

Lecture 2

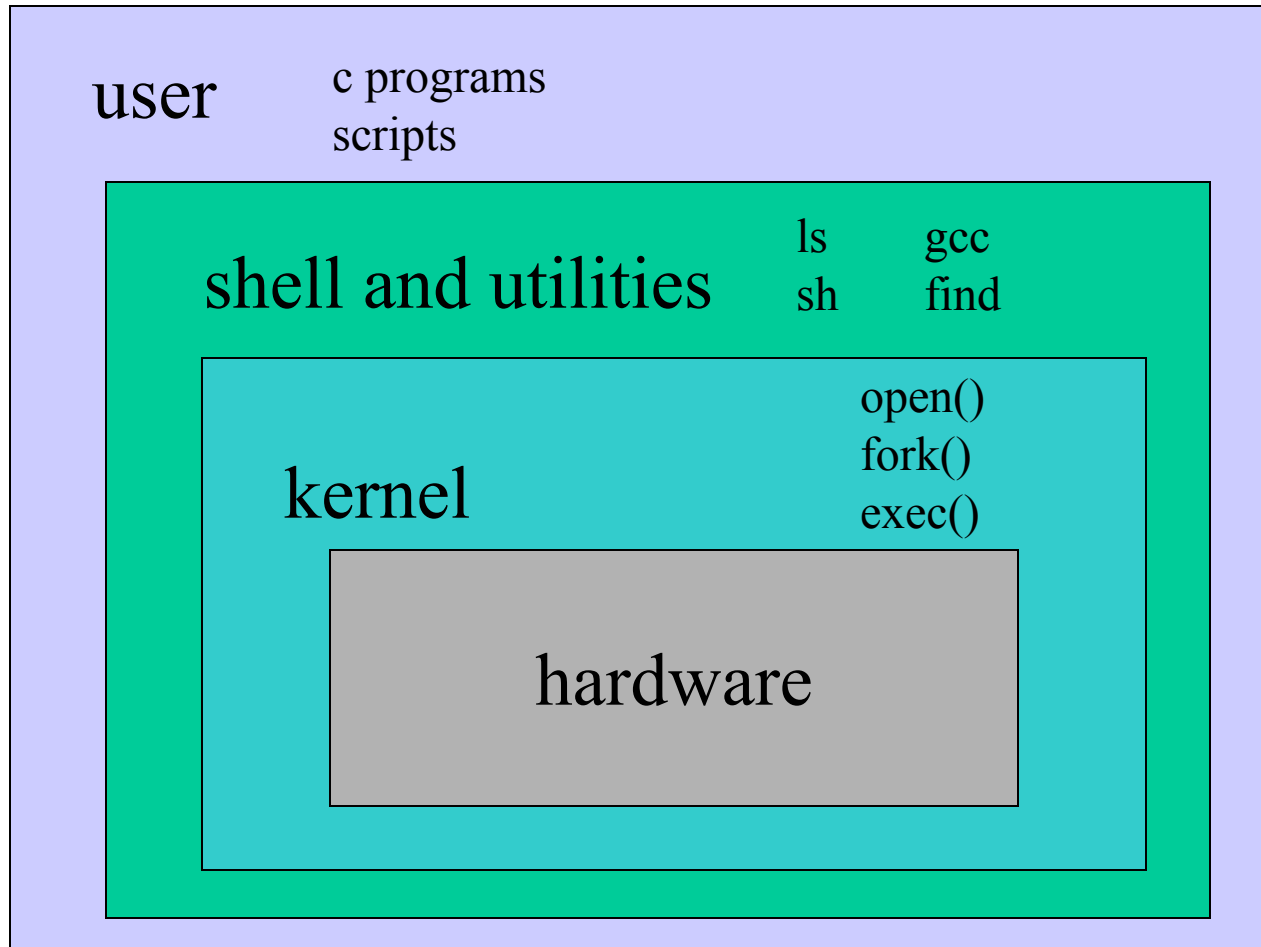
UNIX Basics

The UNIX Filesystem

On the last episode of Open Source Tools...

- Course Info
- History of UNIX and Open Source
- Highlights of UNIX
- The UNIX Philosophy
- System organization

Unix System Structure



Kernel Subsystems

- File system
 - Deals with all input and output
 - Includes files and terminals
 - Integration of storage devices
- Process management
 - Deals with programs and program interaction
 - How processes share CPU, memory and signals
 - Scheduling
 - Interprocess Communication
 - Memory management
- UNIX variants have different implementations of different subsystems.

What is a shell?

- The user interface to the operating system
- Functionality:
 - Execute other programs
 - Manage files
 - Manage processes
- A program like any other
- Executed when you log on



Most Commonly Used Shells

/bin/sh	The Bourne Shell / POSIX shell
/bin/csh	C Shell
/bin/tcsh	Enhanced C Shell
/bin/ksh	Korn shell
/bin/bash	Open Source ksh clone

Basic form of shell:

```
while (read command) {  
    parse command  
    execute command  
}
```

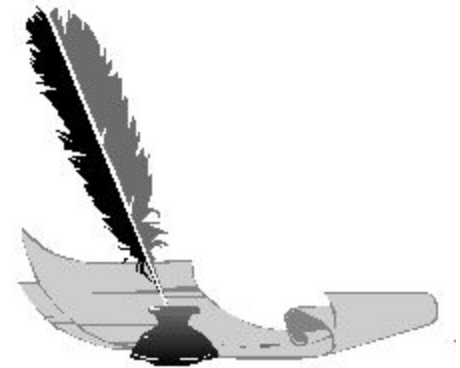
Shell Interactive Use

When you log in, you *interactively* use the shell:

- Command history
- Command line editing
- File expansion (tab completion)
- Command expansion
- Key bindings
- Spelling correction
- Job control

Shell Scripting

- A set of shell commands that constitute an executable *program*
- A shell script is a regular text file that contains shell or UNIX commands
- Very useful for automating repetitive task and administrative tools and for storing commands for later execution



Simple Commands

- *simple command*: sequence of non blanks arguments separated by blanks or tabs.
- 1st argument (numbered zero) usually specifies the name of the command to be executed.
- Any remaining arguments:
 - Are passed as arguments to that command.
 - Arguments may be filenames, pathnames, directories or special options (up to command)
 - Special characters are interpreted by shell

A simple example

```
$ ls -l /bin
-rwxr-xr-x 1 root sys 43234 Sep 10 2008 date
$
```

The diagram illustrates the components of a shell command. Red lines connect the parts of the command `$ ls -l /bin` to labels below: the `$` is labeled *prompt*, `ls` is labeled *command*, and `-l /bin` is labeled *arguments*.

- Execute a basic command
- Parsing a command into arguments is called *splitting*

Types of Arguments

```
$ tar -c -v -f archive.tar main.c main.h
```

- Options / Flags
 - Convention: *-X* or *--longname*
- Parameters
 - May be files, may be strings
 - Depends on command

Getting Help on UNIX

- **man**: display entries from UNIX online documentation
- **whatis, apropos**
- Manual entries organization:
 1. Commands
 2. System calls
 3. Subroutines
 4. Special files
 5. File format and conventions
 6. Games

http://en.wikipedia.org/wiki/Unix_manual

Example Man Page

ls (1)

USER COMMANDS

ls (1)

NAME

ls - list files and/or directories

SYNOPSIS

ls [*options*] [*file ...*]

DESCRIPTION

For each directory argument **ls** lists the contents; for each file argument the name and requested information are listed. The current directory is listed if no file arguments appear. The listing is sorted by file name by default, except that file arguments are listed before directories.

...

OPTIONS

-a, --all

List entries starting with **.**; turns off **--almost-all**.

-F, --classify

Append a character for typing each entry.

-l, --long|verbose

Use a long listing format.

-r, --reverse

Reverse order while sorting.

-R, --recursive

List subdirectories recursively.

SEE ALSO

chmod(1), find(1), getconf(1), tw(1)

Quoting

- The shell interprets special characters (e.g. many punctuation characters). To avoid, use quotes:

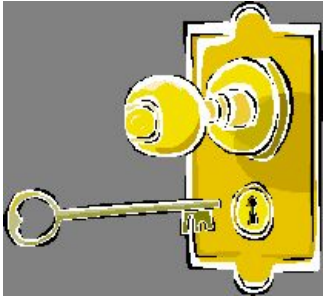
```
$ ls ; date
```

```
$ ls ";" date
```

```
$ ls 'file with spaces'
```

Today

- Discuss several commands relating to:
 - Security
 - File system



Fundamentals of Security

- UNIX systems have one or more users, identified with a number and name.
- A set of users can form a group. A user can be a member of multiple groups.
 - ★ A special user (id 0, name **root**) has complete control.
 - ★ Each user has a primary (default) group.



How are Users & Groups used?

- Used to determine if file or process operations can be performed:
 - Can a given file be read? written to?
 - Can this program be run?
 - Can I use this piece of hardware?
 - Can I stop a particular process that's running?

A simple example

```
$ ls -l /bin
```

```
-rwxr-xr-x 1 root sys 43234 Sep 26 2001 date
```

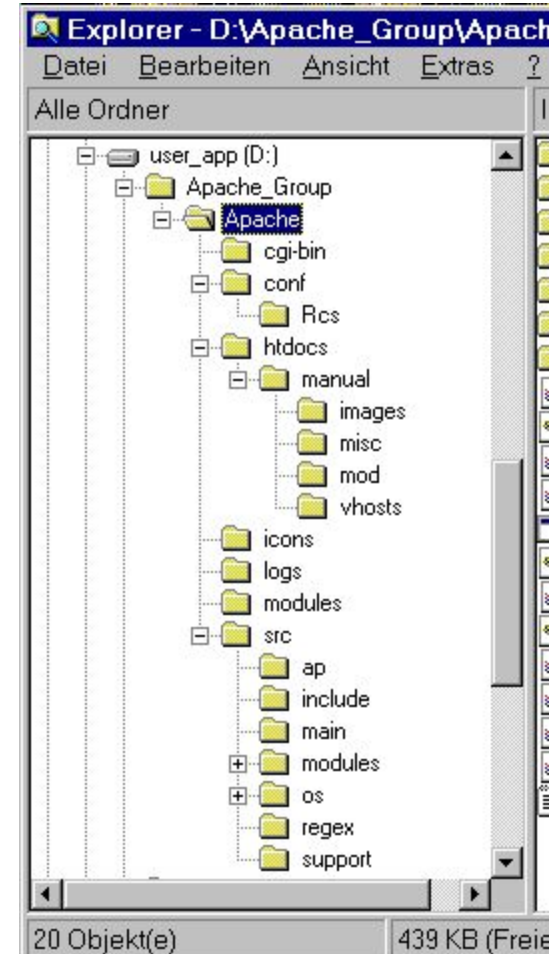
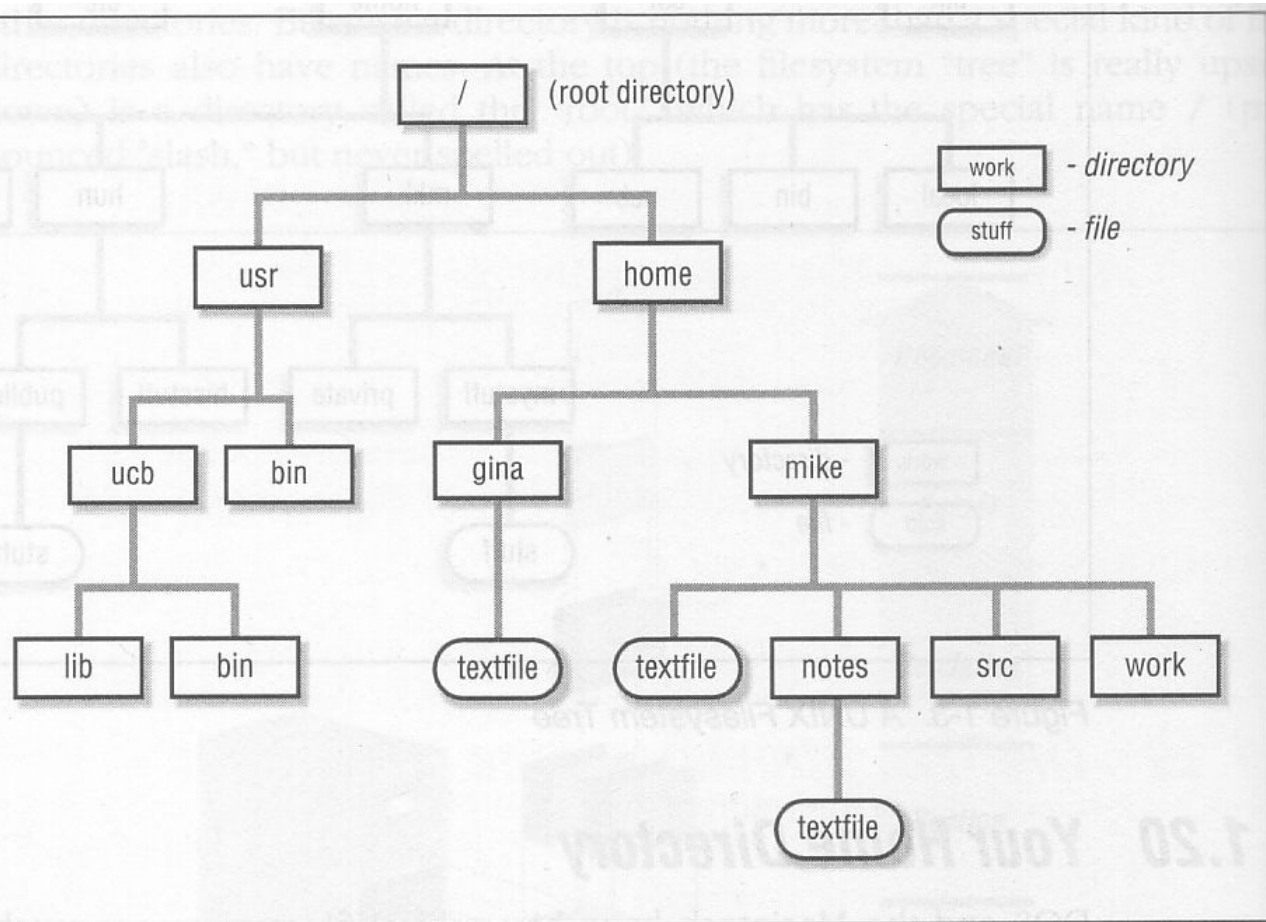
```
$
```

read

write

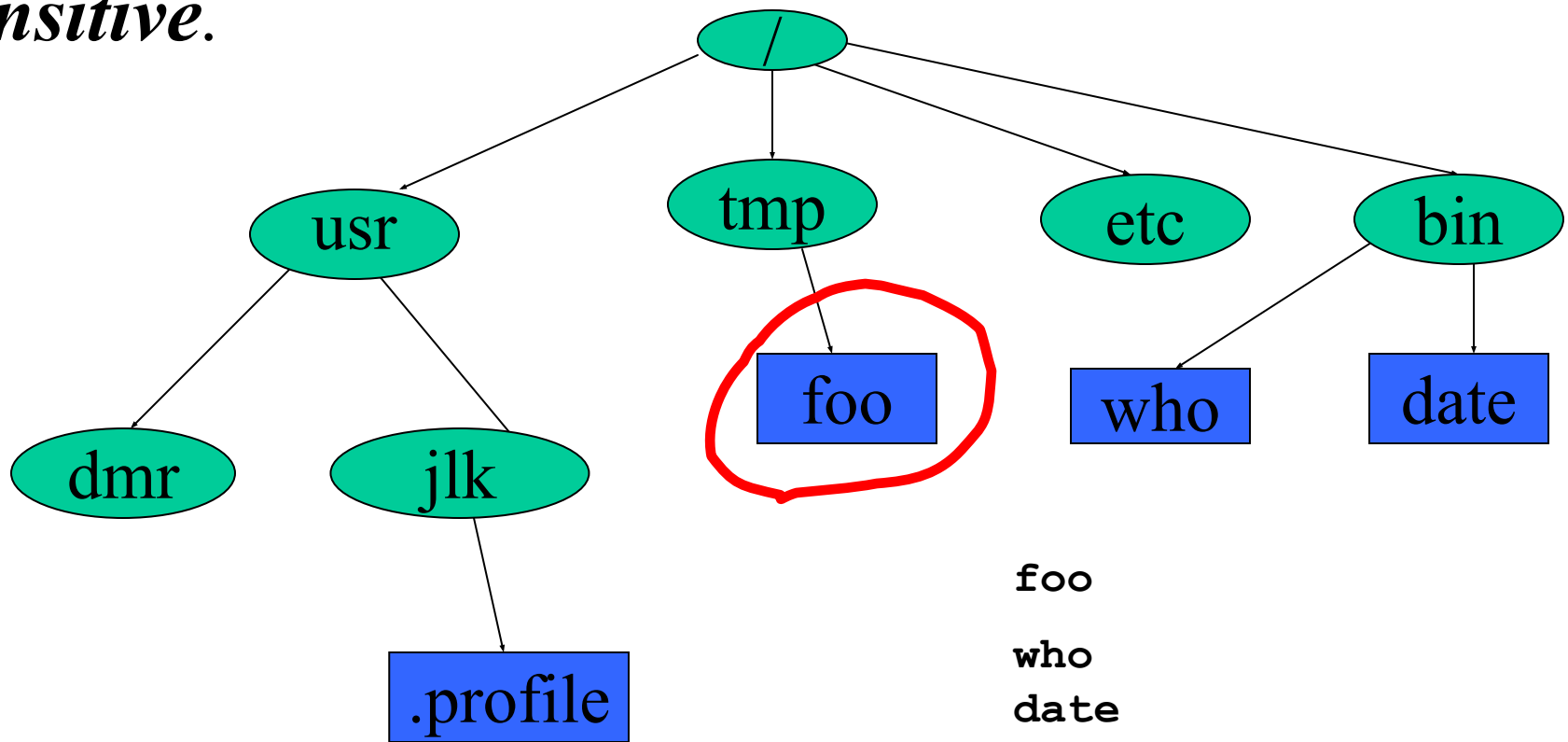
execute

The UNIX File Hierarchy



Definition: Filename

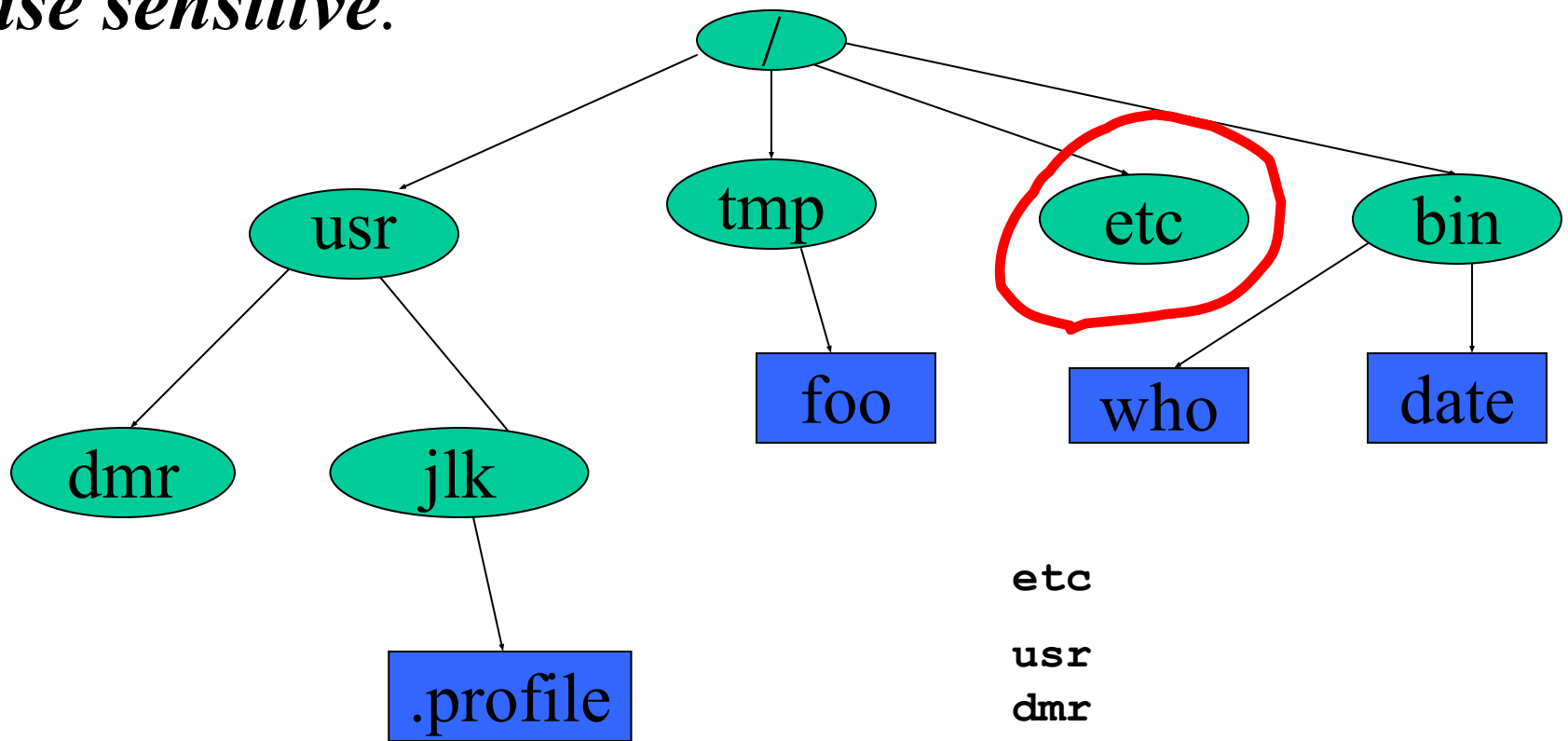
A sequence of characters other than slash. Case sensitive.



Definition: Directory

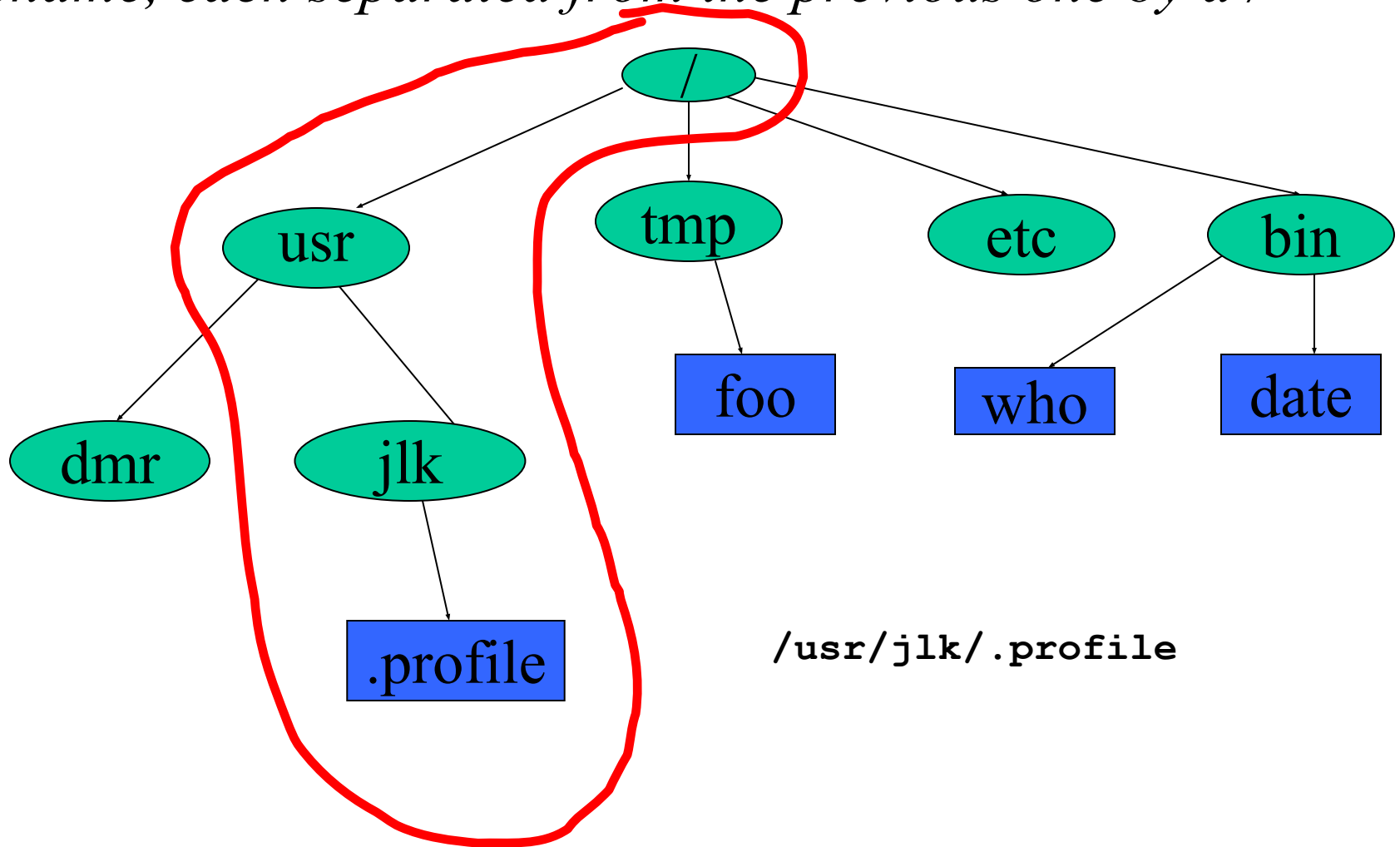
Holds a set of files or other directories.

Case sensitive.



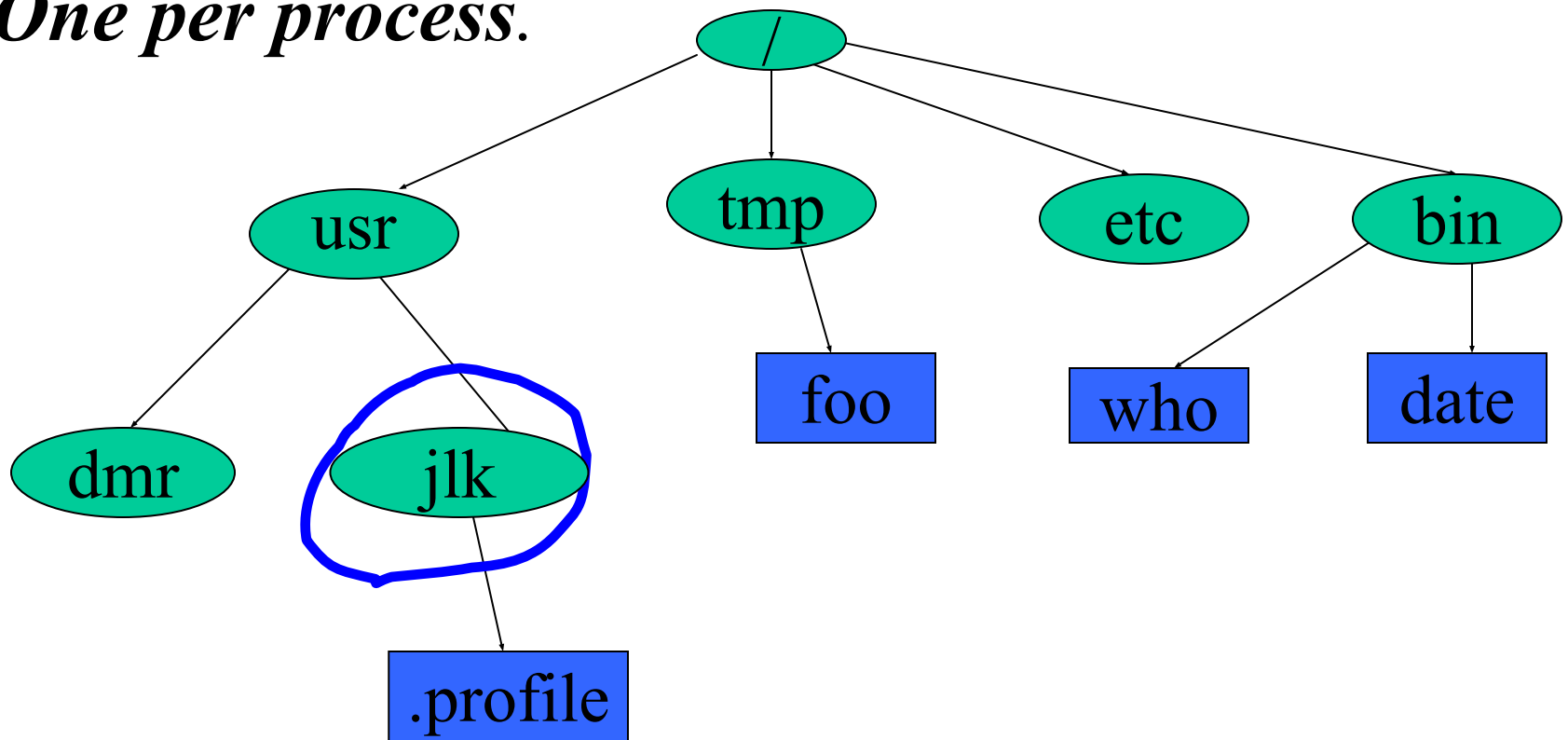
Definition: Pathname

A sequence of directory names followed by a simple filename, each separated from the previous one by a /



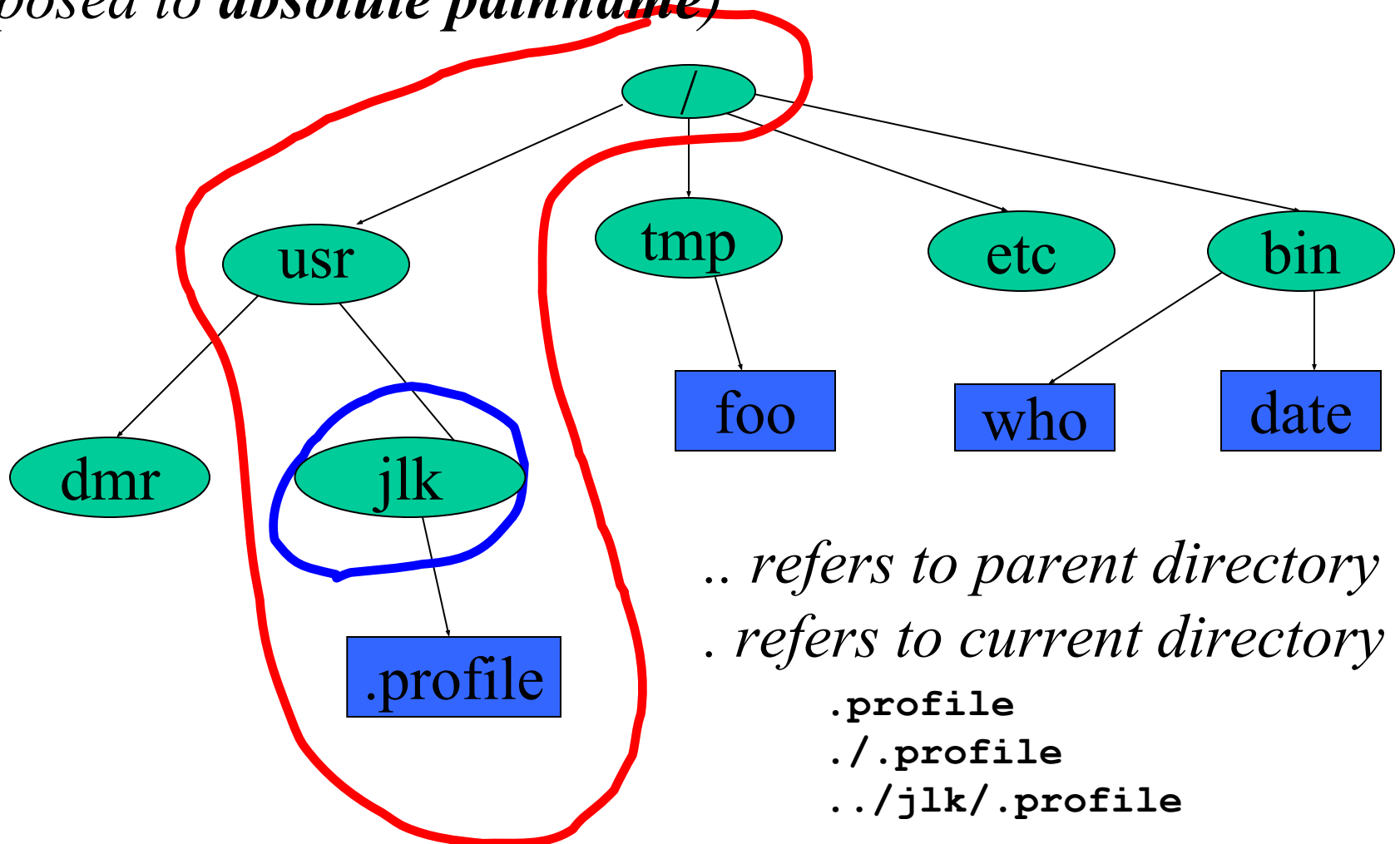
Definition: Working Directory

*A directory that file names refer to by default.
One per process.*



Definition: Relative Pathname

*A pathname relative to the working directory (as opposed to **absolute pathname**)*



Files and Directories

- Files are just a sequence of bytes
 - No file types (data vs. executable)
 - No sections
 - Example of UNIX philosophy
- Directories are a list of files and status of the files:
 - Creation date
 - Attributes
 - etc.

Tilde Expansion

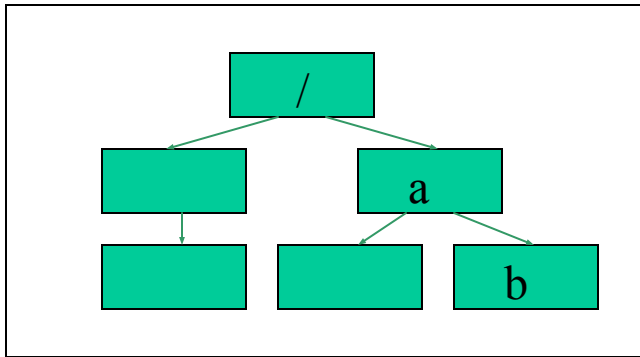
- Each user has a *home* directory
- Most shells (bash, csh) support `~` operator:
 - `~` expands to my home directory
 - `~/myfile` → `/home/kornj/myfile`
 - `~user` expands to user's home directory
 - `~unixtool/file2` → `/home/unixtool/file2`
- Useful because home directory locations vary by machine

Mounting File Systems

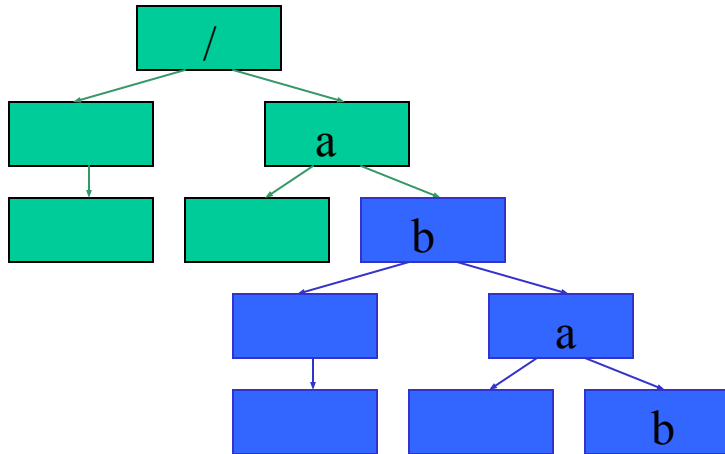
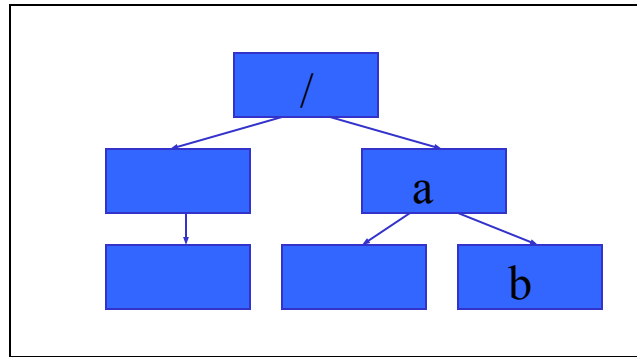
- When UNIX is started, the directory hierarchy corresponds to the file system located on a single disk called the *root device*.
- *Mounting* allows root to splice the root directory of a file system (e.g. usb drive) into the existing directory hierarchy.
- File systems created on other devices can be attached to the original directory hierarchy using the mount mechanism.
- Commands **mount** and **umount** manage



Mounting File Systems

root device



external device



Device	Mount Point
	/
	/a/b

Mount table

Printing File Contents

- The **cat** command can be used to copy the contents of a file to the terminal. When invoked with a list of file names, it **concatenates** them.
- Some options:
 - **-n** number output lines (starting from 1)
 - **-v** display control-characters in visible form (e.g. ^C)
- Interactive commands **more** and **less** show a page at a time

Common Utilities for Managing files and directories

- **pwd** print process working dir
- **ed, vi, emacs...** create/edit files
- **ls** list contents of directory
- **rm** remove file
- **mv** rename file
- **cp** copy a file
- **touch** create an empty file or update
- **mkdir** and **rmdir** create and remove dir
- **wc** counts the words in a file
- **file** determine file contents
- **du** directory usage

File Permissions

- UNIX provides a way to protect files based on users and groups.
- Three **types** of permissions:
 - read, process may read contents of file
 - write, process may write contents of file
 - execute, process may execute file
- Three **sets** of permissions:
 - permissions for owner
 - permissions for group (1 group per file)
 - permissions for other

Directory permissions

- Same types and sets of permissions as for files but different meanings:
 - **read**: process may read the directory *listing* (i.e., list files)
 - **write**: process may add/remove files in the directory
 - **execute**: process may open files in directory or subdirectories

Utilities for Manipulating file attributes

- **chmod** change file permissions
- **chown** change file owner
- **chgrp** change file group
- **umask** user file creation mode mask
- only owner or super-user can change file attributes
- upon creation, default permissions given to file modified by process **umask** value

Chmod command

- Symbolic access modes $\{a,u,g,o\} / \{r,w,x\}$
 - example: `chmod +r file`
- Octal access modes:

octal	read	write	execute
0	no	no	no
1	no	no	yes
2	no	yes	no
3	no	yes	yes
4	yes	no	no
5	yes	no	yes
6	yes	yes	no
7	yes	yes	yes

File System Internals

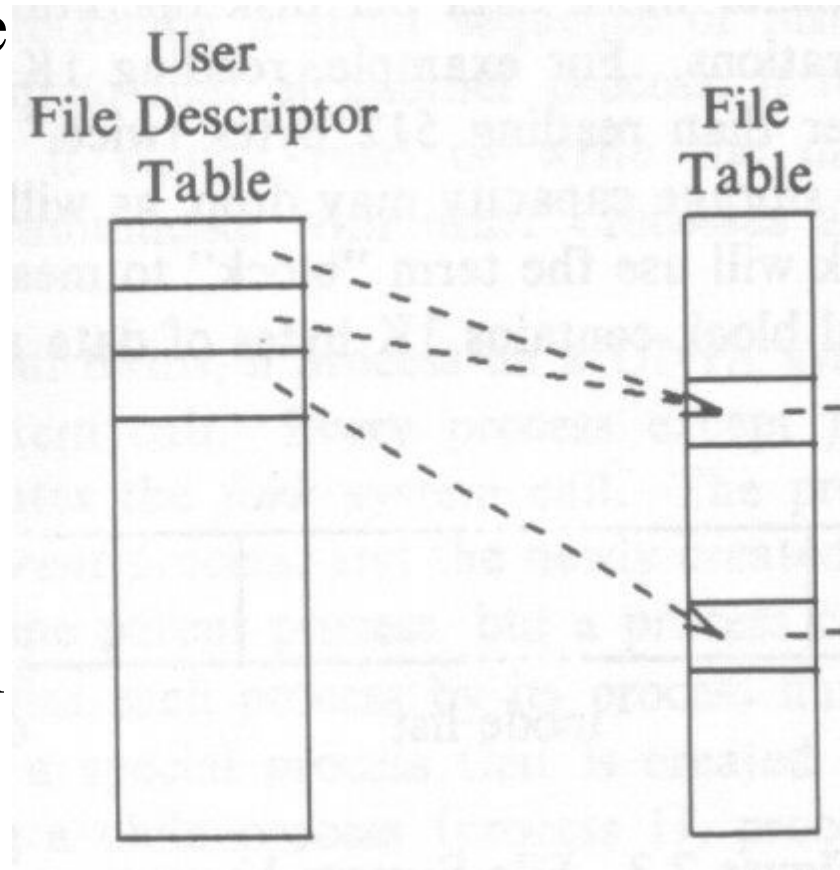
The Open File Table

- I/O operations are done on files by first **opening** them, reading/writing/etc., then **closing** them.
- The kernel maintains a global table containing information about each open file.

Inode	Mode	Count	Position
1023	read	1	0
1331	read/write	2	50
...			

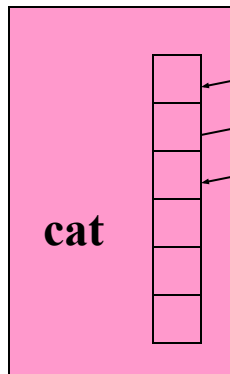
The File Descriptor Table

- Each process contains a table of files it has opened.
- Inherits open files from parent.
- Each open file is associated with a **number** or **handle**, called **file descriptor (fd)**.
- Each entry of this table points to an entry in the open file table.
- Always starts at 0.



Standard in/out/err

- The first three entries in the file descriptor table are special by convention:

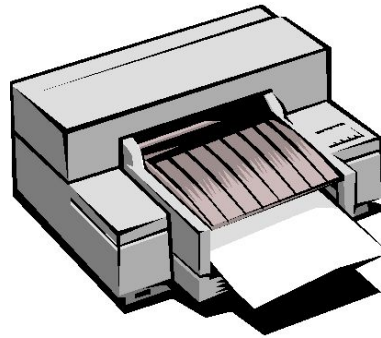


- Entry 0 is for input
- Entry 1 is for output
- Entry 2 is for error messages

- What about reading/writing to the screen?

Devices

- Besides files, input and output can go from/to various hardware devices.

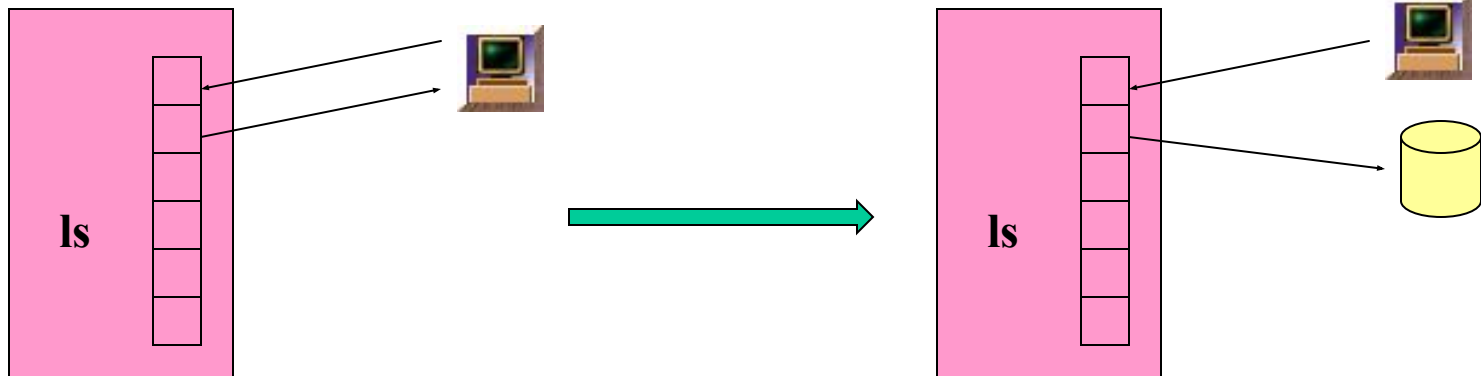


- **UNIX innovation:** Treat these just like files!
`/dev/tty`, `/dev/lpr`, `/dev/modem`
- By default, standard in/out/err opened with
`/dev/tty`
- Programs don't need to distinguish between files and devices

Redirection



- Before a command is executed, the input and output can be changed from the default (terminal) to a file
 - Shell modifies file descriptors in child process
 - The child program knows nothing about this!



Redirection of input/output

- Redirection of output: `>`
 - example: `$ ls > my_files`
- Redirection of input: `<`
 - example: `$ mail kornj <input.data`
- Append output: `>>`
 - example: `$ date >> logfile`
- Bourne Shell derivatives: *fd*`>`
 - example: `$ ls 2> error_log`

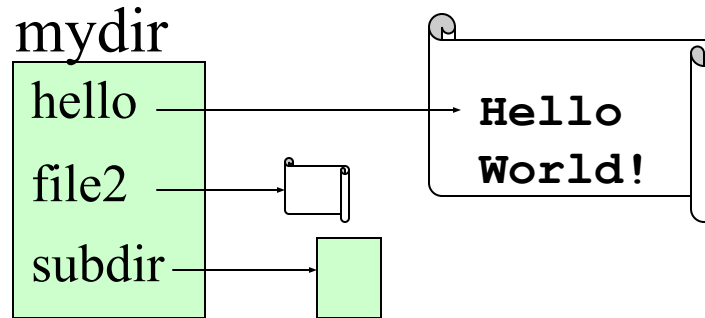
Using Devices

- Redirection works with devices (just like files)
- Special files in **/dev** directory
 - Example: **/dev/tty**
 - Example: **/dev/lp**
 - Example: **/dev/null**
 - `cat big_file > /dev/lp`
 - `cat big_file > /dev/null`

Links



- Directories are a list of files and directories.
 - Each directory entry *links* to a file on the disk

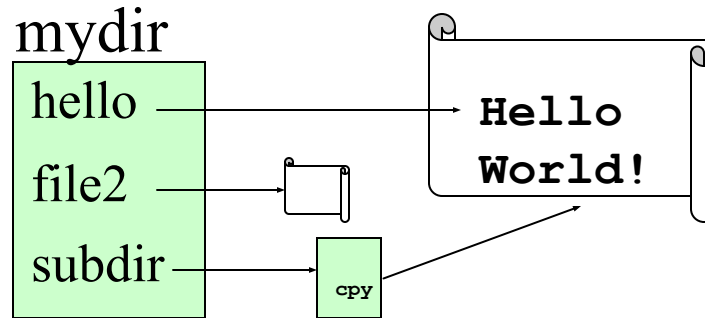


- Two different directory entries can link to the same file
 - In same directory or across different directories
 - Moving a file does not actually move any data around.
 - Creates link in new location
 - Deletes link in old location
- **ln** command

Links



- Directories are a list of files and directories.
 - Each directory entry *links* to a file on the disk

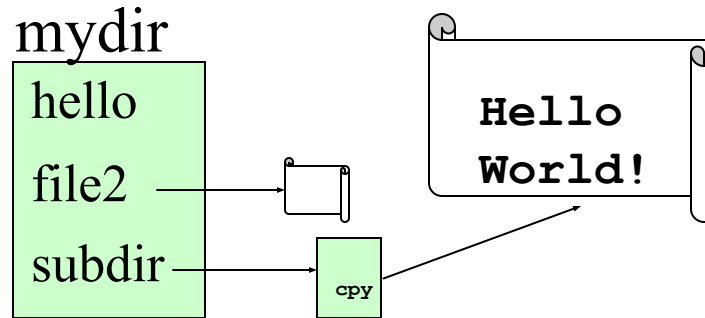


- Two different directory entries can link to the same file
 - In same directory or across different directories
 - Moving a file does not actually move any data around.
 - Creates link in new location
 - Deletes link in old location
- **ln** command

Links



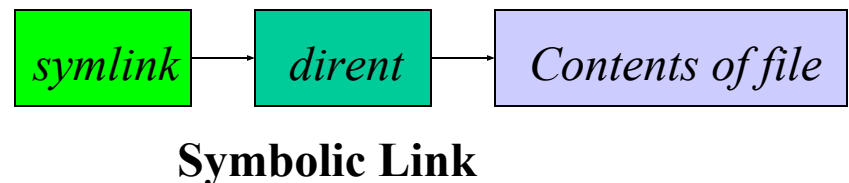
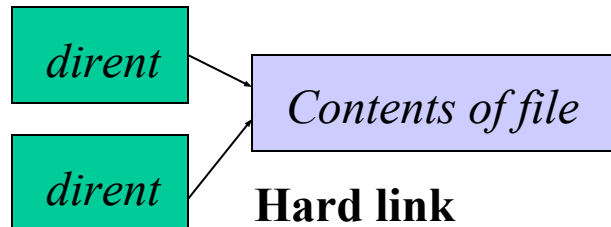
- Directories are a list of files and directories.
 - Each directory entry *links* to a file on the disk



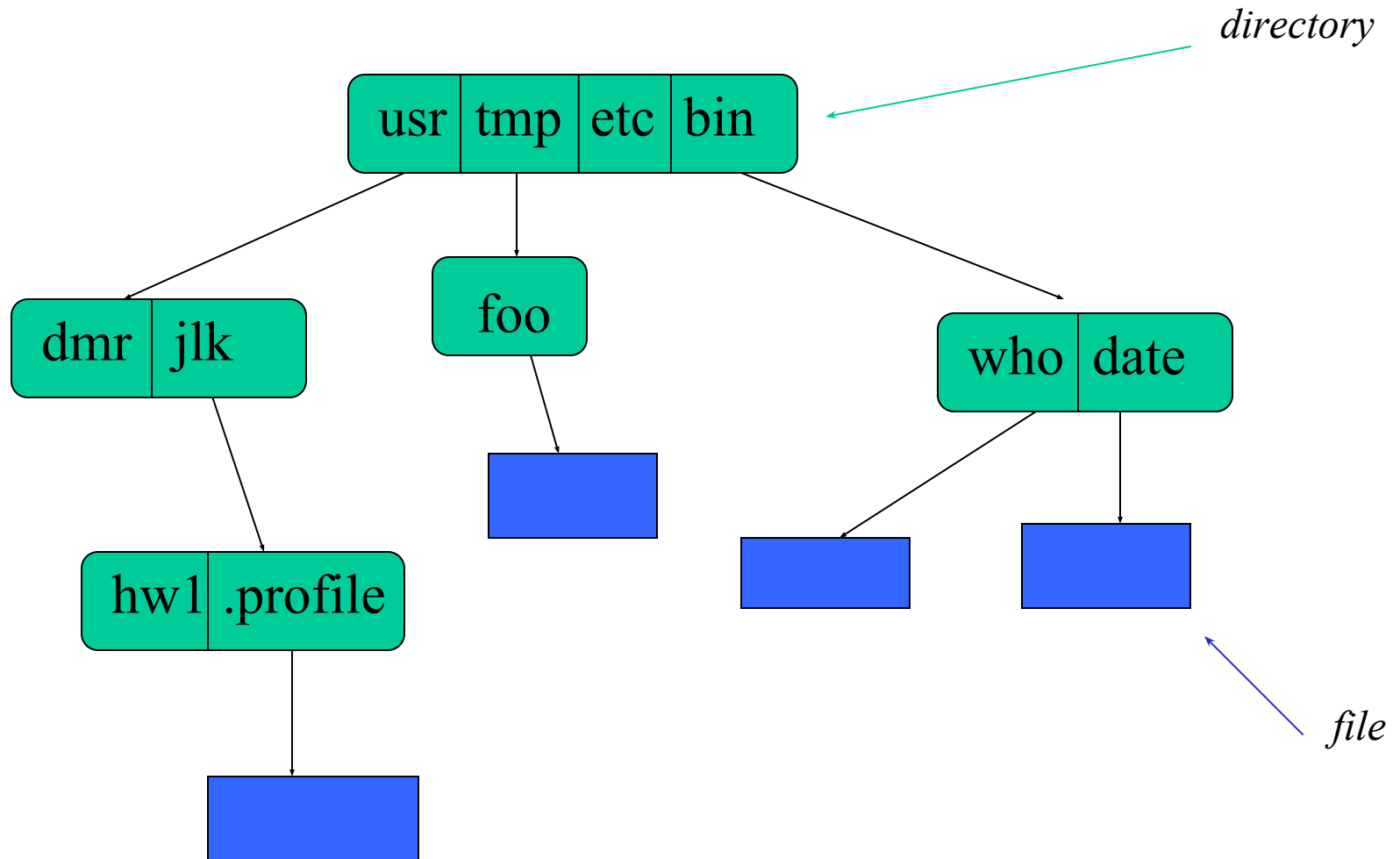
- Two different directory entries can link to the same file
 - In same directory or across different directories
 - Moving a file does not actually move any data around.
 - Creates link in new location
 - Deletes link in old location
- **ln** command

Symbolic links

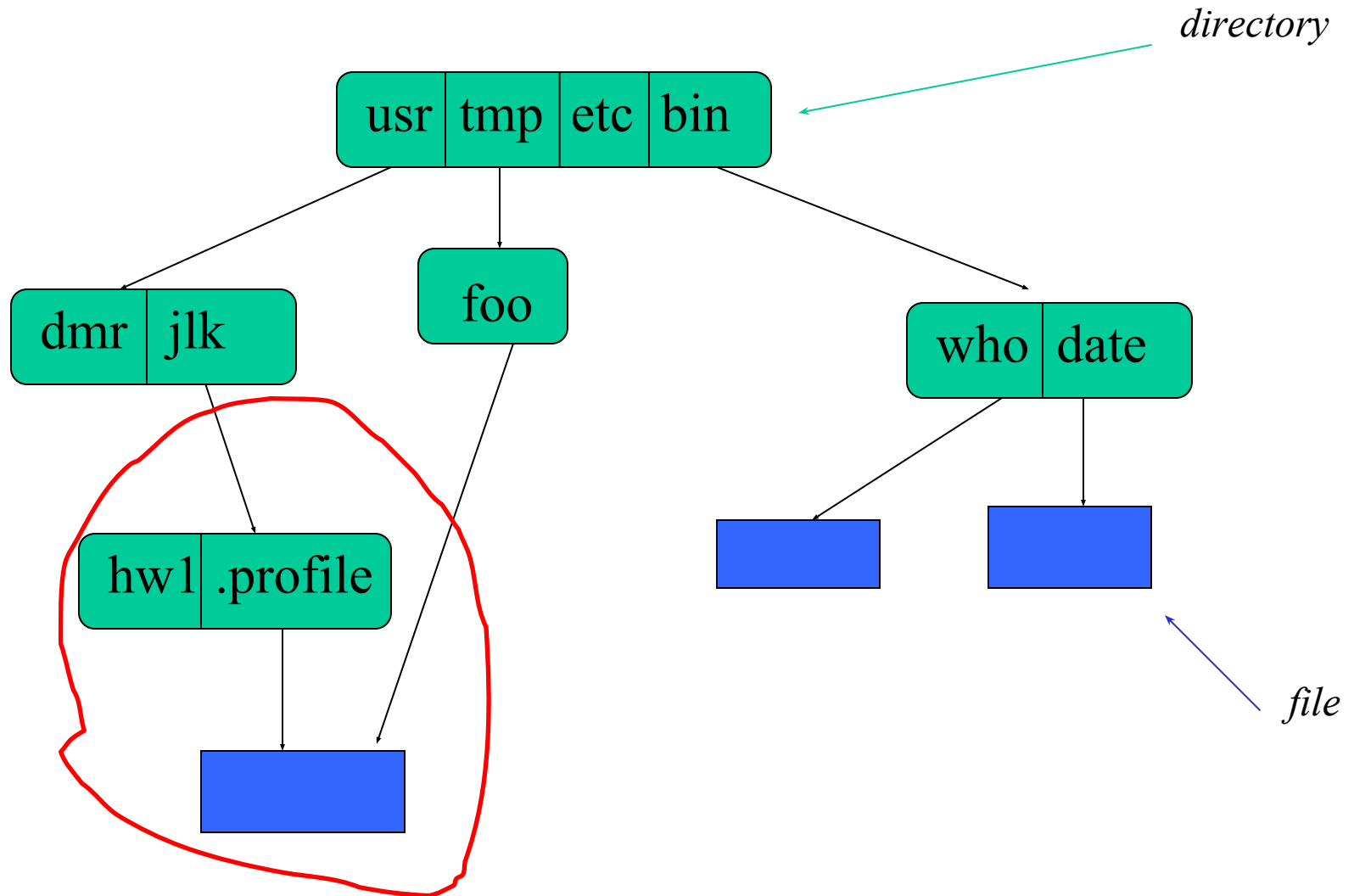
- **Symbolic** links are different than regular links (often called **hard links**). Created with **ln -s**
- Can be thought of as a directory entry that points to the *name* of another file.
- Does not change link count for file
 - When original deleted, symbolic link remains
- They exist because:
 - Hard links don't work across file systems
 - Hard links only work for regular files, not directories



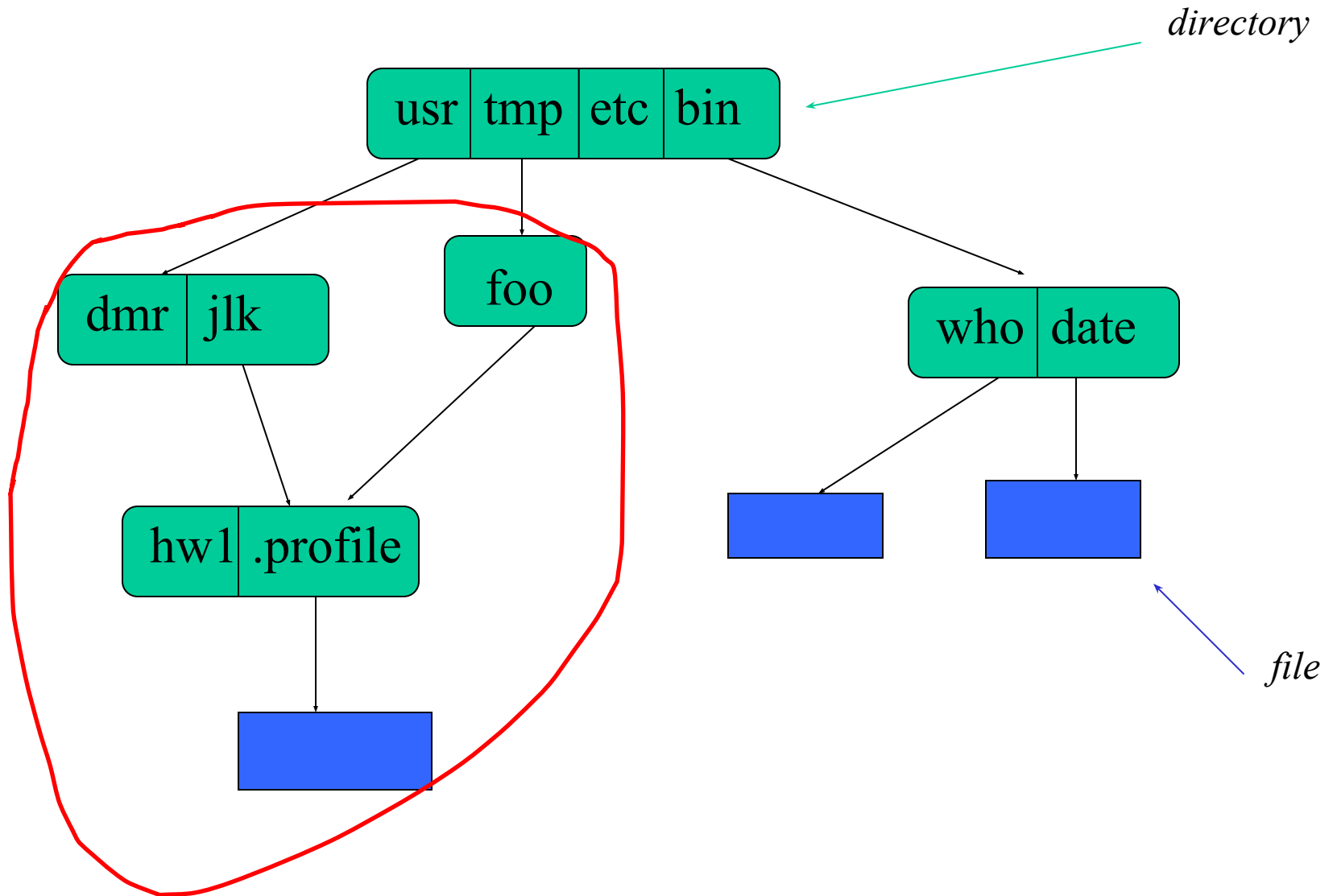
Example



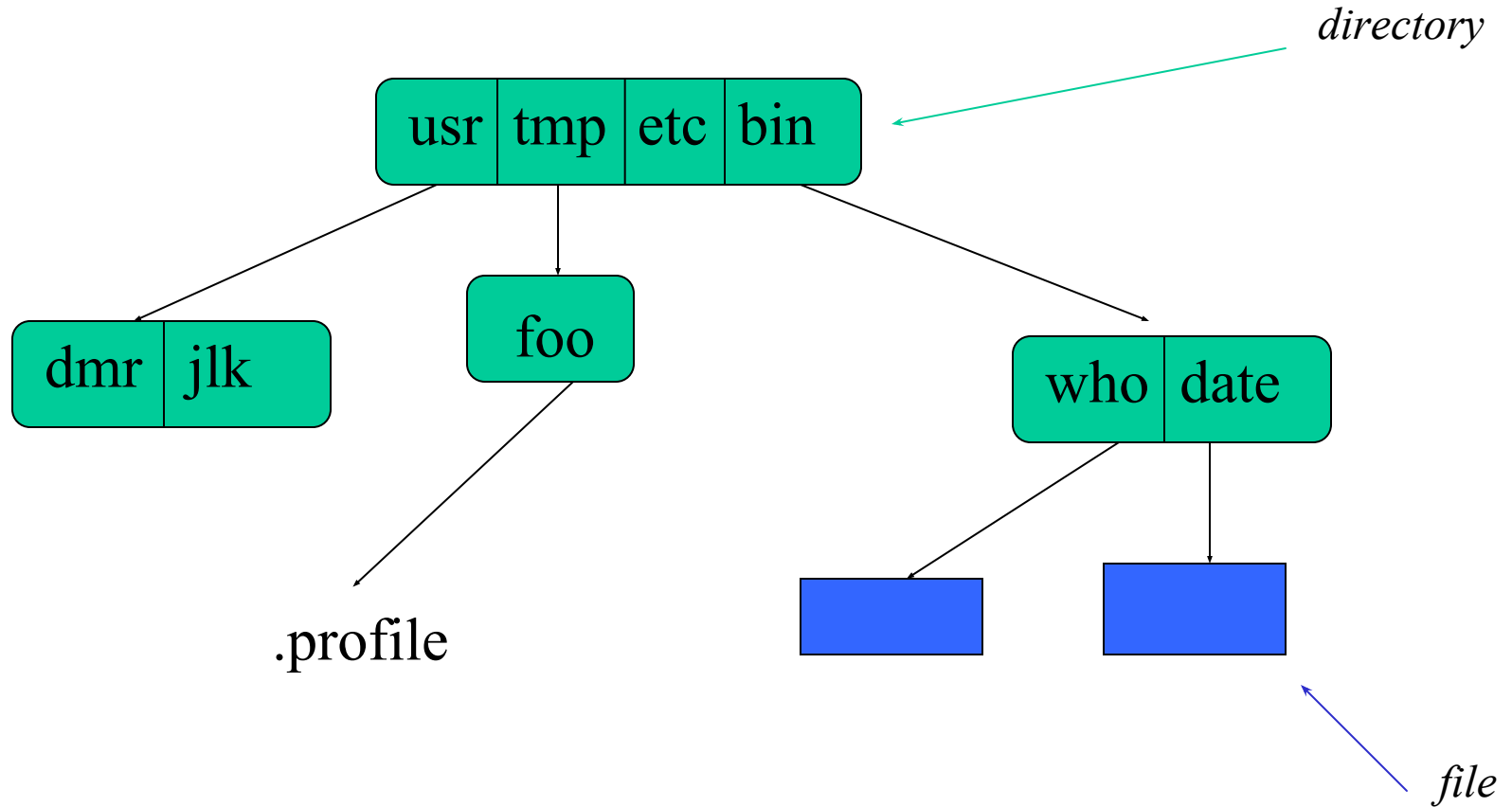
Hard Link



Symbolic Link

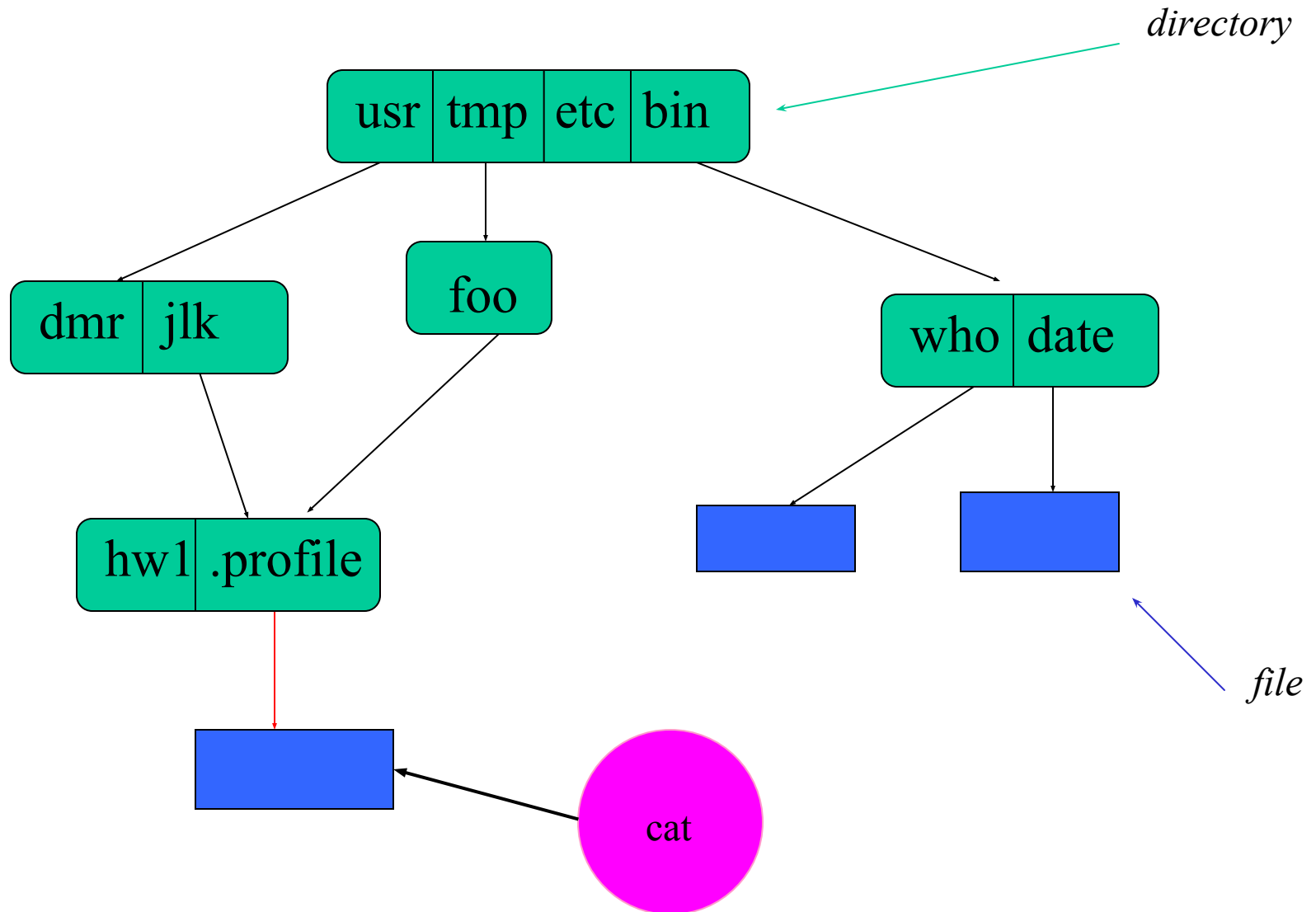


Symbolic Link



removal of original file can cause dangling link

Can a file have no links?



Tree Walking

- How can we find a *set* of files in the hierarchy?
- One possibility:
 - `ls -l -R /`
- What about:
 - All files below a given directory in the hierarchy?
 - All files since Jan 1, 2001?
 - All files larger than 10K?

find utility

- **find** *pathlist expression*
- **find** recursively descends through *pathlist* and applies *expression* to every file.
- *expression* can be:
 - name** *pattern*
true if file name matches pattern. Pattern may include shell patterns such as *, must be in quotes to suppress shell interpretation.

Eg: **find / -name '*.c'**

find utility (continued)

-perm *[+-]mode*

Find files with given access mode, mode must be in octal. Eg: **find**

. -perm 755

-type *ch*

Find files of type *ch* (*c*=character, *b*=block, *f* for plain file, etc..). Eg:

find /home -type f

-user *userid/username*

Find by owner *userid* or *username*

-group *groupid/groupname*

Find by group *groupid* or *groupname*

-size *size*

File size is at least *size*

many more...

find: logical operations

! <i>expression</i>	returns the logical negation of expression
<i>op1</i> -a <i>op2</i>	matches both patterns <i>op1</i> and <i>op2</i>
<i>op1</i> -o <i>op2</i>	matches either <i>op1</i> or <i>op2</i>
()	group expressions together

find: actions

- print** prints out the name of the current file (default)
- prune** skips the named directory and below
- exec *cmd***
 - Executes *cmd*, where *cmd* must be terminated by an escaped semicolon (`\;` or `' ; '`).
 - If you specify `{ }` as a command line argument, it is replaced by the name of the current file just found.
 - **exec** executes *cmd* once per file.
 - Example:

```
find -name "*.o" -exec rm "{}" ";"
```

there is an easier alternative to exec we will discuss later!

find Examples

- Find all files beneath home directory beginning with f
`find ~ -name 'f*' -print`
- Find all files beneath home directory modified in last day
`find ~ -mtime 1 -print`
- Find all files beneath home directory larger than 10K
`find ~ -size 10k -print`
- Count words in files under home directory
`find ~ -exec wc -w {} \; -print`
- Remove core files
`find / -name core -exec rm {} \;`

diff: comparing two files

- **diff**: compares two files and outputs a description of their differences

Usage: **diff** [*options*] *file1 file2*

-i: ignore case

```
apples
oranges
walnuts
```

```
apples
oranges
grapes
```

```
$ diff test1 test2
3c3
< walnuts
---
> grapes
```

Other file comparison utilities

- **cmp**
 - Tests two files for equality
 - If equal, nothing returned. If different, location of first differing byte returned
 - Faster than **diff** for checking equality
- **comm**
 - Reads two files and outputs three columns:
 - Lines in first file only
 - Lines in second file only
 - Lines in both files
 - Files must be sorted
 - Options: fields to suppress (`[-123]`)

Next Time

- Processes
- **Pipes**
- Filters