CHAPTER 5

File Searching Utilities

5.1 THE WHICH COMMAND

Command **which** is a UNIX command used to identify the location of executables. This takes one or more arguments; for each of these arguments, it prints to stdout the full path of the executable that would have been executed if this argument had been entered at the shell prompt. It does this by searching for an executable or script in the directories listed in the environment variable PATH.

Example usage :

which ls gives output as :
/usr/bin/ls
which ls echo gives output as :
/usr/bin/ls
/usr/bin/echo

5.2 THE TYPE COMMAND

This is an internal command of shell. Its syntax is given as :

type [-aftpP] name [name ...]

With no options, indicate how each name would be interpreted if used as a command name. If the -t option is used, type prints a string which is one of alias, keyword, function, built-in, or file if name is an alias, shell reserved word, function, built-in, or disk file, respectively. If the name is not found, then nothing is printed, and an exit status of false is returned. If the -p option is used, type either returns the name of the disk file that would be executed if name were specified as a command name, or nothing if "type -t name" would not return file. The -P option forces a PATH search for each name, even if "type -t name" would not return file. If a command is hashed, -p and -P print the hashed value, not necessarily the file that appears first in PATH. If the -a option is used, type prints all of the places that contain an executable named name. This includes aliases and functions, if and only if the -p option is not also used. The table of hashed commands is not consulted when using -a. The -f option suppresses shell function lookup, as with the command built-in. type returns true if any of the arguments are found, false if none are found.

5.3 THE WHEREIS COMMAND

This command can be used to locate binaries, documentation files, etc., The syntax of this command is given as :

whereis **[** -bmsu **] [** -BMS ***directory*...** -f **]** ***filename* ...**

The command whereis locates source/binary and manuals sections for specified files. The supplied names are first stripped of leading pathname components and any (single) trailing extension of the form **.***ext,* for example, **.c**. Prefixes of **s.** resulting from use of source code control are also dealt with. **whereis** then attempts to locate the desired program in a list of standard Linux places.

The available OPTIONs are :

-b

Search only for binaries.

-m

Search only for manual sections.

-s

Search only for sources.

-u

Search for unusual entries. A file is said to be unusual if it does not have one entry of each requested type. Thus '**whereis -m -u ***' asks for those files in the current directory which have no documentation.

-B

Change or otherwise limit the places where **whereis** searches for binaries.

-M

Change or otherwise limit the places where **whereis** searches for manual sections.

-S

Change or otherwise limit the places where **whereis** searches for sources.

-f

Terminate the last directory list and signals the start of file names, and *must* be used when any of the **-B**, **-M**, or **-S** options are used.

Example

To find all files in **/usr/bin** which are not documented in **/usr/man/man1** with source in **/usr/src** :

```
cd /usr/bin
whereis -u -M /usr/man/man1 -S /usr/src -f *
```

5.4 THE FIND COMMAND

The find command is used to search through the directories of a file system looking for files that match a specific criteria. Once a file matching the criteria is found the find command can be told to perform a number of different tasks including running any UNIX command on the file.

This is the command which is commonly used by the system administrator. A common task for a Systems Administrator is searching the UNIX file hierarchy for files which match certain criteria. Some common examples of what and why a Systems Administrator may wish to do this include

- searching for very large files
- finding where on the disk a particular file is
- deleting all the files owned by a particular user
- displaying the names of all files modified in the last two days.

Given the size of the UNIX file hierarchy and the number of files it contains this isn't a task that can be done by hand. This is where the find command becomes useful.

find command format

The format for the find command is

find [*path-list*] [*expression*]

path-list is a list of directories in which the find command will search for files. The command will recursively descend through all sub-directories under these directories. The expression component is explained in the next section.

Both the path and the expression are optional. If we run the find command without any parameters it uses a default path as the current directory, and a default expression as printing the name of the file. Thus, when we run find command we may get all the entries of current directory.

find expressions

A find expression can contain the following components :

- options,
  These modify the way in which the find command operates.
- tests,
  These decide whether or not the current file is the one you are looking for.
- actions,
  Specify what to do once a file has been selected by the tests.
- and operators.
  Used to group expressions together.

find options

Options are normally placed at the start of an expression. Table 1 summarizes some of the find commands options.

Options are

- **-name** True if pattern matches the current file name. Simple regex (shell regex) may be used. A backslash (\) is used as an escape character within the pattern. The pattern should be escaped or quoted. If we need to include parts of the path in the pattern in GNU find, we should use predicate wholename.
  Use the -iname predicate (GNU find supports it) to run a case-insensitive search, rather than just -name. For example :

find . -follow -iname '*.htm' -print

- **-fstype *type*** True if the file system to which the file belongs is of type *type*. For example on Solaris mounted local file systems have type **ufs** (Solaris 10 added **zfs**). For AIX local file system is **jfs** or **jfs2** (journalled file system). If you want to traverse NFS file systems you can use **nfs** (network file system). If you want to avoid traversing network and special file systems you should use predicate **local** and in certain circumstances **mount**
- **"-atime/-ctime/-mtime" [+|-]*n***
  Specify selection of the files based on ***three UNIX timestamps:*** the last time a files's "access time", "file status" and "modification time".
  n is ***time interval*** — an integer with optional sign. It is measured in 24-hour periods (days) or minutes counted from the current moment.
  - **n:** If the integer n does not have sign this means ***exactly n 24-hour periods (days) ago***, 0 means today.
  - **+n:** if it has plus sing, then it means ***"more then n 24-hour periods (days) ago", or older then n***,
  - **-n:** if it has the minus sign, then it means ***less than n 24-hour periods (days) ago (-n), or younger then n***. It's evident that -1, and 0 are the same and both means "today".
  - **n:** If the integer n does not have sign this means ***exactly n 24-hour periods (days) ago***, 0 means today.
  - **+n:** if it has plus sing, then it means ***"more then n 24-hour periods (days) ago", or older then n***,
  - **-n:** if it has the minus sign, then it means ***less than n 24-hour periods (days) ago (-n), or younger then n***. It's evident that -1 and 0 are the same and both means "today".
- **-newer/-anewer/-cnewer *baseline_file*** The time of modification, access time or creation time are compared with the same timestamp in the baseline file. If file is a symbolic link and the -H option or the -L option is in effect, the modification time of the file it points to is always used.
  - **-newer** Modification time is compared with modification time of the *basline_file* True if file was modified more recently than baseline file.
  - **-anewer** Access time is compared with access time of *basline_file* . True if file was last accessed more recently than baseline file.
  - **-cnewer** Creation file is compared. For example: find everything in your home that has been modified more recently than "**~joeuser/lastbatch.txt** ":

    ```
    • find $HOME -newer ~joeuser/lastbatch.txt
    ```

- **-local** True if the file system type is not a remote file system type. In Solaris those types are defined in the **/etc/dfs/fstypes** file. nfs is used as the default remote file system type if the **/etc/dfs/fstypes** file is not present. The **-local** option skips the hierarchy of non-local directories. You can also search without descending more then certain number of levels as explained later or exclude some directories from the search using
- **-mount** Always true. Restricts the search to the file system containing the directory specified. Does not list mount points to other file systems.

- -xdev Same as the -mount primary. Always evaluates to the value True. Prevents the **find** command from traversing a file system different from the one specified by the *Path* parameter.
- -xattr True if the file has extended attributes.
- **-wholename *simple-regex*** [GNU find only] . File name matches simple regular expression (often called shell patterns). In simple regular expressions the metacharacters **'/'** and **'.'** do not exist; so, for example, you can specify :

  find . -wholename '/lib*'

  which will print entries from directories **/lib64** and **/lib**. ***To ignore the directories specified, use option -prune*** For example, to skip the directory **/proc** and all files and directories under it (which is important for linux as otherwise errors are produced you can something like this :

  find . -wholename '/proc' -prune -o -name *file_to_be_found*

Other useful options of the find command include :

1. -regex *regex [GNU find only]* File name matches regular expression. This is a match on the whole pathname not a filename. Stupidly enough the default regular expressions understood by find are Emacs Regular Expressions, not Perl regular expressions. It is important to note that "-iregex" option provide capability to ignore case.
2. -perm *permissions* Locates files with certain permission settings. Often used for finding world-writable files or SUID files.
3. -user Locates files that have specified ownership. Option **-nouser** locates files without ownership. For such files no user in **/etc/passwd** corresponds to file's numeric user ID (UID). Such files are often created when tar of sip archive is transferred from other server on which the account probably exists under a different UID).
4. -group Locates files that are owned by specified group. Option **-nogroup** means that no group corresponds to file's numeric group ID (GID) of the file.
5. -size Locates files with specified size. -size attribute lets you specify how big the files should be to match. You can specify your size in kilobytes and optionally also use + or - to specify size greater than or less than specified argument. For example :

   find /home -name "*.txt" -size 100k

   find /home -name "*.txt" -size +100k

   find /home -name "*.txt" -size -100k

   The first brings up files of exactly 100KB, the second only files greater than 100KB, and the last only files less than 100KB.
6. **-ls** list current file in 'ls -dils' format on standard output. Try the command :

   find . –ls –print .
7. -type Locates a certain type of file. The most typical options for -type are as following :
   - **d** -Directory
   - **f** - File
   - **l** - Link

   For example to find a list of the directories use can use the -type specifier. Here's one example :

   find . -type d -print

Table 1. find options

Option	Effect
-daystart	for tests using time measure time from the beginning of today
-depth	process the contents of a directory before the directory
-maxdepth *number*	*number* is a positive integer that specifies the maximum number of directories to descend
-mindepth *number*	*number* is a positive integer that specifies at which level to start applying tests
-mount	don't cross over to other partitions
-xdev	don't cross over to other partitions

For example

The following are two examples of using find's options. As we did not specify any path, the current directory, is used.

find -mindepth 2

./Adirectory/oneFile

In this example the mindepth option tells find to only find files or directories which are at least two directories below the starting point, starting directory.

find -maxdepth 1

This option restricts find to those files which are in the current directory.

find tests

Tests are used to find particular files based on :

- when the file was last accessed
- when the file's status was last changed
- when the file was last modified
- the size of the file
- the file's type
- the owner or group owner of the file
- the file's name
- the file's i-node number
- the number and type of links the file has to it
- the file's permissions

Table 2 summarizes find's tests. A number of the tests take numeric values. For example, the number of days since a file was modified. For these situations the numeric value can be specified using one of the following formats (in the following *n* is a number)

- *+n*
  greater than *n*
- *-n*
  less than *n*
- *n*
  equal to *n*

For example

Some examples of using tests are shown below. Note that in all these examples no command is used. Therefore the find command uses the default command which is to print the names of the files.

- find . -user david
  Find all the files under the current directory owned by the user david
- find / -name *.html
  Find all the files one the entire file system that end in .html. **Notice** that the * must be quoted so that the shell doesn't interpret it (explained in more detail below). Instead we want the shell to pass the *.html to the find command and have it match filenames.
- find /home -size +2500k -mtime –7
  Find all the files under the /home directory that are greater than 2500 kilobytes in size and have been in modified in the last seven days.

The last example shows it is possible to combine multiple tests. It is also an example of using numeric values. The +2500 will match any value greater than 2500. The -7 will match any value less than 7.

find actions

Once the files are found, some operations can be done on them. The find command provides a number of actions most of which allow to either

- execute a command on the file, or
- display the name and other information about the file in a variety of formats

For the various find actions that display information about the file you are urged to examine the manual page for find

Executing a command

find has two actions that will execute a command on the files found. They are -exec and -ok.

The format to use them is as follows

-exec *command* ;

-ok *command* ;

command is any UNIX command.

The main difference between exec and ok is that ok will ask the user before executing the command. exec just does it.

Table 2. find tests

Test	Effect
-amin *n*	file last access *n* minutes ago
-anewer *file*	the current file was access more recently than *file*
-atime *n*	file last accessed *n* days ago
-cmin *n*	file's status was changed *n* minutes ago
-cnewer *file*	the current file's status was changed more recently than *file*'s
-ctime *n*	file's status was last changed *n* days ago

-mmin *n*	file's data was last modified *n* minutes ago
-mtime *n*	the current file's data was modified *n* days ago
-name *pattern*	the name of the file matches *pattern* -iname is a case insensitive version of –name -regex allows the use of REs to match filename
-nouser-nogroup	the file's UID or GID does not match a valid user or group
-perm *mode*	the file's permissions match *mode* (either symbolic or numeric)
-size *n*[bck]	the file uses *n* units of space, b is blocks, c is bytes, k is kilobytes
-type *c*	the file is of type *c* where *c* can be block device file, character device file, directory, named pipe, regular file, symbolic link, socket
-uid *n* -gid *n*	the file's UID or GID matches *n*
-user *uname*	the file is owned by the user with name *uname*

For example

Some examples of using the exec and ok actions include :

- find . -exec grep hello \{\} \;
  Search all the files under the local directory for the word hello.
- find / -name *.bak -ok rm \{\} \;
  Find all files ending with .bak and ask the user if they wish to delete those files.

{} and ;

The exec and ok actions of the find command make special use of {} and ; characters. Since both {} and ; have special meaning to the shell they must be quoted when used with the find command.

{} is used to refer to the file that find has just tested. So in the last example rm \{\} will delete each file that the find tests match.

The ; is used to indicate the end of the command to be executed by exec or ok.

For example :

The following command is used to locate files in the UNIX directory tree.

find directoryname -name filenametobefound

Example

find / -name core

The above command displays all the occurrences of the file named core under / directory.

find . -ctime 2 -name

The above command displays names of those files which are created in the last two days and are in the current directory.

The following command displays names of those files which are modified in the last two days and are in the current directory.

find . -mtime 2 -name

To display names of the files of C.W.D whose size is greater than 10 blocks, we can run the following command.

find . -size 10 -name

find . -type d -name

The above command displays names of directories in the current directory.

find / -name Chapter1 -type f -print

This command searches through the root file system ("/") for the file named "Chapter1". If it finds the file, it prints the location to the screen.

find /usr -name Chapter1 -type f -print

This command searches through the "/usr" directory for the file named "Chapter1".

find /usr -name "Chapter*" -type f -print

This command searches through the "/usr" directory for all files that begin with the letters "Chapter". The filename can end with any other combination of characters. This will match filenames such as "Chapter", "Chapter1", "Chapter1.bad", "Chapter_in_life".

find /usr/local -name "*.html" -type f -print

This command searches through the "/usr/local" directory for files that end with the extension ".html". These file locations are then printed to the screen.

find /usr/local -name "*.html" -type f -exec chmod 644 {} \;

This command searches through the "/usr/local" directory for files that end with the extension ".html". When these files are found, their permission is changed to mode 644 (rw-r—r—).

find htdocs cgi-bin -name "*.cgi" -type f -exec chmod 755 {} \;

This command searches through the "htdocs" and "cgi-bin" directories for files that end with the extension ".cgi". When these files are found, their permission is changed to mode 755 (rwxr-xr-x). This example shows that the find command can easily search through multiple sub-directories (htdocs, cgi-bin) at one time.

We say that all the following commands works similarly. That is, all of them simply print the files of C.W.D. Is it true?.

find
find . -print
find -print
find .

Some times find command produces many error messages which may annoy us. Thus, we can redirect the error messages to null device through redirection in the following way.

find / -name foo **2>/dev/null**

Advanced Features And Applications

The -print action lists the files separated by a space when the output is *piped* to another command. This can lead to a problem if any found files contain spaces in their names, as the output doesn't use any quoting. In such cases, when the output of find contains a file name such as foo bar and is piped into another command, that command sees two file names, not one file name containing a space.

In such cases you can specify the action -print0 instead, which lists the found files separated not with a space, but with a *null* character (which is not a legal character in UNIX or Linux file names). Of course the command that reads the output of find must be able to handle such a list of file names. Many commands commonly used with find (such as tar or cpio) have special options to read in file names separated with NULL's instead of spaces.

We can also use shell-style wildcards in the -name search argument :

find . -name foo*bar

This will search from the current directory down for foo*bar (that is, any filename that begins with foo and ends with bar). Note that wildcards in the name argument must be quoted so that the shell doesn't expand them before passing them to find. Also, unlike regular shell wildcards, these will match leading periods in filenames. (For example find -name *.txt.)

We can also search for other criteria beside the name. Also we can list multiple search criteria. When we have multiple criteria any found files must match all listed criteria. That is, there is an implied Boolean *AND* operator between the listed search criteria. find also allows *OR* and *NOT* Boolean operators, as well as grouping, to combine search criteria in powerful ways (not shown here.)

The following command will find any regular files (*i.e.,* not directories or other special files) with the criteria -type f, and only those modified seven or fewer days ago (-mtime -7). The command xargs, a handy utility that coverts a stream of input (in this case the output of find) into command line arguments for the supplied command (in this case tar, used to create a backup archive).

find / **-type f -mtime -7** | xargs tar -rf weekly_incremental.tar

Using the tar option -c is dangerous here; xargs may invoke tar several times if there are many files found and each -c will cause tar to over-write the previous invocation. The -r option *appends* files to an archive. Other options such as those that would permit filenames containing spaces would be useful in a production quality backup script.

Another use of xargs is illustrated below. This command will efficiently remove all files named core from your system (provided you run the command as root of course) :

find / -name core | xargs /bin/rm -f

find / -name core -exec /bin/rm -f '{}' \; # same thing

find / -name core -delete # same if using Gnu find

(The last two forms run the rm command once per file, and are not as efficient as the first form.)

The following command will display the files which are modified in the last 10 minutes.

find / -mmin -10

The option **–user username** can be used to locate files owned by a user.

We may often required to locate files with the given permissions. In this situation, we can use –perm option. We can find files with various permissions set. -perm /*permissions* means to find files with **any** of the specified *permissions* on, -perm -*permissions* means to find files with **all** of the specified *permissions* on, and -perm *permissions* means to find files with **exactly** *permissions*. *Permissions* can be specified either symbolically (preferred) or with an octal number. The following will locate files that are writeable by others (including symlinks, which should be writeable by all) :

find . -perm -o=w

When using find to locate files for backups, it often pays to use the -depth option (really a criteria that is always true), which forces the output to be *depth-first*—that is, files first and then the directories containing them. This helps when the directories have restrictive permissions, and restoring the directory first could prevent the files from restoring at all (and would change the time stamp on the directory in any case). Normally, find returns the directory first, before any of the files in that directory. This is useful when using the -prune action to prevent find from examining any files you want to ignore :

find / -name /dev -prune | xargs tar ...

When specifying time with find options such as -mmin (minutes) or -mtime (24 hour periods, starting from now), we can specify a number *n* to mean exactly *n*, *-n* to mean less than *n*, and *+n* to mean more than *n*. Fractional 24-hour periods are truncated! That means that find -mtime +1 says to match files modified **two or more days ago.**

For example :

```
find . -mtime 0          # find files modified between now and 1 day ago
                         # (i.e., within the past 24 hours)
find . -mtime -1         # find files modified less than 1 day ago
                         # (i.e., within the past 24 hours, as before)
find . -mtime 1          # find files modified between 24 and 48 hours ago
find . -mtime +1         # find files modified more than 48 hours ago
find . -mmin +5 -mmin -10    # find files modified between
                             # 6 and 9 minutes ago
```

Using the -printf action instead of the default -print is useful to control the output format better than you can with ls or dir. You can use find with -printf to produce output that can easily be parsed by other utilities, or imported into spreadsheets or databases. (See the man page for the dozens of possibilities with the -printf action.) The following command display's non-hidden (no leading dot) files in the current directory only (no subdirectories), with a custom output format :

find . -maxdepth 1 -name '[!.]*' -printf 'Name: %16f Size: %6s\n'

As a system administrator we can use find to locate suspicious files (e.g., world writable files, files with no valid owner and/or group, SetUID files, files with unusual permissions, sizes, names, or dates). See the following command.

```
find / -noleaf -wholename '/proc' -prune \
      -o -wholename '/sys' -prune \
      -o -wholename '/dev' -prune \
      -o -perm -2 ! -type l ! -type s \
      ! \( -type d -perm -1000 \) -print
```

This says to search the whole system, skipping the directories /proc, /sys, and /dev. The Gnu -noleaf option tells find not to assume all remaining mounted file systems are UNIX file systems (We might have a mounted CD for instance). The -o is the Boolean OR operator, and ! is the Boolean NOT operator (applies to the following criteria). Thus, the above criteria says to locate files that are world writable (-perm -2, same as -o=w) and NOT symlinks (! -type l) and NOT *sockets* (! -type s) and NOT directories with the *sticky* (or *text*) bit set (! \(-type d -perm -1000 \)). (Symlinks, sockets and directories with the sticky bit set are often world-writable and generally not suspicious.)

Let us assume that we wanted to find all the hard links to some file. First, using ls -li *file* will tell we can find how many hard links the file has, and the *i-node number*. We can locate all pathnames to this file with :

find *mount-point* -xdev -inum *i-node-number*

Since hard links are restricted to a single file system, we need to search that whole file system so we start the search at the file system's *mount point*. (This is likely to be either /home or / for files in your home directory.) The -xdev options tells find to not search any other file systems.

(While most UNIX and all Linux systems have a find command that supports the -inum criteria, this isn't POSIX standard. Older UNIX systems provided the ncheck utility instead that could be used for this.)

Using -exec Efficiently

We already know that the -exec option to find is great, but since it runs the command listed for every found file, it isn't very efficient. On a large system this makes a difference! One solution is to combine find with xargs.

find *whatever...* | xargs *command*

However this approach has two limitations. Firstly not all commands accept the list of files at the end of the command. A good example is cp :

find . -name *.txt | xargs cp /tmp # This won't work!

(Note the Gnu version of cp has a non-POSIX option -t for this, and xargs has options to handle this too.)

Secondly filenames may contain spaces or newlines, which would confuse the command used with xargs. (Again Gnu tools have options for that, **find ... -print0 |xargs -0 ....**)

There are POSIX (but non-obvious) solutions to both problems. An alternate form of -exec ends with a plus-sign, not a semi-colon. This form collects the filenames into groups or sets, and runs the command once per set. (This is exactly what xargs does, to prevent argument lists from becoming too long for the system to handle.) In this form the {} argument expands to the set of filenames. For example :

find / -name core -exec /bin/rm -f '{}' +

This form of -exec can be combined with a shell feature to solve the other problem (names with spaces). The POSIX shell allows us to use :

sh -c '*command-line*' [*command-name* [*args*...]]

Here is an example of efficiently copying found files to /tmp, in a POSIX-compliant way (Posted on comp.UNIX.shell netnews newsgroup on Oct. 28 2007 by Stephane CHAZELAS) :

find . -name '*.txt' -type f \
 -exec sh -c 'exec cp -f "$@" /tmp' find-copy {} +

For example often one needs to find files containing a specific pattern in multiple directories one can use an exec option in find (please note that you should use the -l flag for grep so that grep specifies the matched filenames) :

find . -type f -exec grep -li '/bin/ksh' {} \;

The same thing can be written using xarg and pipes. You can use the xargs to read the output of find and build a pipelines that invokes grep. This way, grep is called only four or

five times even though it might check through 200 or 300 files. By default, xargs always appends the list of filenames to the end of the specified command, so using it is as easy as can be :

find . -type f -print | xargs grep -li 'bin/ksh'

This gave the same output, but it was a lot faster. Also when grep is getting multiple filenames, it will automatically include the filename of any file that contains a match so option for grep **-l** is redundant :

find . -type f -print | xargs grep -i 'bin/ksh'

The other useful options for xargs are: -p option, which makes xargs interactive, and the -n args option, which makes xargs run the specified command with only args number of arguments.

In the following command, we will be asked yes or no to each file.

find /mnt/zip -name "*prefs copy" -print0 | xargs -p rm

We have already habituated in locating files and directories that match or do not match multiple conditions forming complex logical expression. Expressions can contain "escaped parentheses": parentheses have a special meaning in shell, so we need to escape that meaning, and write them as \(and \) or inside of single quotes as '(' and ')'. We cannot use single quotes around the entire expression though, as that will confuse the find command. *It wants each predicate as its own word.*

For example :

- a **—** to have multiple conditions connected using logical AND. By default options are concatenated using AND predicate to this operations is rarely used.
- o **—** to have multiple conditions connected using logical OR. Usually in used in brackets to ensure proper order of evaluation. For example \(-perm -4000 -o -perm -2000 \)
- ! **—** to negate a condition (logical NOT) . NOT should be specified with a backslash before exclamation point (\!). For example
  find . \! -name "*.gz" -exec gzip {} \;
- \(*expression* **\) —** "escaped parentheses" to specify any composite condition. For example
  find / -type f \(-perm -4000 -o -perm -2000 \) -exec ls -l {} \;

By default options are concatenated using AND predicate. For example, if you want to obtain a list of all files accessed in the last 24 hours, execute the following command (with or without **-print** option) :

find . -atime 0 -print

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

find / -name .profile -print

We can also specify multiple "AND" conditions (AND logical condition is a default so you do not specify it explicitly). If you wanted to find a list of files that have been modified in the last 24 hours and which has a permission of 777, you would execute the following command :

find . -perm 777 -mtime 0 -print

Which is the same as :

find . -perm 777 -a -mtime 0 -a -print

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 an .html suffix we can use the following.

find . -name "*.html" -print

To find everything in our home directory modified in the last 24 hours :

find $HOME -mtime -1

To find everything in our home directory modified in the last seven 24-hour periods (days) :

find $HOME -mtime -7

To find everything in our home directory that has NOT been modified in the last year :

find $HOME -mtime +365

To find html files that have been modified in the last seven 24-hour periods (days), we can use **-mtime** with the argument -7 (include the hyphen) :

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

Note: If you use the number 7 (without a hyphen), find will match only html files that were modified exactly seven 24-hour periods (days) ago :

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

To find those html files that were not touched for at least seven 24-hour periods (days), use +7 :

find . -mtime +7 -name "*.html" -print

Common Confusion Clarified

UNIX keeps track of three timestamps. Of them **atime** stands for access time which is when the file was last read. The first important thing to understand the precise meaning of **ctime** and **mtime** timestamps. The most common misconception here is to view **ctime** as file "creation time". It is actually "change time". Here are more formal explanations:

- **ctime** is the i-node change time. When does the i-node change, when you of course update a file, but also when you do things like changing the permissions of the file but not necessarily its contents. It would be better to call this attribute *change time* as it indicates the last time a file's metadata (i-node) was changed. **ctime** changes when you change file's ownership or access permissions. As the man page for stat explains: "The field **st_ctime** is changed by writing or by setting i-node information (*i.e.,* owner, group, link count, mode, etc.)."

- **mtime**: is the modification time, so if you change the contents of the file, this timestamp is updated. Changes of name, ownership and permissions does not affect it

For a given file **ctime** and **mtime** can be different depending on if we just modified the i-node or the contents of the file (which updates ctime as well). Commands like **chown** and **chmod, ln** change only **ctime**. Touch command change only **mtime**. For example if we need to change the date to Jun 21, 2008 9AM to example.txt, then we can go (**-t** parameter in touch has format [[CC]YY]MMDDhhmm[.SS]) :

touch -t 200907210900 example.txt

Those 24 hours periods are usually called "days" but the definition of "days" used in find is different from common usage (calendar days were each new day starts at midnight). The "day" in "find language" is interpreted as ***"24 hour periods starting from the current time".*** Here is how working with time ranges described in GNU find documentation.

For example, to list files in $HOME/nbv that were last read from 2 to 6 minutes ago:

find $HOME/nbv -amin +2 -amin -6

The option -**daystart** measure times from the beginning of today rather than from 24 hours ago. So, to list the regular files in our home directory that were modified yesterday, do

find ~/ -daystart -type f -mtime 1

The '-daystart' option is unlike most other options in that it has an effect on the way that other tests are performed. The affected tests are '-amin', '-cmin', '-mmin', '-atime', '-ctime' and '-mtime'. The '-daystart' option only affects the behavior of any tests which appear after it on the command line.

Comparing Timestamps

We compare locate files based on their relative time stamps with a given file. For this we can use –newer option.

-newerXY *reference*

Succeeds if timestamp 'X' of the file being considered is newer than timestamp 'Y' of the file reference. The letters 'X' and 'Y' can be any of the following letters :

'a' Last-access time of reference

'B' Birth time of reference (when this is not known, the test cannot succeed)

'c' Last-change time of reference

'm' Last-modification time of reference

't'

The reference argument is interpreted as a literal time, rather than the name of a file Tests of the form '-newerXt' are valid but tests of the form '-newertY' are not.

For example, **find . -newerac aa** succeeds for all files which have been accessed more recently than file aa was changed. Here 'X' is 'a' and 'Y' is 'c'.

Not all files have a known birth time. If 'Y' is 'b' and the birth time of reference is not available, find exits with an explanatory error message. If 'X' is 'b' and we do not know the birth time the file currently being considered, the test simply.

Some operating systems (for example, most implementations of UNIX) do not support file birth times. Some others, for example NetBSD-3.1, do. Even on operating systems which support file birth times, the information may not be available for specific files. For example, under NetBSD, file birth times are supported on UFS2 file systems, but not UFS1 file systems.

There are two ways to list files in /usr modified after February 1 of the current year. One uses '-newermt' :

find /usr -newermt "Feb 1"

The other way of doing this works on the versions of find before 4.3.3 :

```
touch -t 02010000 /tmp/stamp$$
find /usr -newer /tmp/stamp$$
rm -f /tmp/stamp$$
```

Also we do have some other tests related to timestamps.

-anewer *file*

-cnewer *file*

-**newer** *file*

True if the file was last accessed (or its status changed, or it was modified) more recently than *file* was modified. These tests are affected by '-follow' only if '-follow' comes before them on the command line. As an example, to list any files modified since /bin/sh was last modified :

find . -newer /bin/sh

-used*n*

True if the file was last accessed *n* days after its status was last changed. Useful for finding files that are not being used, and could perhaps be archived or removed to save disk space.

The following command will select the files that are at least one week old (7 days) but less then 30 days old :

find . -mtime +30 -a -mtime -7 -print0

5.5 THE LOCATE COMMAND

Locate [-d path | —database=path] [-e | —existing] [-i | —ignore-case] [—version] [—help] pattern...

Locate provides a secure way to index and quickly search for files on your system. It uses index database to sped search but that means that it dependent of the currency of the database. This is a deficiency: you buy speed at the expense of currency. The index database to make searching much faster then find. \

-d path—database=path	Instead of searching the default file name database, search the file name databases in path, which is a colon-separated list of database file names. You can also use the environment variable LOCATE_PATH to set the list of database files to search. The option overrides the environment variable if both are used. The file name

	database format changed starting with GNU find and locate version 4.0 to allow machines with different byte orderings to share the databases. This version of locate can automatically recognize and read databases produced for older versions of GNU locate or UNIX versions of locate or find.
-e—existing	Only print out such names that currently exist (instead of such names that existed when the database was created). Note that this may slow down the program a lot, if there are many matches in the database.
-i—ignore-case	Ignore case distinctions in both the pattern and the file names.
—help	Print a summary of the options to locate and exit.
—version	Print the version number of locate and exit.

Examples:

locate perl	Locate file perl in the DB and Print the path.
locate -i perl	Same search as above, case insensitive.
locate -q perl	Run in Quiet Mode.
locate -n 2 perl	Limit the no. of results shown to 2 first.
locate -U locater -o locateDB	Create index DB starting at locater and store the index file in locateDB.

In the above example, the system would locate perl on the local machine.

Note: We may need to run the "updatedb" command to update the database in order to find the file you are searching for. This command should be ran from cron daily or several times a day.

5.6 CONCLUSIONS

This chapter introduces to file searching commands such as find, whereis, and type, locate, etc.,. We have included detailed description of find command along with many live examples. Some of the examples which involve piping can be advised to refer after reading the chapter on piping. Also, we have given emphasis that find is a handy tool for system administration.