**Chapter 9**

**$Find and other useful Commands**

**1. find**

The "find" command is very powerful. It can search the entire filesystem for one or more files that we specify to look for. This is very helpful when a file has been "lost".

We can also use the find command to locate files, and then perform some type of action on the files after they've been located. With this capability, we can locate files using powerful search criteria, and then run any Unix command we want on the files we locate. (See the examples below.):

The find command allows the Unix user to process a set of files and/or directories in a file subtree.

We can specify the following:

* where to search (pathname)
* what type of file to search for (-type: directories, data files, links)
* how to process the files (-exec: run a process against a selected file)
* the name of the file(s) (-name)
* perform logical operations on selections (-o and -a)

**Syntax**

$ find <path> options action

Major options of the find command include

* -name : Finds files with certain naming conventions in the directory structure
* -ctime ***time interval***Locates files that that were created during the specified time interval
* -mtime ***time interval*** Finds files that have been modified during the specified time interval
* -atime **time interval** Locates files that have been accessed during the specified time interval
* -perm permissions Locates files with certain permission settings
* -user Locates files that have specified ownership
* -group Locates files that are owned by specified group
* -size Locates files with specified size
* -type Locates a certain type of file

Time interval in options -ctime, -mtime and -atime is an integer with optional sign.

* n: If the integer n does not have sign this means exactly n days ago, 0 means today.
* +n: if it has plus sing, then it means "more then n days ago", or older then n ,
* -n: if it has the minus sign, then it means less than n days ago (-n), or younger then n.
* It's evident that -1 and 0 are the same and means "today".

It is possible to locate files and directories that match or do not match multiple conditions, for example:

* a to have multiple conditions ANDed
* o to have multiple conditions ORed
* ! to negate a condition
* Expression to satisfy any complex condition

It is possible to specify the action to be taken on the files or directories that are found:

* print prints the names of the files on standard output (usually enabled by default)
* exec command executes the specified command.

The most common reason for using the find command is to utilize its capability to recursively process the subdirectories. For example, if we want to obtain a list of all files accessed in the last 24 hours, execute the following command:

**$ find . -atime 0 -print**

If the system administrator wants a list of .profile used by all users, the following command should be executed:

**$ find / -name .profile -print**

We can also execute the find command with multiple conditions. If we wanted to find a list of files that have been modified in the last 24 hours and which has a permission of 777, we would execute the following command:

**$ find . -perm 777 -a -mtime 0 –print**

**Total Options:**

|  |  |
| --- | --- |
| -atime n | True if the file was accessed n days ago. find updates the access times of the directories in the *pathname\_list.* |
| -ctime n | True if the file's status was changed n days ago. A change to a file means the inode of the file was modified. |
| -depth | Always true if specified on the command line. Causes find to descend the directory structures and perform the specified options on the subdirectories and files before processing the top directory. We may find this useful when using find to pipe pathnames to cpio. If we do not have write permission in directories this will allow we to transfer the files to the archive file. |
| -exec cmd | True if the command cmd executes successfully and returns a zero exit status. We can use a set of braces ({}) to signify the presence of the current pathname. The cmd command must end with an escaped semicolon (\;). For example,  **$ find . -name '\*.o' -exec rm {} \;**  Would remove all files ending with a .o from the current directory structure. |
| -group name | True if the file belongs to the group *name* .We may specify a group ID instead of a group name. |
| -inum n | True if the file has inode number n. |
| -links n | True if the file has n links. |
| -local | True if the specified file physically resides on the local UNIX system. This is useful when searching directories on systems that are networked together and share disk space. |
| -mtime n | True if the file's contents were modified n days ago. |
| -name file | True if the filename matches file. The shell's filename generation characters may be used ifenclosed in quotes. For example,  **$ find . -name '\*.c' –print**  Prints all files ending with a .c in the current directory or any of its subdirectories. |
| -newer file | True if the current file is newer than the specified file . The modification dates are compared to decide which is newer. |
| -nogroup | True if the file belongs to a group (groupid) not in /etc/passwd. |
| -nouser | True if the file belongs to a user (userid) not in /etc/passwd. |
| -ok cmd | Same as the -exec option except the command is displayed followed by a question mark. The command cmd is executed only if we respond with a y. |
| -perm on | True if the permissions of the file match the octal number on. Refer to Module 17 on chmod for a full description of the octal number representation used for on . We can prefix on with a minus sign to force the comparison of only the bits that are set in on to the file permission. |
| -print | Always true if specified on the command line. Causes the current pathname to be displayed. |
| -size n[c] | True if the file contains n blocks. If the n is followed by c then n is counted in characters instead of blocks. |
| -type c | True if the file type is type c , where c is one of the following. |

The find command checks the specified options, going from left to right, once for each file or directory encountered. The simplest invocation of find can be used to create a list of all files and directories below the current directory:

**$ find . –print**

We can use regular expressions to select files, for example those that have a .html suffix):

**$ find . -name "\*.html: -print**

We can search for files more recent than, older than, or exactly the same age as a specified date,

* -n - more recent then n days old
* +n - older then n days old
* n exactly of age n

Here are some useful examples. To find html files that have been modified in the last seven days, we can use –mtime with the argument -7 (include the hyphen):

**$ find . -mtime -7 -name "\*.html" -print**

If we just use the number 7 (without a hyphen), we will match only html files that were modified exactly seven days ago:

**$ find . -mtime 7 -name "\*.html" -print**

19. We can specify more than one directory as a starting point for the search. To look across the /bin and /usr directory trees for filenames that contain the pattern \.htm, we can use the following command:

**$ find /usr /bin -name "\*\.htm\*" -print**

20. To find a list of the directories use the -type specifier. Here's one example:

**find . -type d -print**

The most typical options for -type are as following:

* d –Directory
* f – File
* l – Link

**1.1 Using -exec option with find**

Find is able to execute one or more commands for each file it has found with the -exec option. Unfortunately, one cannot simply enter the command. We need to remember two tricks:

The command that we want to execute need to contain a special (obscure) argument {}, which will be replaced by the matched filename, and \; (or ';' ) at the end of the command. (If the \ is left out, the shell will interpret the; as the end of the find command.) If {} id is the last item in the command then it should be a space between the {} and the \;,

for example:

**$ find . -type d -exec ls -ld {} \;**

Here are several "global" chmod tricks based on fine -exec capabilities:

**$ find . -type f -exec chmod 500 {} ';'**

**$ find . -name "rc.conf" -exec chmod o+r '{}' ';'**

This command will search in the current directory and all sub directories for a file named rc.conf.

Note: The -print option will print out the path of any file that is found with that name. In general -print will print out the path of any file that meets the find criteria.

**$ find . -name core -ctime +4 -exec /bin/rm -f {} \;**

There's no output from this command because we didn't use the - print at the end of the command. What it does is find all files called "core" that have a creation time that's more than 4 days ago and remove them.

The find command is a powerful command in UNIX. It helps us find files by owner, type, filename, and other attributes. The most awkward part of the command is the required elements of the -exec option, and that's where the xargs command helps immensely.

**2. Mail**

The mail command is a quick and easy way to send an email to someone. Just type mail and the address of the person we want to mail. W e will then be prompted for a subject and any cc's. Then just type our message and control-d on a line by itself to send the mail

**2.1 Mail Headers**

Mail headers have the following construction

**$mail <username>**

**Subject: Title describing the message (optional)**

**Cc: List of people to receive a carbon copy (optional)**

**Bcc: List of people to receive blind carbon copy (they do not see user names in the received message. Optional)**

**2.2 Mailboxes**

UNIX uses two mailboxes to hold mail messages

**system mailbox (/vsr/spool/mail/)**

**user mail box (..../.../mbox)**

Mail arrives in the system mailbox, and is saved in our user mail box after we have read it. The user mail box is normally located in their $HOME directory.

To list all mails which he got say command only mail without any arguments Type

**$mail**

The mail program displays a title message and lists all available mail headers,

**SCO System V Mail (version 3.2) Type ? for help.**

**"/usr/spool/mail/brianb": 3 messages 3 new**

**N 3 brianb Mon May 31 15:02 10/299 My message3**

**N 2 brianb Mon May 31 15:01 9/278**

**>N 1 brianb Mon May 31 15:00 12/415 My first message**

**&**

This initial screen displays the subject fields of messages which have arrived. The format of the display is,

**Type Message\_number From\_User Date/Time Subject**

**N denotes a new message**

**> denotes the current message**

**& mail prompt symbol**

Note how message number 2 does not have a subject heading. This is because the mail message was sent from a file, and the -s option was not specified on the command line when the mail program was invoked. To read message type number associated to that mail. To quit from this mail prompt say q.

**2.3 Sending Mail to People at Other Host Machines or Sites**

We send mail to other people at remote sites by specifying their specific mail address.

**Examples**

**$ mail your\_user\_name@server.com**

**3. write**

The write command is used to send on-line messages to another user on the same machine. The format of the write command is as follows:

**$ write username**

**text of message**

**^D**

After typing the command, we enter our message, starting on the next line, terminating with the end-of-file character. The recipient will then hear a bleep, then receive our message on screen, with a short header attached. The following is a typical exchange. User UserRavi types:

**$ write UserRavi**

**Hi there - want to go to lunch?**

**^D**

**$**

User lnp8zz will hear a beep and the following will appear on his/her screen:

**Message from UserRavi on sun050 at 12:42**

**Hi there - want to go to lunch?**

**EOF**

If UserAjay wasn't logged on, the sender would see the following:

**$ write UserAjay**

**UserAjay not logged in.**

**4. talk**

The command talk creates a two-way, screen-oriented communication program. It allows users to type simultaneously, with their output displayed in separate regions of the screen. To send a talk request, type talk and then <user name>@<their machine>. So, for example, to send a talk request to dmb, who is logged on to cslab0a, type talk raju@company. The recipient of the request is then prompted for a response.

**Example:**

**$ talk <username>**

**$ talk username@<their machinename> [to send a message to remote user]**

Note: The mesg command may, of course, be used to disable write,talk access to a terminal.

We can stop messages being flashed up on our screen if we wish. To turn off direct communications type:

**% mesg n**

It will remain off for the remainder of our session, unless we type:

**% mesg y**

To turn the facility back on. Typing just mesg lets we know whether it is on or off.

**5. walls**

**wall** is a Unix command line utility. That would only invoked by Administrator, it displays the contents of a file or standard input to all logged-in users.

**Invocation**

wall is invoked as follows:

**$ raju@company:# wall <filename>**

To take standard input as the file, omit the filename.

**Output:**

When invoked, wall displays the following output:

**Broadcast Message from raju@wilshiresoft**

**(/dev/pts/0) at 01:23 ...**

**6. Pine (e-mail client)**

Pine is a powerful freeware e-mail client: the University of Washington's " Program for Internet News & Email." Many people believe that Pine stood for "Pine is not Elm." However, its original author, Laurence Lundblade

**$pine <Enter>**

**Exercises:**

1. Send a message to another user on our Unix system, and get them to reply.

2. Create a small text file and send it to another user.

3. When we receive a message, save it to a file other than our mailbox. (Remember we can always send

yourself a message if we don't have one.)

4. Send a message to a user on a different computer system.

5. Send a note to our course tutor telling him that we can use mail now.