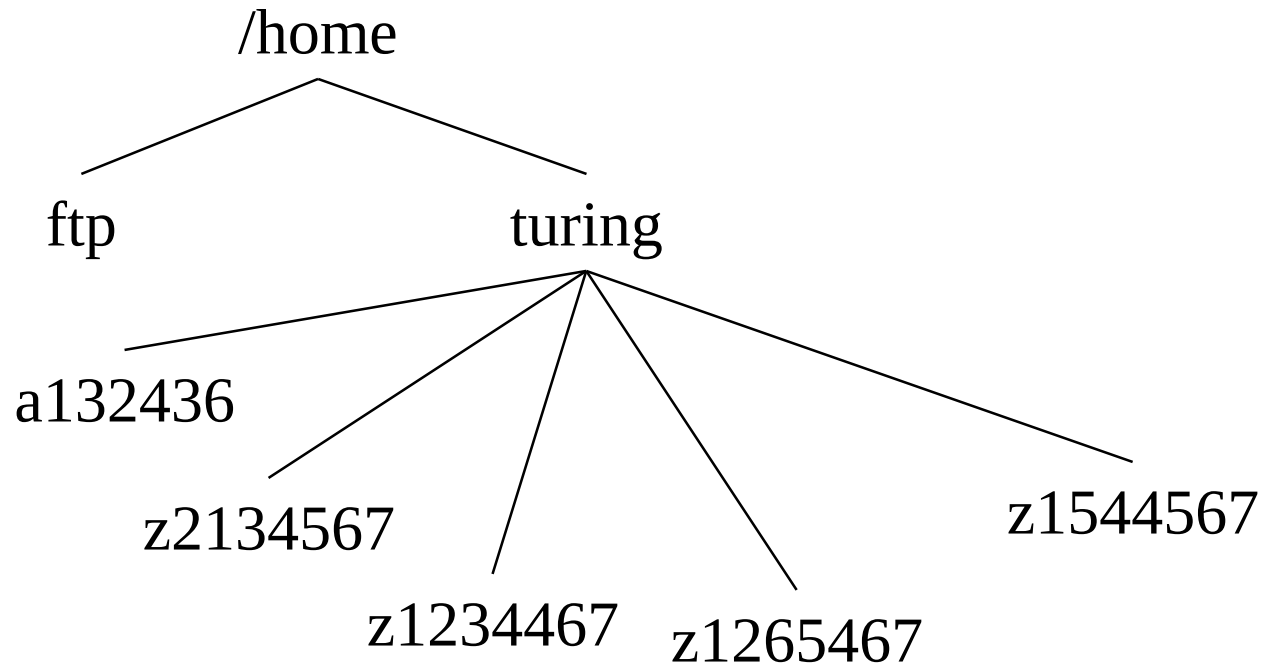
UNIX FILE SYSTEM LAYOUT



COMMON UNIX DIRECTORIES

bin	Essential command binaries
boot	Static files of the boot loader
dev	Device files
etc	Host-specific system configuration
lib	Essential shared libraries and kernel modules
media	Mount point for removable media
mnt	Mount point for temporary file systems
opt	Add-on application software packages
proc	data on running system
root	home directory for system administrator
sbin	Essential system binaries
srv	Data for services provided by this system
tmp	Temporary files
usr	Secondary hierarchy
var	Variable data

HOME DIRECTORIES ON TURING



DIRECTORY TERMINOLOGY

- Root Directory: `/`
 - top-most directory in any UNIX file structure
- Home Directory: `~`
 - directory owned by a user
 - default location when user logs in
- Current Directory: `.`
 - default location for working with files
- Parent Directory: `..`
 - directory immediately above the current directory

DIRECTORY CONTENT

- regular files
 - text, data
 - binaries, executables
 - links to other files or directories
- system files
 - device files: character or block special
 - networking endpoints:
 - sockets
 - FIFO
- directories

PATHS AND PATHNAMES

- path: list of directories separated by “/”

- Absolute Pathname

- Traces a path from root to a file or a directory
 - Always begins with the root (/) directory

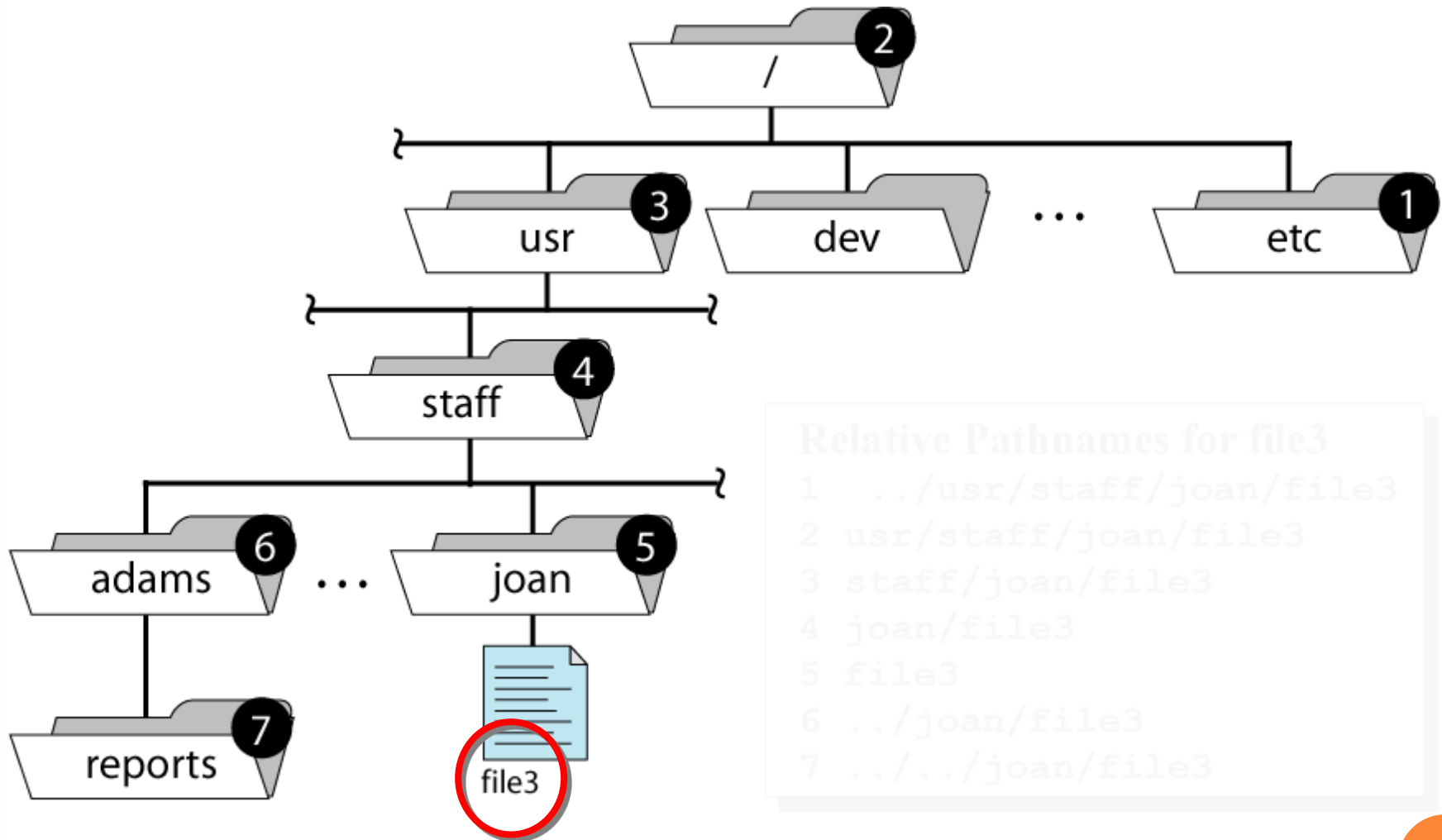
Example: /home/turing/ray/unix/assign1.txt

- Relative Pathname

- Traces a path from the current directory
 - No initial forward slash (/)
 - dot (.) refers to current directory
 - two dots (..) refers to one level up in directory hierarchy

Example: unix/assign1.txt

PATHNAMES FOR FILE3



Relative Pathnames for file3

```
1 ../usr/staff/joan/file3
2 usr/staff/joan/file3
3 staff/joan/file3
4 joan/file3
5 file3
6 ../joan/file3
7 ../../joan/file3
```

Absolute Pathname: `/usr/staff/joan/file3`

FILE SYSTEM COMMANDS

- Provide information

 - pwd

 - ls

- Change view

 - cd

- Change

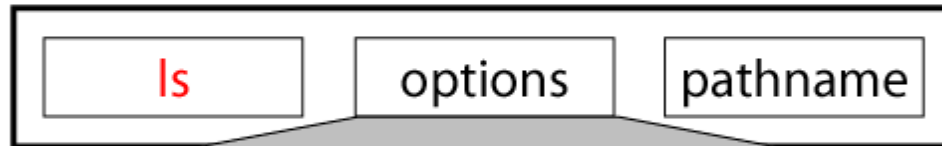
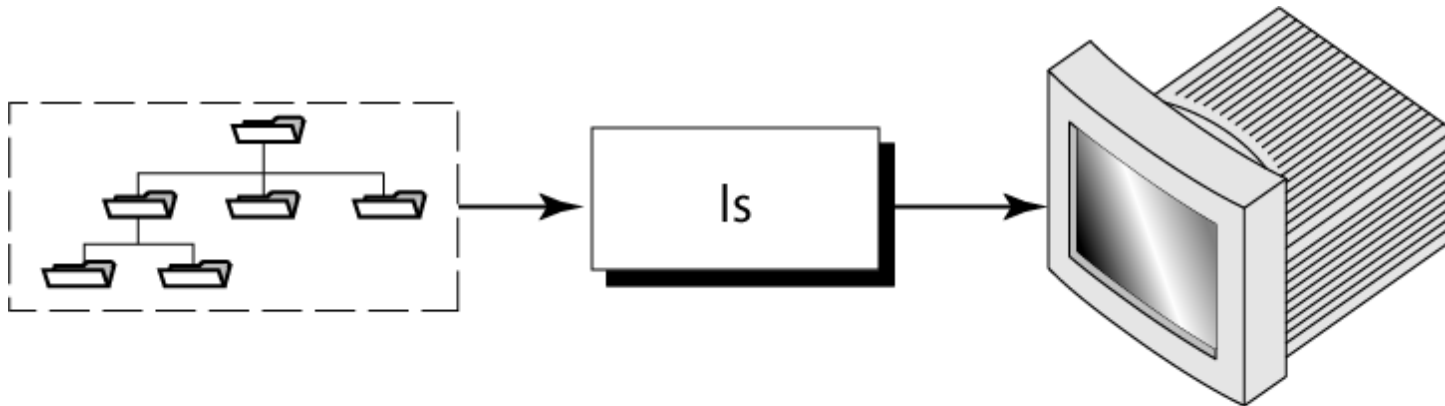
 - mkdir

 - rmdir

PATH TO CURRENT DIRECTORY

- “pwd” gives the full pathname of the current working directory
- pwd = print working directory
 - Example:
% pwd
/home/turing/ray/unix

LIST DIRECTORY CONTENT



-l: long list	-c: time—inode date
-d: working directory	-p: identify directories
-n: user/group id's	-R: recursive
-r: reverse order	-1: print one column
-t: time sequence	-i: print inode
-u: time—last access	

USEFUL DIRECTORY OPTIONS

- `ls -Alst | less`
 - `-A` include hidden files
 - “dot files” start with `.`
 - `-A` lists all except `.` and `..`
 - `-a` lists all including `.` and `..`
 - `-l` long format
 - `-s` show size of each file in blocks
 - 1K blocks on some systems
 - 1/2 K blocks on other systems
 - `-t` in reverse time sequence (most recent first)
 - piping into *less* prevents running off end of screen

LONG LIST OPTION

```
$ ls -l
```

```
total 2
```

```
-rw-r--r--
```

```
-rw-r--r--
```

```
1 gilberg staff
```

```
1 gilberg staff
```

```
12 May 17 08:45 f-t
```

```
12 May 17 08:45 f1t
```

↑ ↑
File Permissions
Type

↑ ↑ ↑
Links Owner Group

↑ ↑ ↑
File Last Mod Filename
Size

LIST EVERYTHING IN DIRECTORY

```
% ls -la
```

```
total 126
```

```
drwxr-xr-x 13 ray csci 1024 Apr 26 15:49 .
drwxr-xr-x 15 root root 512 Apr 24 15:18 ..
-rwx----- 1 ray csci 1120 Apr 12 13:11 .cshrc
-rwxr--r-- 1 ray csci 885 Dec 2 13:07 .login
-rw-r--r-- 1 ray csci 141 Mar 14 13:42 .logout
-rwx----- 1 ray csci 436 Apr 12 11:59 .profile
drwx----- 7 ray csci 512 May 17 14:11 330
drwx----- 3 ray csci 512 Mar 19 13:31 467
drwx----- 2 ray csci 512 Mar 31 10:16 Data
-rw-r--r-- 1 ray csci 80 Feb 27 12:23 quiz.txt
```

List contents of the current directory in long format

. is current dir.
.. is parent dir.

dot (.)
names
are
hidden
files

directories

plain file

LIST ALL IN A SPECIFIC DIRECTORY

```
% ls -l unix/grades
```

```
total 10
```

```
-rwxr-xr-x 3 ray  csci  72 Jan 19 19:12 330assign-graderun  
-rwxr-xr-x 1 ray  csci  70 Jan 19 19:13 330exam-graderun  
-rwxr-xr-x 2 ray  csci  70 Jan 19 19:12 330quiz-graderun  
-r-x----- 1 ray  csci 468 Feb  1 11:55 test-330grade  
-r-x----- 1 ray  csci 664 Feb  1 11:55 test-330grade,v
```

PERMISSIONS: TERMINOLOGY

○ user

- any one who has account on the system
- recognized via a number called “user id”

○ super user

- “root” user, also known as system administrator
- has user id “0”
- has the maximum set of privileges in the system, i.e. no restrictions apply to what “root” can do

○ group

- users are organized into groups
- user can belong to multiple groups

PERMISSIONS: CORE CONCEPTS

- user info is stored in file `/etc/passwd`
 - userid, user name, group, home directory, shell
 - passwords is listed in separate file: `/etc/shadow`
- group info is stored in file `/etc/group`
 - groupid, group name
 - additional group members
- To find out group information, use the command
named: `groups user-id`

Example:

```
% groups z036473
```

```
student csci
```

SECURITY LEVELS

- There are three levels of security in UNIX:
system, directory and file
- System security
 - controlled by the super user
- Directory and file
 - controlled by the user who owns them

ACCESS PERMISSION TYPES

3 general types of access permissions:

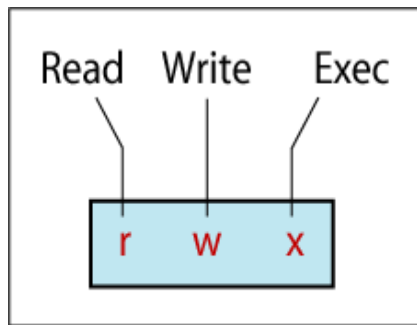
r	read
w	write
x	execute
-	permission denied

ACCESS PERMISSION TYPES

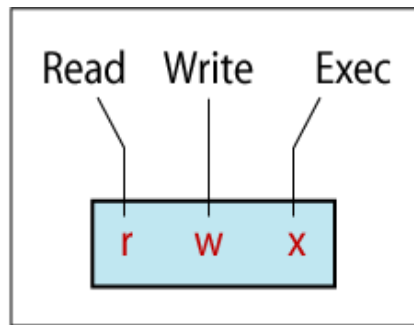
Access Type	Meaning on File	Meaning on Dir.
r (read)	View file contents (open, read)	List directory contents
w (write)	Change file contents	Change directory contents
x (execute)	Run executable file	Make it current directory Access files in it
-	Permission denied	Permission denied

CATEGORIES OF USERS

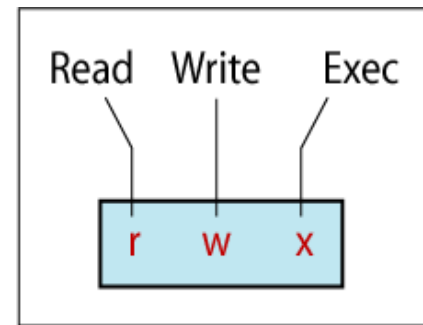
- 3 categories of users want access



User



Group



Others

CHECKING PERMISSIONS

- To check the permissions of an existing file or an existing directory, use the command: `ls -l`

Example:

```
% ls -l unix
```

```
total 387
```

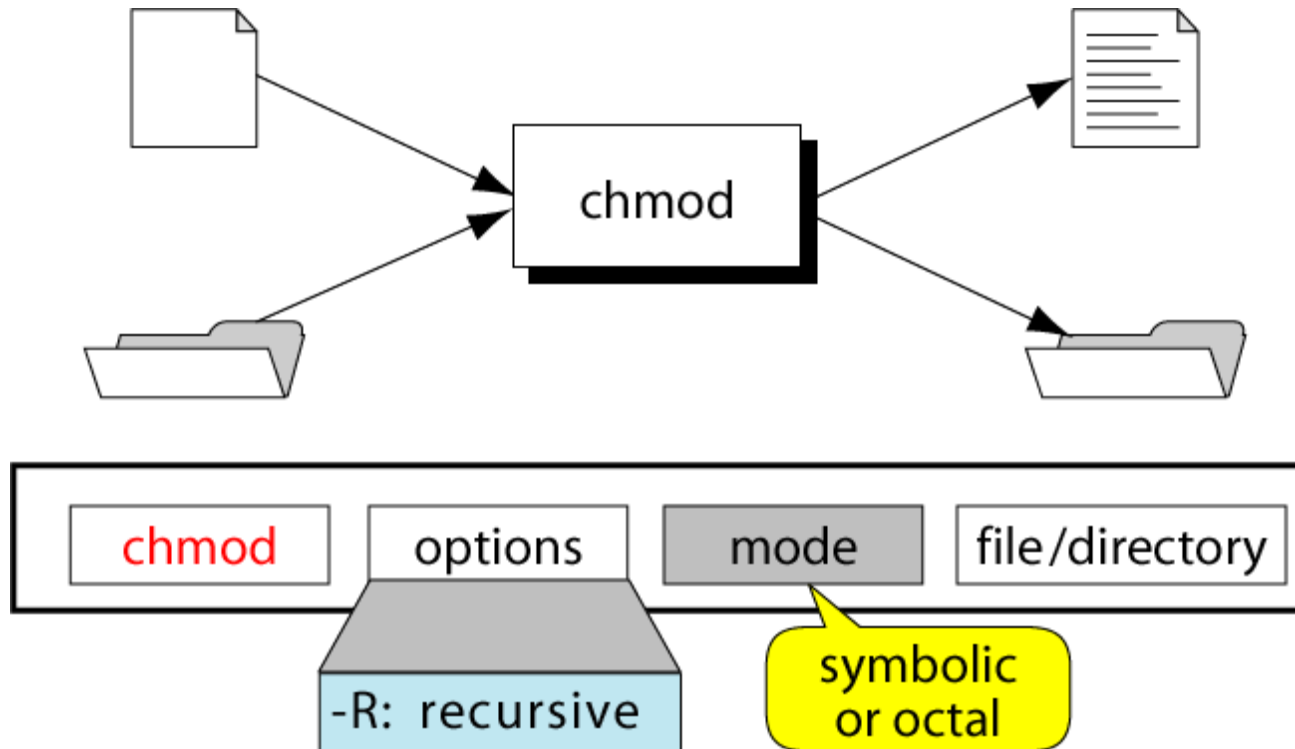
```
drwxr--r-- 1 z036473 student 862 Feb  7 19:22 grades
```

```
-rw-r--r-- 1 z036473 student  0 Jun 24  2007 uv.nawk
```

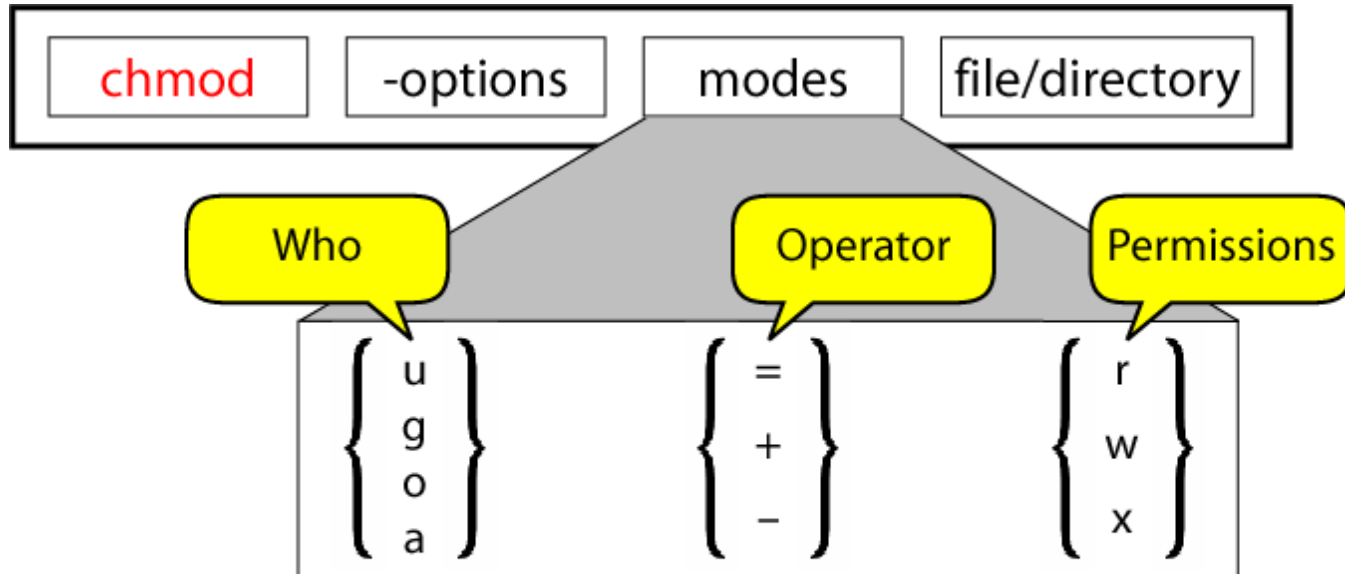
```
-rw-r--r-- 1 z036473 student  0 Jun 24  2007 wx.nawk
```

```
-rw-r--r-- 1 z036473 student  0 Jun 24  2007 yz.nawk
```

THE CHMOD COMMAND



CHANGING PERMISSIONS: SYMBOLIC MODE



Example

```
chmod u=rwx,g+w,o-w memo.doc
```

CHANGING PERMISSIONS: SYMBOLIC MODE

chmod who operation permissions filename

u for user
g for group
o for others
a for all

+ for add
- for remove
= for assign

r for read
w for write
x for execute

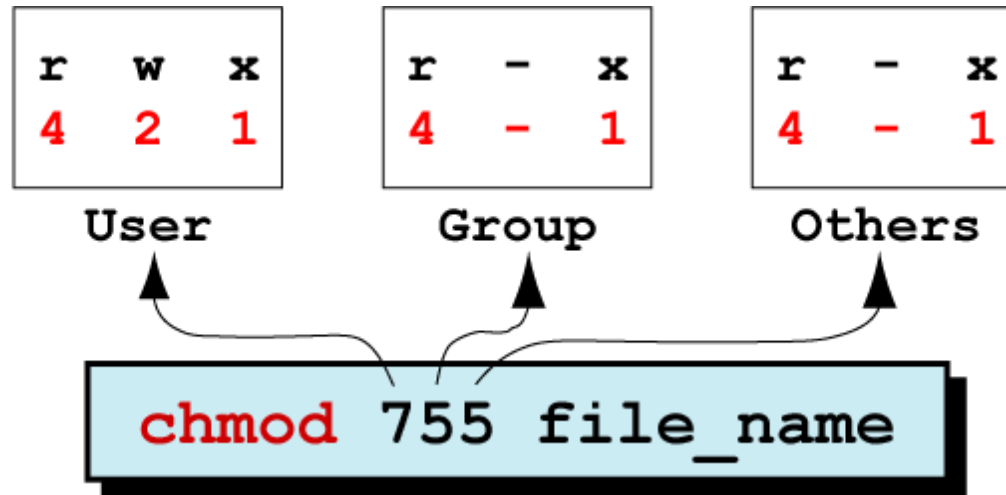
Examples:

```
% chmod ug=rwx,o=rx sort.c
```

```
% chmod ugo+rx,go+w sort.c
```

THE CHMOD COMMAND: OCTAL MODE

Permission
Octal Value



CHANGING PERMISSIONS: OCTAL MODE

Step	Perform...	Settings
1	List the desired setting	<code>rwX rwX r-X</code> <code>111 111 101</code> <code>421 421 401</code> <code>7 7 5</code> <code>chmod 775 sort.c</code>
2	Assign binary: 1 for access; 0 for no access	
3	List octal values for the corresponding binary 1's	
4	Convert the octal values to a 3- digit number	
5	Write the command	

```
% ls -l sort.c
```

```
-rwxrwxr-x 1 ege csci 80 Feb 27 12:23 sort.c
```

CHANGING PERMISSIONS: EXAMPLE

- Goal: set mode of file “myfile”
 - Read, write, and execute permissions to owner
 - Read and execute permissions to group
 - Execute permission to others

We want: `rwx|r-x|--x`

1) Using Symbolic Mode:

`chmod u=rwx,g=rx,o=x myfile`

2) Using Octal Mode:

`chmod 751 myfile`

PERMISSION DEFAULT

- `umask` (user mask)
 - governs default permission for files and directories
- `umask -S` `umask`
`u=rwx,g=rx,o=rx` `0022`
- in octal form is subtracted from:
 - 777 for a directory
 - 666 for a file
- can be set to individual value

Example:

```
% umask 011
```

USER MASK VALUES

User mask Value	Directory Default: 777	File Default: 666
000	777 (rwx rwx rwx)	666 (rw- rw- rw-)
111	666 (rw- rw- rw-)	666 (rw- rw- rw-)
222	555 (r-x r-x r-x)	444 (r- - r- - r- -)
333	444 (r- - r- - r- -)	444 (r- - r- - r- -)
444	333 (-wx -wx -rx)	222 (-w- -w- -w-)
555	222 (-w- -w- -w-)	222 (-w- -w- -w-)
666	111 (- -x - -x - -x)	000 (--- --- ---)
777	000 (--- --- ---)	000 (--- --- ---)

SPECIAL PERMISSIONS

- The regular file permissions (rwx) are used to assign security to files and directories
- Three additional special permissions can be optionally used on files and directories
 - Set User Id (SUID)
 - Set Group ID (SGID)
 - Sticky bit

SPECIAL PERMISSIONS: SUID

- SUID used for executable files
 - makes executable run with privileges of file owner, rather than the invoker
- Example:
 - “passwd” command and file “/usr/bin/passwd”

```
-rwsr-xr-x 1 root root 34888 2007-02-27 13:13 /usr/bin/passwd
```

- allows regular user access to system files while changing password

SPECIAL PERMISSIONS: SGID

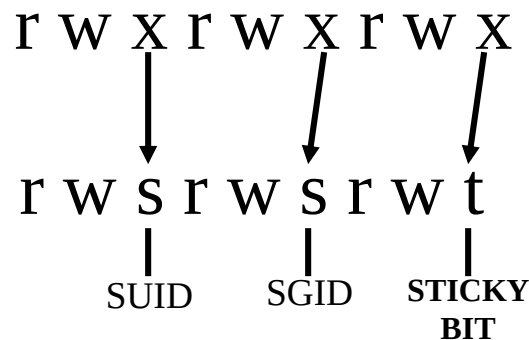
- logic is similar to SUID bit
- used for executable files
- runs program with group permission of file, rather than group of invoker
- Example:
if a file is owned by the system group and also has the SGID bit set, then any user who executes that file will be a member of the system group during the execution

SPECIAL PERMISSIONS: STICKY BIT

- not clearly defined
- for executable files:
 - executable is kept in memory even after it ended
- for directories:
 - file can only be deleted by the user that created it

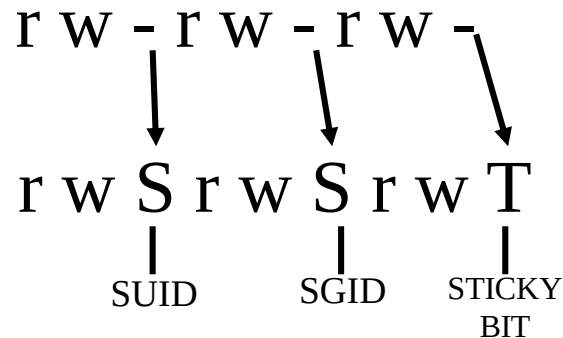
SPECIAL PERMISSIONS: DISPLAY

- “ls -l” command does not have a section for special permission bits
- however, since special permissions required “execute”, they mask the execute permission when displayed using the “ls -l” command.



SPECIAL PERMISSIONS: DISPLAY

- If special permission bits are set on a file or a directory without “execute” permission, the special permissions are shown in capital letters



SETTING SPECIAL PERMISSIONS

suid	sgid	stb	r	w	x	r	w	x	r	w	x
4	2	1	4	2	1	4	2	1	4	2	1
7			7			7			7		
Special			user			group			others		

Use the “chmod” command with octal mode:

`chmod 7777 filename`

SETTING SPECIAL PERMISSIONS

- `chmod` with symbolic notation:

<code>+s</code>	add SUID and SGID
<code>g+s</code>	add SGID
<code>g-s</code>	remove SGID
<code>u+s</code>	add SUID
<code>u-s</code>	remove SUID
<code>+t</code>	set sticky bit

SUMMARY

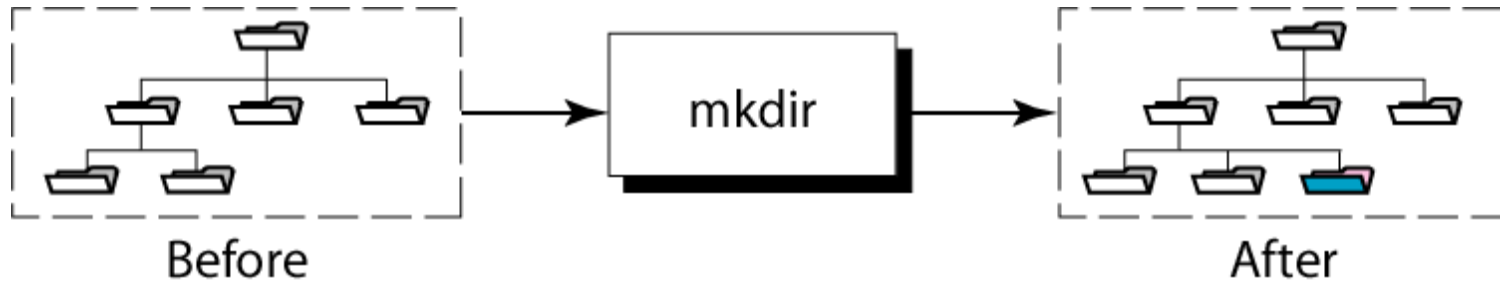
- r, w, x
 - and extra bits
- user, group, world
- user mask

FILE NAME EXPANSION

- Wildcard characters allow to select files that satisfy a particular name pattern

Character	Description	Example
*	Match zero or more char.	ls *.c
?	Match any single character	ls conf.?
[<i>list</i>]	Match any single character in <i>list</i>	ls conf.[co]
[lower-upper]	Match any character in range	ls lib-id[3-7].o
<i>str</i>{<i>str1</i>,<i>str2</i>,...}	Expand <i>str</i> with contents of { }	ls c*.{700,300}

CREATING A NEW DIRECTORY



Syntax:

`mkdir [ -p ] directory-list`

MKDIR EXAMPLES

Can create one or more directories at a time

-p = creates intermediate directories if necessary

Examples:

```
% mkdir csci330
```

```
% mkdir dirOne dirTwo
```

```
% mkdir /home/turing/ray/unix/demo
```

(intermediate directories must already exist)

```
% mkdir -p /home/turing/ray/unix/demo
```

(creates intermediate directories if needed)

FILE AND DIRECTORY NAMES

- Use the following characters:
 - Uppercase letters (A-Z)
 - Lowercase letters (a-z)
 - Numbers (0-9)
 - Underscore (_)
 - Period/dot (.)

FILE AND DIRECTORY NAMES

- avoid the following characters:

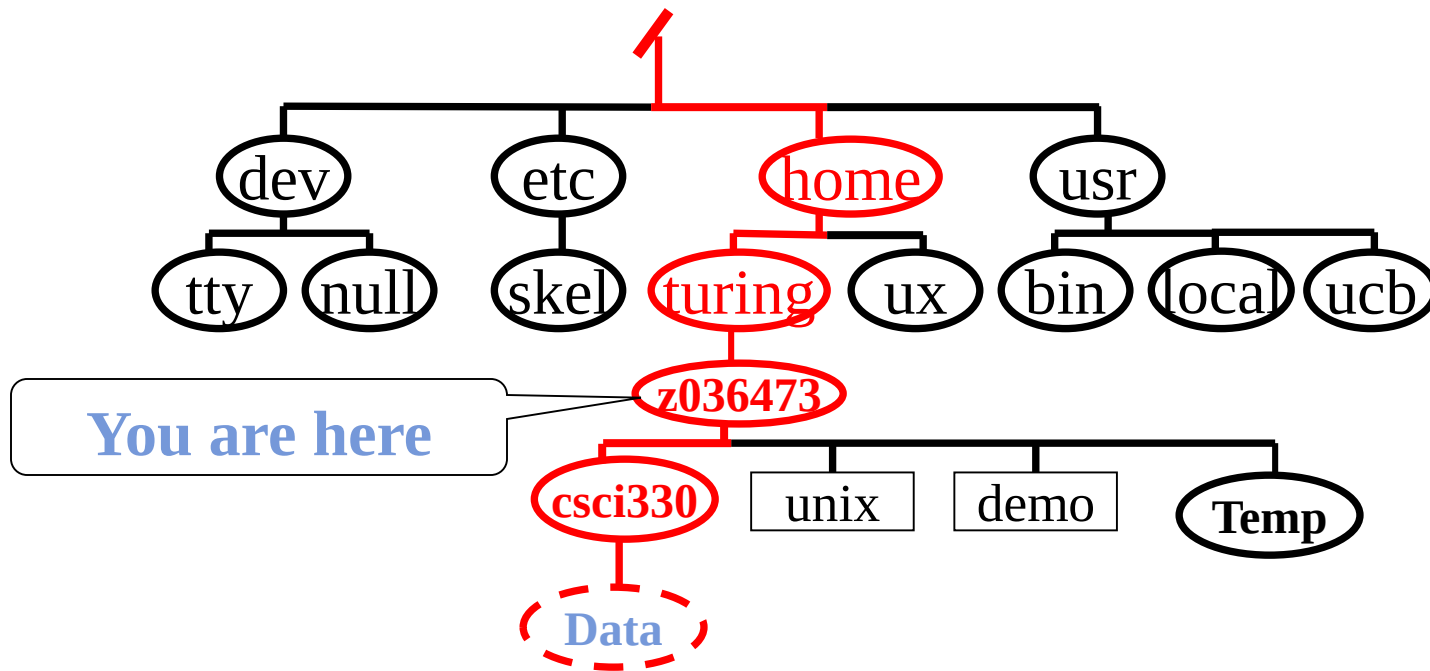
& * \ | [] { }

\$ < > () # ? /

“ ‘ ; ^ ! ~

Space Tab

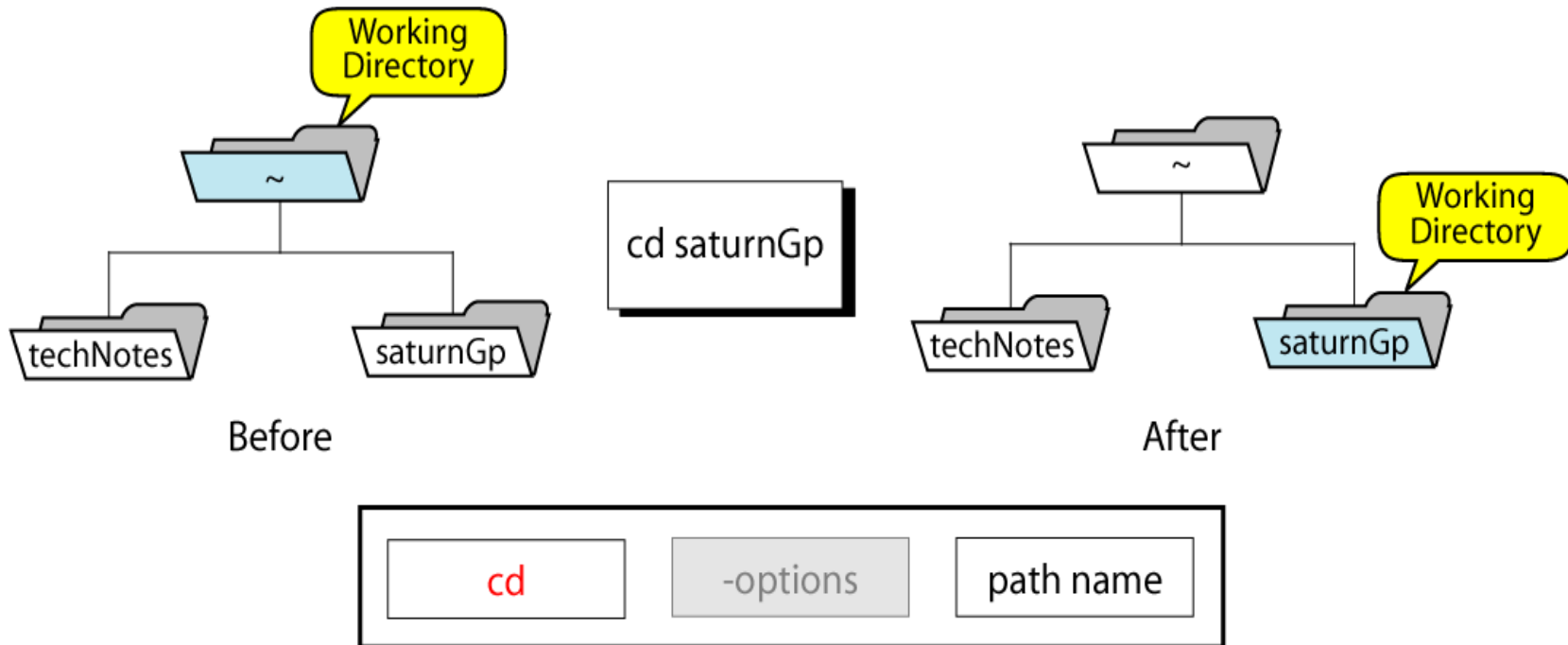
EXAMPLE: CREATE A DIRECTORY



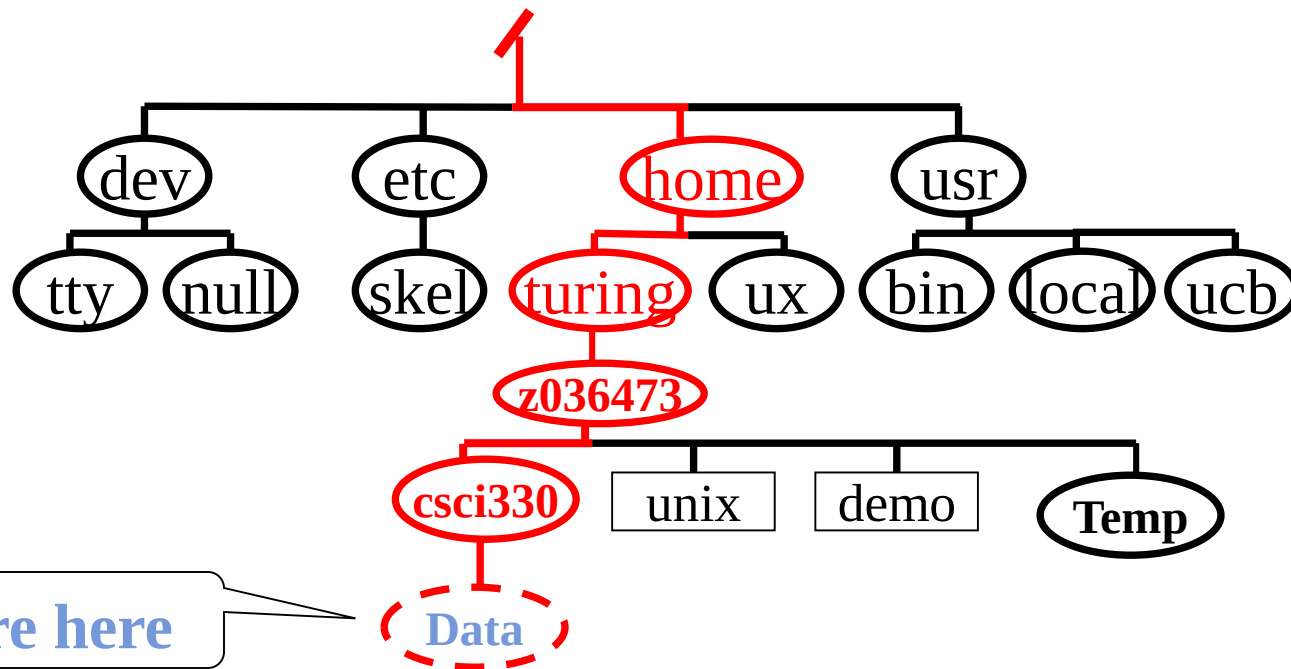
- to create directory called Data under csci330
- Absolute Pathname: `mkdir /home/turing/z036473/csci330/Data`
- Relative Pathname: `mkdir csci330/Data`

CHANGING DIRECTORY

(just changes the view!)



CHANGING DIRECTORY

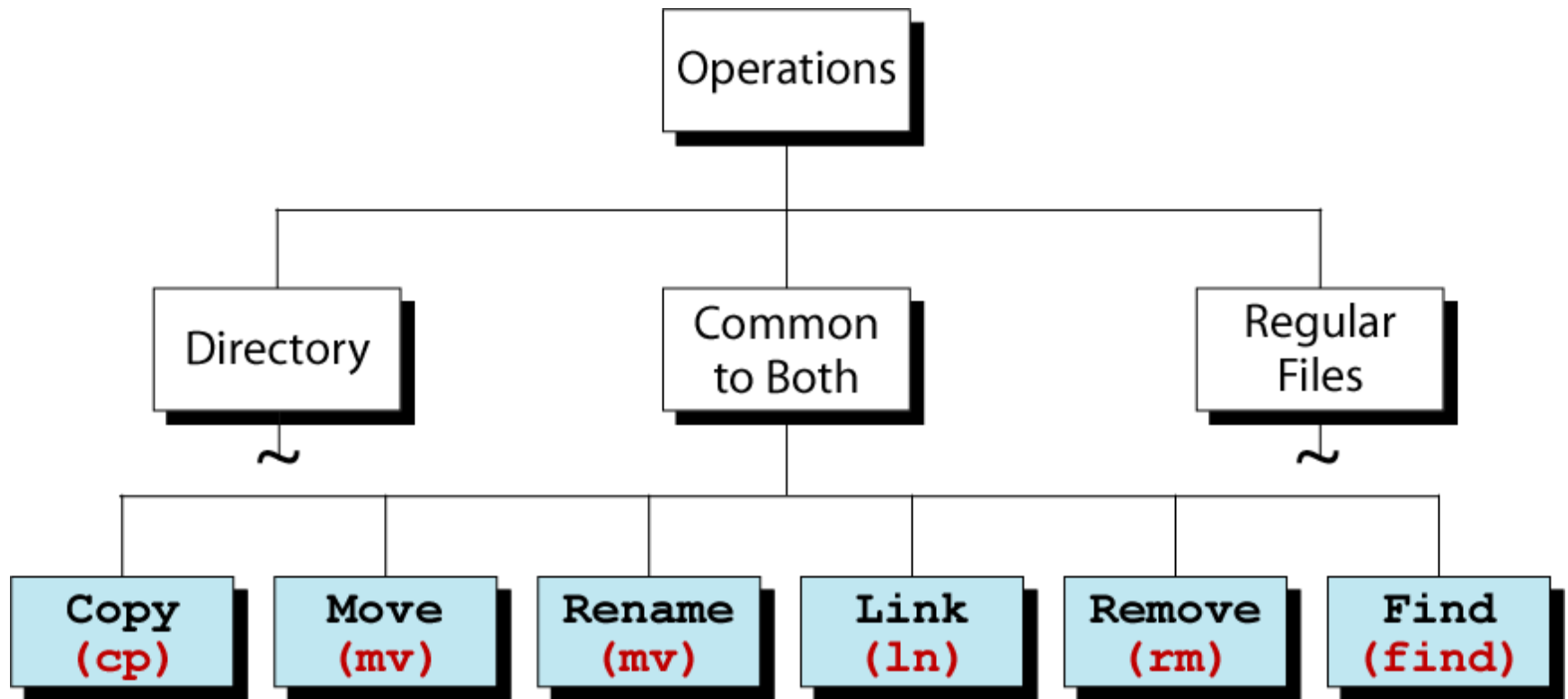


- from the Data directory, go to home directory
- Absolute Pathname: `cd /home/turing`
- Relative Pathname: `cd ~` `cd` `cd ../../..`

REMOVE DIRECTORIES

- If empty, use “rmdir”
 - Example: To remove an empty directory called “test”
`% rmdir test`
- if non-empty, use “rm -r”
 - Example: To remove non-empty directory “old-data”
`% rm -r old-data`
Safer to just delete the contents first

FILE SYSTEM COMMANDS



COPYING FILES

- To copy a file, use “cp”

Syntax: cp source-file target

- Commonly used options:

- i if “target” exists, the command cp prompts for confirmation before overwriting

- i is not the default, but should be
you can make -i the default*

- p preserve permissions and modification times

- p is not the default, but should be
you can make -p the default*

- r recursively copy files and subdirectories

COPYING A FILE

- Make a copy of a file

```
% cp assign1.txt assign1.save
```

- Copy “assign1.txt” to a different directory

```
% cp assign1.txt ~/archive
```

```
% cp assign1.txt ~/archive
```

but suppose archive isn't a directory

suppose it doesn't exist

- Copy “assign1.txt” to a new name in a different directory

```
% cp assign1.txt ~/archive/assign1.save
```

COPYING MULTIPLE FILES

Syntax: `cp source-files destination-directory`

```
% cp assign1.txt assign2.txt ~/archive
```

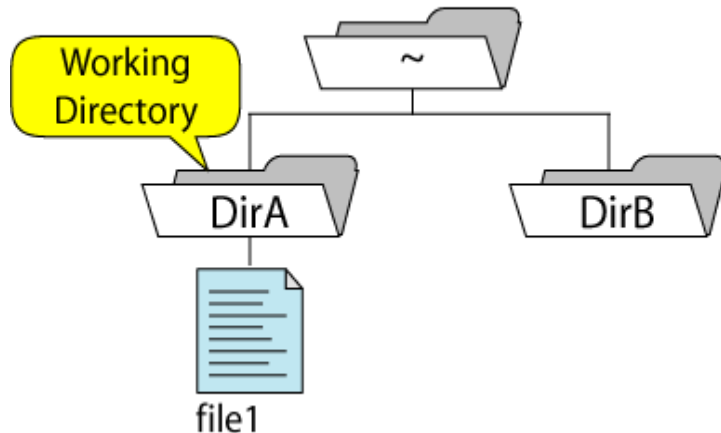
```
% cp assign?.txt ~/archive
```

- Files will have same name in destination directory

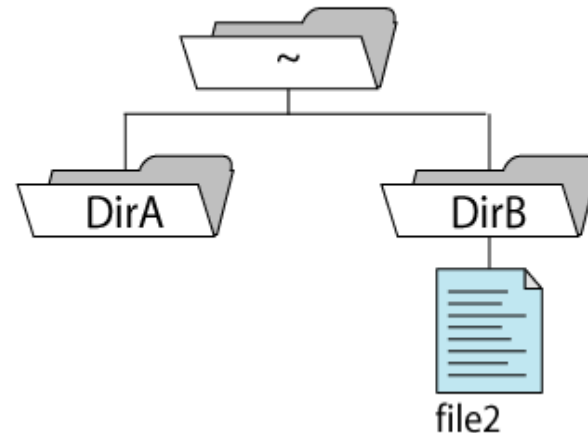
MOVING FILES

- To move files from one directory to another directory, or to re-name a file, use: “mv”

```
$ mv file1 ~/DirB/file2
```



(a) Before



(b) After

MOVING A FILE

- Move “assign1.txt” a different directory
 - If the destination file exists, “mv” will not overwrite exiting file:

```
% mv assign1.txt ~/archive
```

- Move “assign1.txt” a different directory and rename it to “assign1.save”

```
% mv assign1.txt ~/archive/assign1.save
```

MOVING MULTIPLE FILES

Syntax: mv source-files destination-directory

```
% mv assign1.txt assign2.txt ~/archive
```

```
% mv assign?.txt ~/archive
```

- Files will have same name in destination directory

RENAMING FILES OR DIRECTORIES

- use “mv”
- Example: rename file “unix” to “csci330”

```
% mv unix csci330
```

Caveat: what if “csci330” exists and is a directory ?

DELETING FILES

Syntax: `rm file-list`

- Commonly used options:
 - f force remove regardless of permissions
 - i prompt for confirmation before removing
 - r removes everything under the indicated directory

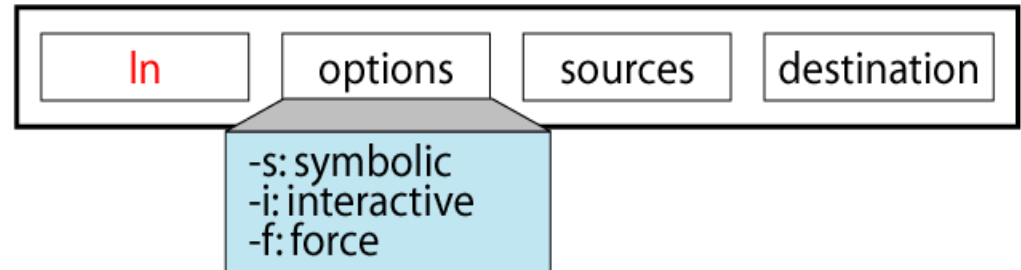
Example: remove file “old-assign”

```
% rm unix/assign/old-assign
```

LINKING FILES

- Allows one file to be known by different names
- A link is:
 - A reference to a file stored elsewhere on the system
 - A way to establish a connection to a file to be shared
- Two types:
 - Hard link
 - Symbolic link (a.k.a. “soft link”)

THE LN COMMAND



- hard link:

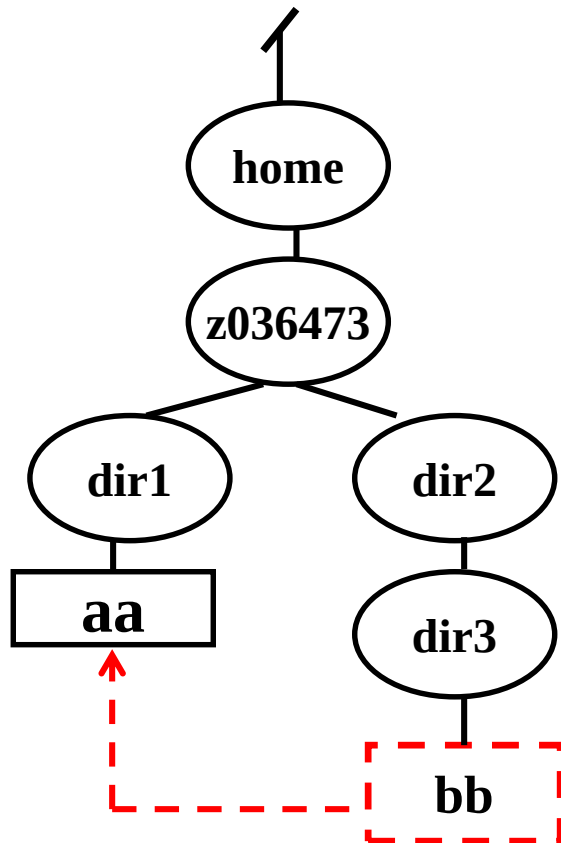
`ln shared-file link-name`

- symbolic link:

`ln -s shared-file link-name`

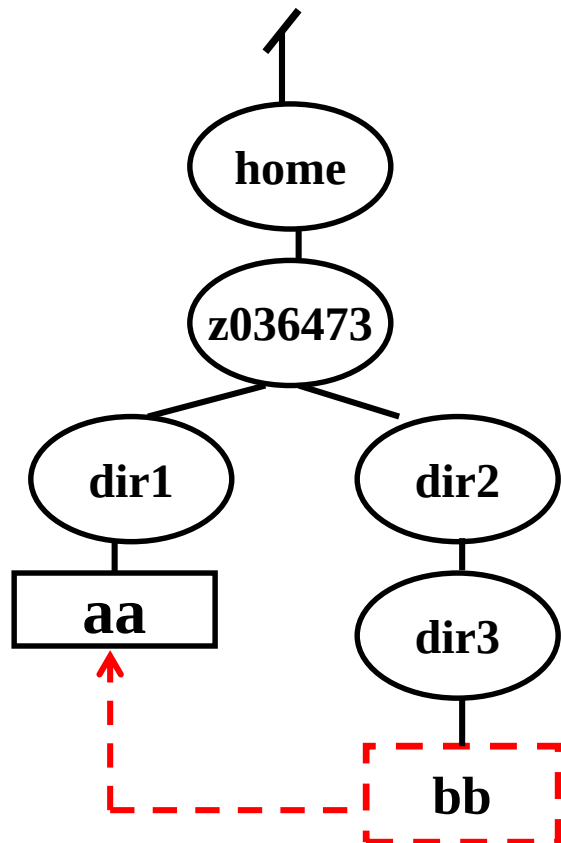
Note that the long listing (`ls -l`) of a soft link does not accurately reflect its associated permissions. To view the permissions of the file or directory that the symbolic link references, use the `-L` option of the `ls` command.

LINK ILLUSTRATION



- create entry “bb” in “dir3” as link to file “aa” in “dir1”

HARD LINK EXAMPLE

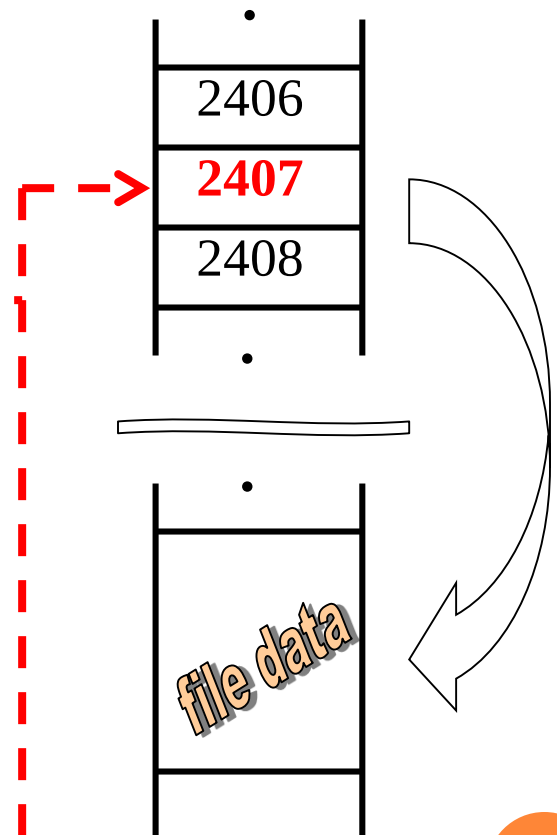


Contents of **dir1**

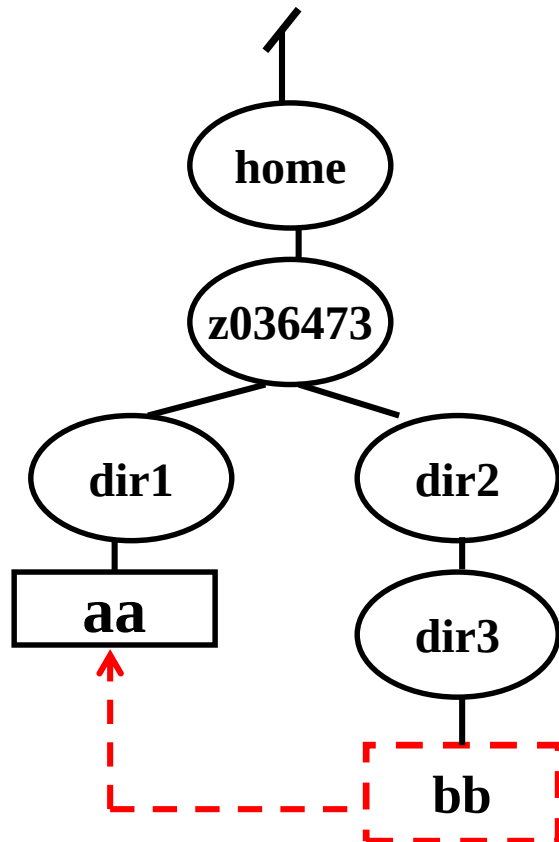
.	1076
..	2083
aa	2407

Contents of **dir3**

.	1070
..	2050
bb	2407



SYMBOLIC LINK EXAMPLE

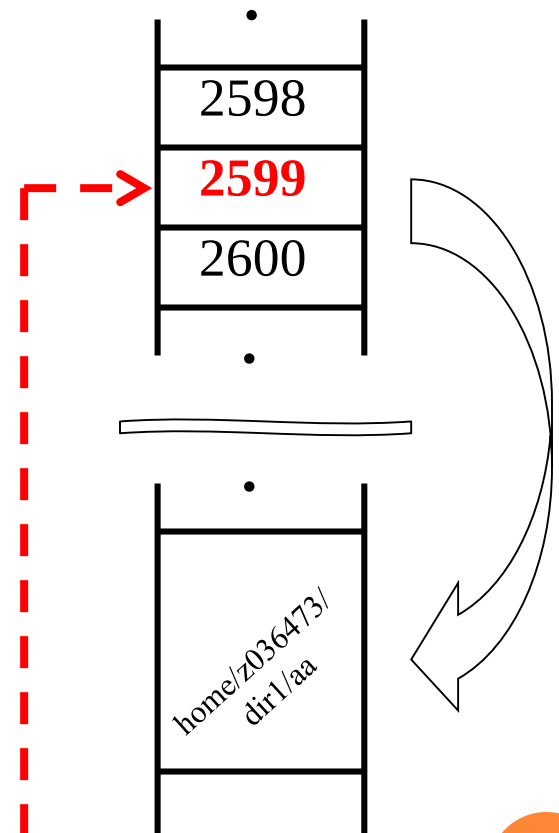


Contents of **dir1**

.	1076
..	2083
aa	2407

Contents of **dir3**

.	1070
..	2050
bb	2599



HARD LINK VS. SYMBOLIC LINK

Hard Link Advantages	Symbolic Link Advantages
• Checks for the existence of the original file • The original file continues to exist as long as at least one directory contains it	• Can use either relative or absolute path to access the original file • Can cross physical file systems

HARD LINK VS. SYMBOLIC LINK

Hard Link Disadvantages	Symbolic Link Disadvantages
• Cannot link to a file in a different file system • Prevents deleting file if another link is exists	• Created without checking the existence of the shared file • Cannot access the shared file if its path has restricted permissions • Can be circular linked to another symbolic linked file

FINDING FILES

- The command named “find” can be used to locate a file or a directory

Syntax: `find pathname-list -name expression`

- “find” recursively descends through pathname-list and applies expression to every file

FINDING FILES

- Example 1: Find all files, in your directory hierarchy, that have name ending with “.bak”

```
% find ~ -name "*.bak" -print
```

- Example 2: Find all files, in your directory hierarchy, that were modified yesterday

```
% find ~ -mtime -1 -print
```