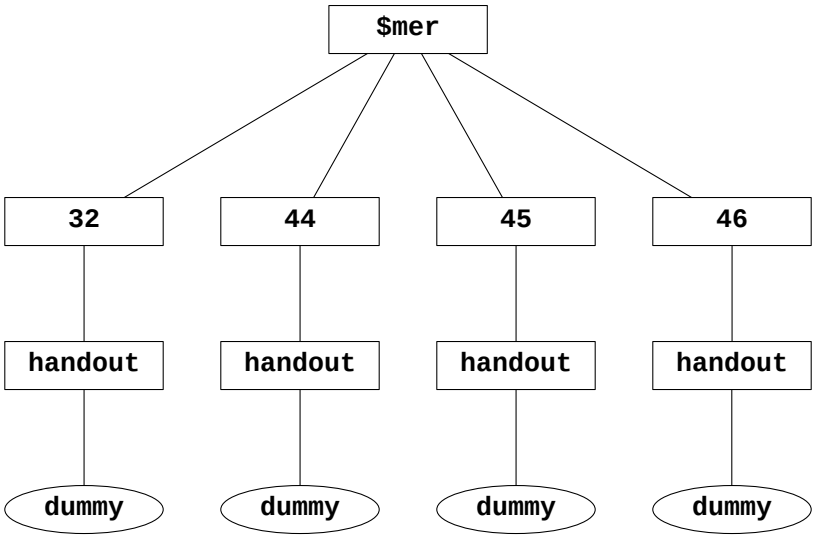
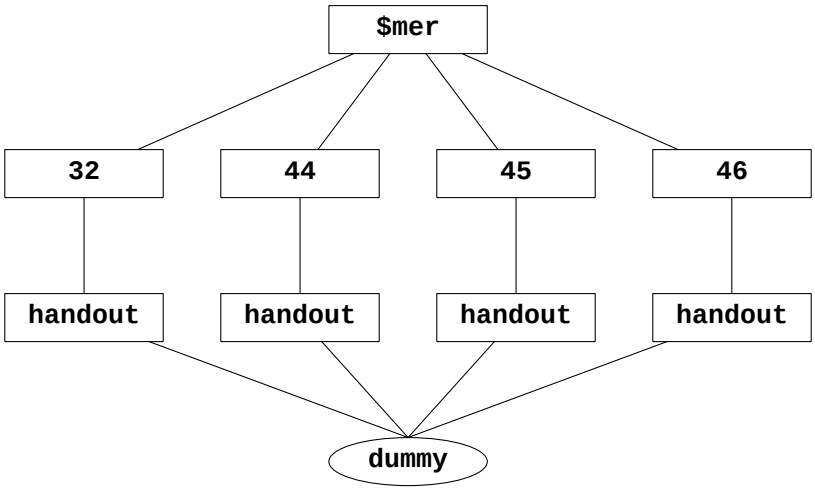
Summer 2004 Handout 4

A file with links to two or more directories: KP pp. 59−60




```
1$ ls -l \
    $mer/32/handout/dummy \
    $mer/44/handout/dummy \
    $mer/45/handout/dummy \
    $mer/46/handout/dummy

2$ cd $mer
3$ ls -l \
    32/handout/dummy \
    44/handout/dummy \
    45/handout/dummy \
    46/handout/dummy

-r--r--r--   1 mm64     users         29 Jun 23 10:53 32/handout/dummy
-r--r--r--   1 mm64     users         29 Jan  9 00:36 44/handout/dummy
-r--r--r--   1 mm64     users         29 Jan  9 00:36 45/handout/dummy
-r--r--r--   1 mm64     users         29 Jan  9 00:36 46/handout/dummy

4$ cd $mer
5$ ls -li \
    32/handout/dummy \
    44/handout/dummy \
    45/handout/dummy \
    46/handout/dummy
```

```
640324 -r--r--r--   1 mm64     users          29 Jun 23 10:53 32/handout/dummy
641441 -r--r--r--   1 mm64     users          29 Jan  9 00:36 44/handout/dummy
639191 -r--r--r--   1 mm64     users          29 Jan  9 00:36 45/handout/dummy
639398 -r--r--r--   1 mm64     users          29 Jan  9 00:36 46/handout/dummy
```



A *link* is the thing that connects a file to the directory that contains it. When you **mv** a file from directory to directory, you're not moving the file around the disk: you're merely removing one link and creating another.

cp, **mv**, and **ln** take the same command line arguments:

```
6$ cp existing new
7$ mv existing new
8$ ln existing new
```

ln is just like **mv**, except that it doesn't remove the old link. I created the above illusion of four identical files by

```
9$ cd $mer/32/handout
10$ vi dummy                                  I created the file dummy.

11$ ln dummy $mer/44/handout
12$ ln dummy $mer/45/handout
13$ ln dummy $mer/46/handout
14$ chmod 444 dummy                           changes all four files
```

There is no longer any way to tell which of the four files was the original: all are equally authentic.

▼ Homework 4.1: create and rm a file with links to two or more directories (not to be handed in)

Create and **rm** a file with links to two or more directories. When you create a new link, does the link count output by **ls -l** increase by one? When you **chmod** any one of the ''copies'', do the mode bits of the others automatically change? When you edit any one of the ''copies'', do the date, size, and contents of the others automatically change? When you **rm** a link, does the link count output by **ls -l** decrease by one? Do the links to the other directories survive?

▲

▼ Homework 4.2: display a file on the World Wide Web

A executable program must be in your **$HOME/bin** directory. A file that you display on the Web must be in your **$HOME/public_html** directory or one of its descendants. To display an executable file on the Web while keeping it executable, link it to both directories:

```
1$ cd
2$ cd bin
3$ pwd
4$ chmod a+r myscript                    Turn on all three r bits.
5$ ln myscript $HOME/public_html

6$ cd
7$ cd public_html
8$ pwd
9$ ls -l | more
```

Now point your browser at

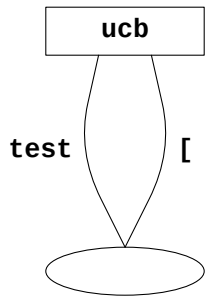
```
http://i5.nyu.edu/~abc1234/myscript
```

▲

A file with two or more different names

```
#!/bin/sh

if [ $# -ne 3 ]
then
    echo $0: requires 3 command line arguments 1>&2
    exit 1
fi

if test $# -ne 3                                        #KP p. 140
then
    echo $0: requires 3 command line arguments 1>&2
    exit 1
fi

exit 0
```

```
1$ cd /usr/ucb
2$ ls -l | more
-rwxr-xr-x   1 root     bin         7556 Apr  6  2002 test
```



See **man chmod** for the "sticky bit" **t**:

```
3$ man chmod              and of course press RETURN
/sticky                   and press RETURN to search for the word sticky
control-U                 scroll up to see the whole paragraph

4$ cd /usr/ucb
5$ ls -li | more
      8293 lrwxrwxrwx   1 root     root           9 Aug 18  2003 vi -> ../bin/vi

6$ cd /bin
7$ ls -li | more
       264 -r-xr-xr-x   1 root     bin        10232 Apr  6  2002 grep
```

To find all examples like the above in the **/usr/ucb** directory,

```
8$ cd /usr/ucb
9$ ls -lai | awk 'NR >= 2 && $2 ~ /^-/ && $3 > 1' | sort +0n +9 | head -11
      1408 -r-xr-xr-x  42 root     bin         5424 Jan  6  2003 ps

10$ cd $mer/bin
11$ ls -li | more
    640016 -r-xr-xr-x   1 mm64     users        699 Oct 26  1998 fax
    640038 -r-xr-xr-x   1 mm64     users        699 Oct 26  1998 postfax
```

The name of a file that you see with **ls -l** is not stored in the file itself: it's stored in the link. A file with two links can have two different names. Here is how the superuser could have created **test** and **[**:

```
12$ cd /usr/bin
13$ vi test                          Create the test program—assume it's a shellscript.

14$ ln test '['                      just like mv or cp
15$ chmod 755 test
```

A file with multiple links can have more than one name and be linked to more than one directory. Of course, a file can't have two links with the same name to the same directory.

Create a link to the newest version of a file

Suppose you keep creating new versions of a program, each in a file with a different name:

```
/home1/a/abc1234/prog1.c
/home1/a/abc1234/prog2.c
/home1/a/abc1234/prog3.c
```

Each time you create a new version, it would seem that you have to change all the software and documentation that mentions the name of the file that holds the newest version:

```
#!/bin/sh
#Compile the most recent version of the program.

gcc /home1/a/abc1234/prog3.c
```

To make your software and documentation require less maintenance, give the newest file an additional name with a link:

```
3$ cd /home1/a/abc1234
4$ pwd
```

```
5$ ln prog3.c prog.c
6$ ls -l
-rw-r--r--   1 abc1234  users        100 Oct 19 16:24 prog1.c
-rw-r--r--   1 abc1234  users        200 Oct 19 16:25 prog2.c
-rw-r--r--   2 abc1234  users        300 Oct 19 16:26 prog3.c
-rw-r--r--   2 abc1234  users        300 Oct 19 16:26 prog.c
```

```
#!/bin/sh
#Compile the most recent version of the program.

gcc /home1/a/abc1234/prog.c
```

When **prog4.c** is created, your software and documentation can remain unchanged. Simply

```
7$ cd /home1/a/abc1234
8$ pwd

9$ ls -l prog.c                 first make sure that prog.c has more than one link
10$ rm prog.c                   does not remove prog3.c
11$ ln prog4.c prog.c
12$ ls -l
```

▼ Homework 4.3: create a file with two different names (not to be handed in)

Create a file with two links to the same directory, with a different name in each link (exactly like **test** and **[**, or **prog3.c** and **prog.c**).

▲

Symbolic links

The tree of directories is not stored all on one disk. It's divided into filesystems, each of which may be stored on a different disk or disk partition.

```
1$ df
/                  (/dev/md/dsk/d0    ): 3944106 blocks   338598 files
/usr               (/dev/md/dsk/d6    ):11133212 blocks   940460 files
/proc              (/proc             ):       0 blocks    29933 files
/etc/mnttab        (mnttab            ):       0 blocks        0 files
/dev/fd            (fd                ):       0 blocks        0 files
/var               (/dev/md/dsk/d1    ): 5324398 blocks   502759 files
/var/run           (swap              ):21530800 blocks   800906 files
/tmp               (swap              ):21530800 blocks   800906 files
/opt               (/dev/md/dsk/d5    ): 1222460 blocks   271416 files
/home1             (/dev/md/dsk/d8    ):34714786 blocks  3302616 files
/local             (/dev/md/dsk/d7    ):25891702 blocks  1766207 files
```

You're not allowed to link a file to two directories in two different filesystems:

```
2$ cd
3$ pwd
/home1/a/abc1234

4$ date > junk
5$ ls -l junk
-rw-------   1 abc1234  users         29 Jun 23 10:53 junk
```

```
6$ ln junk /tmp
ln: junk and /tmp/junk are located on different file systems.

7$ df junk                                  See which filesystem contains a given file.
/home1             (/dev/md/dsk/d8    ):34714786 blocks  3302616 files

8$ df /tmp                                 See which filesystem contains a given directory.
/tmp               (swap              ):21531168 blocks   800905 files

9$ cd /tmp
10$ pwd
/tmp

11$ cd /
12$ ls -ld /tmp
drwxrwxrwt   4 root     sys         2901 Jun 23 10:53 /tmp
```

You can create another type of link which is not subject to this limitation. The new type of link is called a *symbolic* link; the old type is called a *hard* link. Specify the full pathname of the existing file when creating a symbolic link:

```
13$ cd
14$ cat junk
Wed Jun 23 10:53:48 EDT 2004

15$ ln -s /home1/a/abc1234/junk /tmp
16$ ln -s `pwd`/junk /tmp                  easier way to do the same thing

17$ cd /tmp
18$ cat junk
Wed Jun 23 10:53:48 EDT 2004

19$ cd /tmp
20$ ls -l junk
lrwxrwxrwx   1 abc1234  users          20 Jun 23 10:53 junk -> /home1/a/abc1234/junk
```

Every file is born with exactly one hard link. If it is later given additional hard links, it's impossible to tell which is the original. But a symbolic link is easy to recognize: **`ls -l`** displays it with an arrow. Why did the **`ln -s`** command create a new file containing exactly 24 characters?

Examples of symbolic links

```
1$ cd /usr/local/bin
2$ ls -l cc
lrwxrwxrwx   1 root     other         16 Feb  5 12:05 cc -> /opt/sfw/bin/gcc

3$ cd /bin
4$ ls -l perl
lrwxrwxrwx   1 root     root          23 Aug 18  2003 perl -> ../perl5/5.6.1/bin/perl
```

Get the right awk automatically

`/usr/local/bin/gnuawk` is better than our other **`awk`**'s:

```
1$ cd /usr/bin
2$ ls -l *awk*
-r-xr-xr-x   2 root     bin        85296 Feb 19 17:37 awk
-r-xr-xr-x   1 root     bin       119804 Feb 19 17:37 nawk
-r-xr-xr-x   2 root     bin        85296 Feb 19 17:37 oawk
```

You have to use a symbolic link because a hard link can't reach far enough:

```
3$ cd
4$ cd bin
5$ ln -s /usr/local/bin/gnuawk awk
6$ ls -l awk
```

Symbolic links between directories

A link is also used to connect each directory with its parent directory. When you create a directory with **mkdir**, you automatically give it a hard link to its parent. Only the superuser is allowed to create additional hard links from the directory to a parent. But anyone can create a symbolic link from the directory to a parent.

Suppose you keep creating new versions of a software project, each in a different directory:

```
/home1/a/abc1234/project/ver-1.0
/home1/a/abc1234/project/ver-1.1
/home1/a/abc1234/project/ver-1.2
```

Each time you create a new version, it would seem that you have to change all the software and documentation that mentions the name of the directory that holds the newest version:

```
#!/bin/sh
#List all the files in the most recent version of the project.

ls -l /home1/a/abc1234/project/ver-1.2
```

To make your software and documentation require less maintenance, give the newest directory an additional name with a symbolic link:

```
1$ cd
2$ cd /home1/a/abc1234/project
3$ pwd

4$ ln -s /home1/a/abc1234/project/ver-1.2 ver-newest

5$ ls -l
drwxr-xr-x   2 abc1234  users        512 Oct 19 16:24 ver-1.0
drwxr-xr-x   2 abc1234  users        512 Oct 19 16:25 ver-1.1
drwxr-xr-x   2 abc1234  users        512 Oct 19 16:26 ver-1.2
lrwxr-xr-x   1 abc1234  users         53 Oct 19 16:27 ver-newest ->
                                        /home1/a/abc1234/project/ver-1.2
```

```
#!/bin/sh
#List all the files in the most recent version of the project.

ls -l /home1/a/abc1234/project/ver-newest
```

When version 1.3 is created, your software and documentation can remain unchanged. Simply

```
6$ cd
7$ cd project
8$ pwd

9$ rm ver-newest                                    does not remove ver-1.2
10$ ln -s /home1/a/abc1234/project/ver-1.3 ver-newest
11$ ls -l
```

Shell abbreviations for names of files and directories: pp. 26–29

```
1$ rm *
2$ rm *core*
3$ rm *.c

4$ rm handout?.ms
5$ rm ??

6$ rm handout[12345].ms
7$ rm handout[1-5].ms
8$ rm [a-z][a-z][a-z][0-9][0-9][0-9][0-9][0-9]
```

shell language filename abbreviation	*regular expression*
`*`	`.*`
`?`	`.`
`.`	`\.`
`'*'`	`\*`
`[abc]`	`[abc]`
`[a-z]`	`[a-z]`
`[!a-z]` *in* **ksh** *and* **bash**	`[^a-z]`

Command line arguments for find: see `find`(1)

```
1$ find ~ -name core -print          Put this line in your .profile file; -print optional.
2$ find ~ -name '*.c' -print         Need quotes; can also use ? [ ]
3$ find ~ -name '*.c' -ls            minus lowercase LS: just like ls -l
4$ find ~ -type d -print             d for directory, f for file
```

Two consecutive conditions (such as **`-type d`** and **`-name handout`**) are assumed to have an implicit ''and'' between them. If you want ''or'', write an explicit **`-o`** between them.

```
5$ find ~ -type d -a -name '*bin*' -print      directories whose name contains bin
6$ find ~ -type d    -name '*bin*' -print      -a is optional, -o is mandatory
```

There are also parentheses, which must both be in **'**single quotes**'**, and **`!`** for ''not''. Surprisingly, **`!`** is not a special character in the Korn and Bourne shells, and is not a special character when followed by a blank in the C shell.

```
7$ find ~   -user abc1234 -print          files & directories owned by abc1234
8$ find ~ ! -user abc1234 -print          files & directories not owned by abc1234
```

```
9$ find ~   -perm  644 -print
10$ find ~   -perm -644 -print
11$ find ~ ! -perm -644 -print
```
files and directories with **rw-r--r--**
files and directories with at least **rw-r--r--**
files and directories with less than **rw-r--r--**

```
12$ find ~/public_html -type f -a ! -perm -444 -print
13$ find ~/public_html -type d -a ! -perm -555 -print
14$ find ~/public_html -type f -a ! -perm -444 -o -type d -a ! -perm -555 -print
```

```
15$ find ~ -type f -a -size  1000c -print
16$ find ~ -type f -a -size +1000c -print
17$ find ~ -type f -a -size -1000c -print
```
files whose size is exactly 1000 bytes ("characters")
files whose size is greater than 1000 bytes
files whose size is less than 1000 bytes

What's the simplest way to **find** all files whose size is greater than or equal to 10 bytes?

```
18$ find ~ -type f -a -atime  10 -print
19$ find ~ -type f -a -atime +10 -print
20$ find ~ -type f -a -atime -10 -print
```
files last accessed 10 days ago: **ls -lu**
files last accessed more than 10 days ago
files last accessed less than 10 days ago:

```
21$ find ~ -type f -a -mtime +10 -print
22$ find ~ -type f -a -ctime +10 -print
```
files last modified more than 10 days ago: **ls -lt**
files last changed more than 10 days ago: **ls -lc**

"Accessed" means that the file was input into a program. "Modified" means that a program's output was deposited into the file, i.e., that the contents of the file were changed. "Changed" means that the contents of the file were changed, or the nine permission bits were changed (with **chmod**), or the owner of the file was changed (with **chown**), etc.

```
23$ date > junk1
24$ date > junk2
25$ date > junk3
```
junk3 *was most recently modified.*

```
26$ chmod 400 junk2
```
junk2 *was most recently changed.*

```
27$ cat junk3
28$ cat junk1
```
junk1 *was most recently accessed.*

```
29$ ls -l junk[123]                                   alphabetical, showing modification times
-rw-------   1 abc1234  users        29 Jun 23 10:53 junk1
-r--------   1 abc1234  users        29 Jun 23 10:54 junk2
-rw-------   1 abc1234  users        29 Jun 23 10:55 junk3

30$ ls -ltu junk[123]                                                          accessed
-rw-------   1 abc1234  users        29 Jun 23 10:58 junk1
-rw-------   1 abc1234  users        29 Jun 23 10:57 junk3
-r--------   1 abc1234  users        29 Jun 23 10:54 junk2

31$ ls -lt junk[123]                                                           modified
-rw-------   1 abc1234  users        29 Jun 23 10:55 junk3
-r--------   1 abc1234  users        29 Jun 23 10:54 junk2
-rw-------   1 abc1234  users        29 Jun 23 10:53 junk1

32$ ls -ltc junk[123]                                                          changed
-r--------   1 abc1234  users        29 Jun 23 10:56 junk2
-rw-------   1 abc1234  users        29 Jun 23 10:55 junk3
-rw-------   1 abc1234  users        29 Jun 23 10:53 junk1
```

Search the entire tree of directories

find can only search directories for which you have both **r** and **x** permission. Use **2>** (textbook p. 93; **ksh**(1) p. 17) **/dev/null** (pp. 68−69) to throw away the error message that **find** issues when it is rebuffed by a directory for which you do not have both permissions.

To use your terminal for something else while **find** is running, run **find** in the background with an ampersand (textbook p. 33; **ksh**(1) p. 1).

```
1$ find / -type f -user abc1234 -print > ~/find.out 2> /dev/null &
2$                                        The prompt reappears immediately.
```

Lightning review of back quotes

```
1$      ls -t | lpr                  Print the names of the files in chronological order.
2$ lpr ‘ls -t‘                       Print the contents of the files in chronological order.
```

Typical find commands

```
1$ find / -type f -a -user abc1234 -print          Output the names of files belonging to abc1234.
2$ find / -type f -a -user abc1234 -print | lpr    Print the names of files belonging to abc1234.

3$ lpr   ‘find / -type f -a -user abc1234 -print‘  Print the contents of files belonging to abc1234.
4$ rm    ‘find / -type f -a -user abc1234 -print‘  Remove the files belonging to abc1234.
5$ rmdir ‘find / -type d -a -user abc1234 -print‘  Remove the directories belonging to abc1234.
```

```
#!/bin/ksh
#A separate lpr command for each file will do a page eject before
#each file.

for filename in ‘find / -type f -a -user abc1234 -print‘
do
    lpr $filename
done
```

The following command does the same thing as the above loop:

```
6$ find / -type f -a -user abc1234 -exec lpr {} ’;’
```

Find all hard links to a file

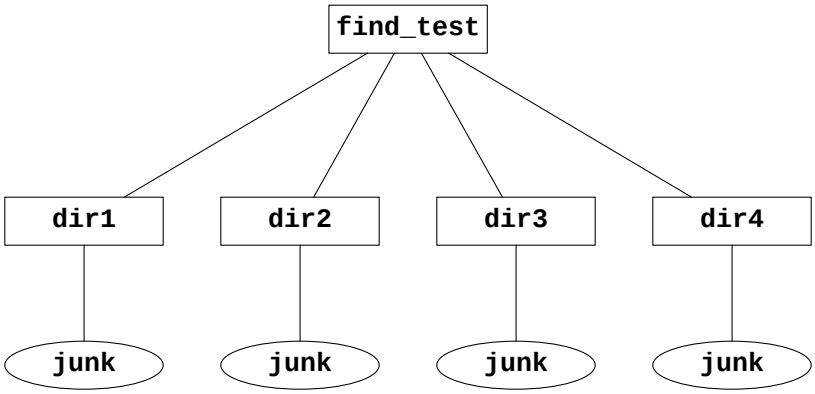
A file can have many names, but it can have only one inode number. This number never changes. To find all the hard links to a file,

```
1$ cd /etc/init.d
2$ pwd

3$ ls -li inetinit
5830 -rwxr--r--   5 root     sys        12655 Jan 13 12:19 inetinit

4$ ls -li ‘find /etc -inum 5830‘
5830 -rwxr--r--   5 root     sys        12655 Jan 13 12:19 /etc/init.d/inetinit
5830 -rwxr--r--   5 root     sys        12655 Jan 13 12:19 /etc/rc0.d/K43inet
5830 -rwxr--r--   5 root     sys        12655 Jan 13 12:19 /etc/rc1.d/K43inet
5830 -rwxr--r--   5 root     sys        12655 Jan 13 12:19 /etc/rc2.d/S69inet
5830 -rwxr--r--   5 root     sys        12655 Jan 13 12:19 /etc/rcS.d/K43inet
```

See **init**(1M) for the eight runlevels in our version of Unix.

Avoid error messages with "minus prune"

find outputs an error message unless the person running **find** has both **r** and **x** permission in a directory:




```
1$ cd
2$ mkdir find_test
3$ cd find_test
4$ mkdir dir1 dir2 dir3 dir4

5$ date > dir1/junk
6$ date > dir2/junk
7$ date > dir3/junk
8$ date > dir4/junk

9$ chmod 000 dir1
10$ chmod 100 dir2
11$ chmod 400 dir3
12$ chmod 500 dir4

13$ ls -l
d---------   2 abc1234  users        183 Jun 23 10:58 dir1
d--x------   2 abc1234  users        183 Jun 23 10:58 dir2
dr--------   2 abc1234  users        183 Jun 23 10:58 dir3
dr-x------   2 abc1234  users        183 Jun 23 10:58 dir4

14$ find . -type f -name junk -print
find: cannot read dir dir1: Permission denied
find: cannot read dir dir2: Permission denied
find: cannot read dir dir3/: Permission denied
dir4/junk
```

The **-prune** argument tells **find** not to attempt to visit the files and subdirectories that a directory contains. I wish we could write an **if** statement to steer **find** away from directories that do not have at least **r-xr-xr-x**, thus avoiding the above error messages:

```
if (-type d -a ! -perm -555) {
    -prune
}
```

The **-o** argument of **find** short circuits like the **||** in the language C:

```
if (a == b || c == d || e == f) {
```

The following eleven additional arguments therefore do the job of the above **if**:

```
#!/bin/ksh

find . \
    '(' ! -type d -o -perm -555 -o -prune ')' -a \
    -type f -a -name junk -print
```

▼ Homework 4.4: find things

(1) **find** every file named **stdio.h**.

(2) **find** every file named **iostream** or **iostream.h**.

(3) **find** every directory named **font**.

(4) **find** every file named **httpd.conf**. See Handout 3, pp. 6−7.

(5) **find** every file named **httpd.conf** except in **/home1** and its subdirectories. Use **-prune**.

(6) **find** every directory named **man** except in **/home1** and its subdirectories. Use **-prune**.

(7) **find** every directory descended from **/usr/include**. How many **.h** files are there in **/usr/include** and its descendants?

(8) How many directories are you allow to visit on i5.nyu.edu? How many directories are there at each level? How many files are there?

(9) What is the inode number (X52.9545 Handout 2, p. 6) of the file **/etc/init.d/inetsvc**? **ls -l** every hard link to that file.

▲

□