Many tools used by the system administrator look at these time stamps to decide whether a particular file will participate in a backup or not. A file is often incorrectly stamped when it is extracted from a backup with **tar** or **cpio**. Section 19.3.1 discusses how the **touch** command is used to rectify such situations.


Note

It's possible to change the access time of a file without changing its modification time. In an inverse manner, when you modify a file, you generally change its access time as well. However, on some systems, when you redirect output (with the > and >> symbols), you change the contents but not the last access time.


Tip

What happens when you copy a file with **cp**? By default, the copy has the modification and access time stamps set to the time of copying. Sometimes, you may not like this to happen. In that case, use **cp -p** (preserve) to retain both time stamps.

4.11 find: Locating Files

find is one of the power tools of the UNIX system. It *recursively* examines a directory tree to look for files matching some criteria and then takes some action on the selected files. It has a difficult command line, and if you have ever wondered why UNIX is hated by many, then you should look up the cryptic **find** documentation. However, **find** is easily tamed if you break up its arguments into three components:

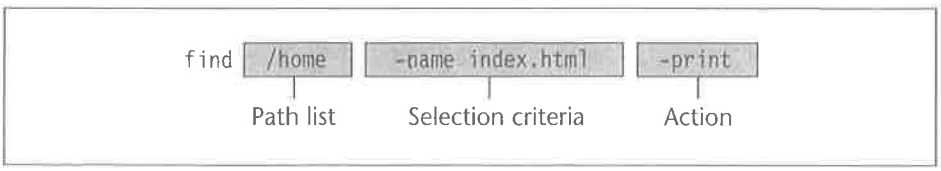
`find` *path_list selection_criteria action*

Fig. 4.6 shows the structure of a typical **find** command. The command completely examines a directory tree in this way:

- First, it recursively examines all files in the directories specified in *path_list*. Here, it begins the search from `/home`.
- It then matches each file for one or more *selection_criteria*. This always consists of an expression in the form *-operator argument* (`-name index.html`). Here, **find** selects the file if it has the name `index.html`.
- Finally, it takes some action on those selected files. The action `-print` simply displays the **find** output on the terminal.

All **find** operators (also referred to as *options* in this text) begin with a hyphen. You can provide one or more subdirectories as the *path_list* and multiple *selection_criteria* to match one or more files. This makes the command difficult to use initially, but it is a program that every user must master since it lets her select files under practically any condition.

FIGURE 4.6 *Structure of a* **find** *command*



As our first example, let's use **find** to locate all files named `a.out` (the executab file generated by the C compiler):

```
$ find / -name a.out -print
/home/romeo/scripts/a.out
/home/andrew/scripts/reports/a.out
/home/juliet/a.out
```

Since the search starts from the root directory, **find** displays absolute pathnames. Yo can also use relative names in the path list, and **find** will then output a list of relativ pathnames. Moreover, when **find** is used to match a group of filenames with a wild-car pattern, the pattern should be quoted to prevent the shell from looking at it:

```
find . -name "*.c" -print
find . -name '[A-Z]*' -print
```

All files with extension .c
Single quotes will also do

The first command looks for all C program source files in the current directory tree The second one searches for all files whose names begin with an uppercase letter. You must not forget to use the `-print` option because without it, **find** on UNIX systems will look for files all right but won't print the list.

find in UNIX displays the file list only if the `-print` operator is used. However, Linux doesn't need this option; it prints by default. Linux also doesn't need the path list; it uses the current directory by default. Linux even prints the entire file list when used without any options whatsoever! This behavior is not required by POSIX.

4.11.1 Selection Criteria

The `-name` operator is not the only operator used in framing the selection criteria; there are many others (Table 4.4). We'll consider the selection criteria first, and then the possible actions we can take on the selected files.

Locating a File by Inode Number (`-inum`) Refer to Section 4.7.1, where we found that **gzip** has three links and **gunzip** was one of them. **find** allows us to locate files by their inode number. Use the `-inum` option to find all filenames that have the same inode number:

```
$ find / -inum 13975 -print
find: cannot read dir /usr/lost+found: Permission denied
/usr/bin/gzip
/usr/bin/gunzip
/usr/bin/gzcat
```

Inode number obtained from Section 4.7.1

"Cats" a compressed file

Now we know what the three links are. Note that **find** throws an error message when it can't change to a directory. Read the following Tip.

TABLE 4.4 *Major Expressions Used by* **find** *(Meaning gets reversed when - is replaced by +, and vice versa)*

Selection Criteria	*Selects File*
`-inum` *n*	Having inode number *n*
`-type` *x*	If of type *x*; *x* can be `f` (ordinary file), `d` (directory), or `l` (symbolic link)
`-perm` *nnn*	If octal permissions match *nnn* completely
`-links` *n*	If having *n* links
`-user` *usname*	If owned by *usname*
`-group` *gname*	If owned by group *gname*
`-size` *+x*`[c]`	If size greater than *x* blocks (characters if `c` is also specified) (*Chapter 19*)
`-mtime` *-x*	If modified in less than *x* days
`-newer` *flname*	If modified after *flname* (*Chapter 19*)
`-mmin` *-x*	If modified in less than *x* minutes (*Linux only*)
`-atime` *+x*	If accessed in more than *x* days
`-amin` *+x*	If accessed in more than *x* minutes (*Linux only*)
`-name` *flname*	*flname*
`-iname` *flname*	As above, but match is case-insensitive (*Linux only*)
`-follow`	After following a symbolic link
`-prune`	But don't descend directory if matched
`-mount`	But don't look in other file systems
Action	***Significance***
`-print`	Prints selected file on standard output
`-ls`	Executes **`ls -lids`** command on selected files
`-exec` *cmd*	Executes UNIX command *cmd* followed by `{} \;`
`-ok` *cmd*	Like `-exec`, except that command is executed after user confirmation


Tip

If you use **find** from a nonprivileged account to start its search from root, the command will generate a lot of error messages on being unable to "cd" to a directory. Since you might miss the selected file in an error-dominated list, the error messages should be directed by using the command in this way: `find / -name typescript -print 2>/dev/null`. Note that you can't do this in the C shell. Section 6.7 explains the significance of `2>/dev/null`.

File Type and Permissions (-type and -perm) The `-type` option followed by the letter `f`, `d`, or `l` selects files of the ordinary, directory, and symbolic link type. Here's how you locate all directories of your home directory tree:

```
$ cd ; find . -type d -print 2>/dev/null
.                                          Shows the . also
./.netscape                                Displays hidden directories also
./java_progs
./c_progs
./c_progs/include
./.ssh
```

Note that the relative pathname **find** displays, but that's because the pathname itself was relative (.). **find** also doesn't necessarily display an ASCII sorted list. The sequence in which files are displayed depends on the internal organization of the file system.

The -perm option specifies the permissions to match. For instance, -perm 666 selects files having read and write permission for all user categories. Such files are security hazards. You'll often want to use two options in combination to restrict the search to only directories:

```
find $HOME -perm 777 -type d -print
```

find uses an AND condition (an implied -a operator between -perm and -type) to select directories that provide all access rights to everyone. It selects files only if both selection criteria (-perm and -type) are fulfilled.

Finding Unused Files (-mtime and -atime) Files tend to build up incessantly on disk. Some of them remain unaccessed or unmodified for months—even years. **find**'s options can easily match a file's modification (-mtime) and access (-atime) times to select them. The -mtime option helps in backup operations by providing a list of those files that have been modified, say, in less than two days:

```
find . -mtime -2 -print
```

Here, -2 means less than two days. To select from the /home directory all files that have not been accessed for more than a year, a positive value has to be used with -atime:

```
find /home -atime +365 -print
```


Note

+365 means greater than 365 days; -365 means less than 365 days. For specifying exactly 365, use 365.

4.11.2 The find Operators (!, -o, and -a)

There are three operators that are commonly used with **find**. The ! operator is used before an option to negate its meaning. So,

```
find . ! -name "*.c" -print
```

selects all but the C program files. To look for both shell and **perl** scripts, use the -o operator, which represents an OR condition. We need to use an escaped pair of parentheses here:

```
find /home \( -name "*.sh" -o -name "*.pl" \) -print
```

The (and) are special characters that are interpreted by the shell to run commands in a group *(7.6.2)*. The same characters are used by **find** to group expressions using the -o and -a operators, the reason why they need to be escaped.

The -a operator represents an AND condition, and is implied by default whenever two selection criteria are placed together.

4.11.3 Operators of the Action Component

Displaying the Listing (-ls) The -print option belongs to the *action* component of the **find** syntax. In real life, you'll often want to take some action on the selected files and not just display the filenames. For instance, you may want to view the listing with the -ls option:

```
$ find . -type f -mtime +2 -mtime -5 -ls                    -a option implied
475336   1 -rw-r--r--  1 romeo    users   716 Aug 17 10:31 ./c_progs/fileinout.c
```

find here runs the **ls -lids** command to display a special listing of those regular files that are modified in more than two days and less than five days. In this example, we see two options in the selection criteria (both -mtime) simulating an AND condition. It's the same as using \(-mtime +2 -a -mtime -5 \).

Taking Action on Selected Files (-exec and -ok) The -exec option allows you to run any UNIX command on the selected files. -exec takes the command to execute as its own argument, followed by {} and finally the rather cryptic symbols \; (backslash and semicolon). This is how you can reuse a previous **find** command quite meaningfully:

```
find $HOME -type f -atime +365 -exec rm {} \;                    Note the usage
```

This will use **rm** to remove all ordinary files unaccessed for more than a year. This can be a risky thing to do, so you can consider using **rm**'s -i option. But not all commands have interactive options, in which case you should use **find**'s -ok option:

```
$ find $HOME -type f -atime +365 -ok mv {} $HOME/safe \;
< mv ... ./archive.tar.gz > ? y
< mv ... ./yourunix02.txt > ? n
< mv ... ./yourunix04.txt > ? y
  .......
```

mv turns interactive with -i but only if the destination file exists. Here, -ok seeks confirmation for every selected file to be moved to the $HOME/safe directory irrespective of whether the files exist at the destination or not. A y deletes the file.

find is the system administrator's tool, and in Chapter 19, you'll see it used for a number of tasks. It is especially suitable for backing up files and for use in tandem with the **xargs** command (see Going Further of Chapter 6).


Note

The pair of {} is a placeholder for a filename. So, -exec cp {} {}.bak provides a .bak extension to all selected files. Don't forget to use the \; symbols at the end of every -exec or -ok option.