finder-keepers.

In it's simplest use the find command searches for files in the current directory and its subdirectories:

```
$ find .
./tp1301.txt
./up1301.txt
./tp1302.txt
./up1302.txt
./Up1303.txt
./misc/uploads
./misc/uploads/patch12_13.diff
```

As always, the dot indicates the current directory. Here find has listed all files found in the current directory and its subdirectories.

If we only want to find files with 'up' at the start of their name, we use the '-name' argument.
So the following would be used:

```
$ find . -name up\*
./up1301.txt
./up1302.txt
./misc/uploads
```

find defaults to being case sensitive. If we want the find utility to locate the file 'Up1303.txt' we could either do '`find -name Up\*`' or use the iname argument instead of the name argument.

The wildcard character is escaped with a slash so BASH sends a literal asterisk to the find utility as an argument instead of performing filename expansion and passing any number of files in as arguments.
This 'gotcha' is important. Be aware of the characters which the shell attaches special meaning to.

Now we know there are files that should have their names in lowercase we can utilise find to get a list of files with names that aren't:

```
$ find -iname up\* -not -name up\*
```

Smooth Operator

find supports boolean algebra with the -and, -or and -not arguments. These are abbreviated as -a, -o and ! (which in bash must be escaped as \!) respectively. The and operator is mentioned here for completeness. Its presence is implied:

```
$ find . -iname david\*gray\*ogg -type f > david_gray.m3u
```

These operators are processed in the following order:

Parentheses
Use parentheses to force the order in which the operators are evaluated.

-not
Invert the result of the tested expression.

-and
E.g. ex1 -and ex2; the second expression isn't checked if the first evaluated to true

-or
E.g. ex1 -or ex2; as with -AND, the second expression isn't checked if the first evaluated to true

','
This is the list operator where unlike the '-AND' and '-OR' operators both expressions are evaluated. Read the '2 into 1 does go' section for more information.

The example in the Smooth Operator boxout creates an m3u playlist listing all ogg files that start 'David Gray' (and all case-permutations)

```
$ find . -iname david\ gray\*ogg -type f > david_gray.m3u
```

This will find any files called, in one way or the other, "david gray....ogg".

This is semantically equivalent to:

```
$ find . -iname david\ gray\*ogg -and -type f > david_gray.m3u
```

It's equivalent to:

```
$ find . -iname "david gray*ogg" -and -type f > david_gray.m3u
```

What if the ogg files themselves mightn't have the artists name in them and are in some subdirectory of one called 'David Gray', how do we find them?

```
$ find . -ipath \*david\ gray\*ogg -type f > david_gray.m3u
```

The expression starts with a wildcard because its possible there's more than one subdirectory named 'david gray' that might really be nothing more than symlinks for categorisations.

Here's another example, we list the contents of the humour directory (one line per file) and do a case-insensitive search for .mp3 files with 'yoda' in the name of the file:

```
$ ls humour -1
Weird Al - Yoda.mp3
welcome_to_the_internet_helpdesk.mp3
werid al - livin' la vida yoda.mp3

$ find -ipath \*humour\*yoda\* -type f
./humour/Weird Al - Yoda.mp3
./humour/werid al - livin' la vida yoda.mp3
```

2 into 1 does go

As implied in the Smooth Operator boxout, it's possible to have one invocation of find perform more than one task.

To compile two lists, one containing the names of all .php files and the other the names of all .js files use:

```
$ find ~ -type f \( -name \*.php -fprint php_files ,
                    -name \*.js -fprint javascript_files \)
```

Pruning

Suppose you have a playlist file listing all David Gray .ogg files but there are a few albums you don't want included. You can prevent those albums from going into the playlist by using the -prune action which works by attempting to match the names of directories against the given expression.
This example excludes the Flesh and Lost Songs albums :

```
$ find   \( -path  ./mp3/David_Gray/Flesh\* -o -path
"./mp3/David_Gray/Lost Songs" \* \) -prune -o -ipath \*david\ gray\*
```

The first thing you'll notice here is the parentheses are escaped out so BASH doesn't misinterpret them. Notice using -prune takes the form

"don't look for these, look for these other ones instead". ie:

```
$ find (-path <don't want this> -o -path <don't want this#2>)
\-prune -o -path <global expression for what I do want>
```

It might take a bit longer to invoke find to use the -prune action: decide exactly what you want to do first. I find using the -prune action saves me time I can use on other tasks.

Fussy Fozzy!

There's a host of other expressions and criteria that can be used with find.

Here is a brief rundown on the ones you'll most likely want to use:

-nouser	file is owned by someone no longer listed in /etc/passwd
-nogroup	the group the file belongs to is no longer listed in /etc/groups
-owner <username>	file is owned by specified user.

We'll delve into using these, and others, later on.

Print me the way you want me, baby!

Changing the output information

If you want more than just the names of the files displayed, find's -printf action lets you have just about any type of information displayed. Looking at the man page there is a startling array of options.

These are used the most:

%p	filename, including name(s) of directory the file is in
%m	permissions of file, displayed in octal.
%f	displays the filename, no directory names are included
%g	name of the group the file belongs to.
%h	display name of directory file is in, filename isn't included.
%u	username of the owner of the file

As an example:

```
$ find . -name \*.ogg -printf %f\\n
```

generates a list of the filenames of all .ogg files in and under the current directory.
The 'double backslash n' is important; '\n' indicates the start of a new line. The single backslash needs to be escaped by another one so the shell doesn't take it as one of its own.

Where to output information?

find has a set of actions that tell it to write the information to any file you wish. These are the -fprint, -fprint0 and -fprintf actions.

Thus

```
$ find . -iname david\ gray\*ogg -type f -fprint david_gray.m3u
```

is more efficient than

```
$ find . -iname david\ gray\*ogg -type f > david_gray.m3u
```

Execute!

File is an excellent tool for generating reports on basic information regarding files, but what if you want more than just reports? You could just pipe the output to some other utility:

```
$ find ~/oggs/ -iname \*.mp3 | xargs rm
```

This isn't all that efficient though.
It is much better to use the -exec action:

```
$ find ~/oggs/ -iname \*.mp3 -exec rm {} \;
```

It mightn't read as well, but it does mean the files are immediately deleted once found.
'{}' is a placeholder for the name of the file that has been found and as we want BASH to ignore the semicolon and pass it verbatim to find we have to escape it.

To be cautious, the -ok action can be used instead of -exec. The -ok action means you'll be asked for confirmation before the command is executed.

There are many ways these can be used in 'real life' situations:
If you are locked out from the default Mozilla profile, this will unlock you:

```
$ find ~/.mozilla -name lock -exec rm {} \;
```

To compress .log files on an individual basis:

```
$ find . -name \*.log -exec bzip {} \;
```

Give user ken ownership of files that aren't owned by any current user:

```
$ find . -nouser -exec chown ken {} \;
```

View all .dat files that are in the current directory with vim. Don't search any subdirectories.

```
$ vim -R `find . -name \*.dat -maxdepth 1`
```

Look for directories called CVS which are at least four levels below the current directory:

```
$ find -mindepth 4 -type d -name CVS
```

Time waits for no-one

You might want to search for recently created files, or grep through the last 3 days worth of log files.

Find comes into its own here: it can limit the scope of the files found according to timestamps.

Now, suppose you want to see what hidden files in your home directory changed in the last 5 days:

```
$ find ~ -mtime -5 -name \.\*
```

If you know something has changed much more recently than that, say in the last 14 minutes, and want to know what it was there's the mmin argument:

```
$ find ~ -mmin 14 -name \.\*
```

Be aware that doing a 'ls' will affect the access time-stamps of the files shown by that action. If you do an ls to see what's in a directory and try the above to see what files were accessed in the last 14 minutes all files will be listed by find.

To locate files that have been modified since some arbitrary date use this little trick:

```
$ touch -d "13 may 2001 17:54:19" date_marker
$ find . -newer date_marker
```

To find files created before that date, use the cnewer and negation conditions:

```
$ find . \! -cnewer date_marker
```

To find a file which was modified yesterday, but less than 24 hours ago:

```
$ find . -daystart -atime 1 -maxdepth
```

The -daystart argument means the day starts at the actual beginning of the day, not 24 hours ago.
This argument has meaning for the -amin, -atime, -cmin, ctime, -mmin and -mtime options.

Finding files of a specific size

A file of character (bytes)

To locate files that have a certain amount of characters present then you can't go far wrong with

```
# find files with exactly 1000 characters
```

```
$ find . -size 1000c
#find files containing between 600 to 700 characters, inclusive.
$ find . -size +599c -and -size -701c
```

'Characters' is a misnomer: 'c' is find's shorthand for bytes; thus this will only work for ASCII text not Unicode.

Consulting the man page we see
c = bytes
w = 2 byte words
k = kilobytes
b = 512-byte blocks

Thus we can use find to list files of a certain size:

```
$ find /usr/bin -size 48k
```

Empty files

You can find empty files with `$ find . -size 0c`
Using the `-empty` argument is more efficient.

To delete empty files in the current directory:

```
$ find . -empty -maxdepth 1 -exec rm {} \;
```

Users & Groupies

Users

To locate files belonging to a certain user:

```
# find /etc -type f \!  -user root -exec ls -l {} \;
-rw------- 1 lp sys 19731 2002-08-23 15:04 /etc/cups/cupsd.conf
-rw------- 1 lp sys    97 2002-07-26 23:38 /etc/cups/printers.conf
```

A subset of that same information, without having the cost of an exec:

```
root@ttyp0[etc]# find /etc -type f \!  -user root \
                 -printf "%h/%f %u\\n"
/etc/cups/cupsd.conf lp
/etc/cups/printers.conf lp
```

If you know the uid and not the username then use the -uid argument:

```
$ find /usr/local/htdocs/www.linux.ie/ -uid 401
```

`-nouser` means there is no user in the /etc/passwd file for the files in question.

Groupies

find can locate files that belong to a specific group - or not, depending on how you use it.
This is especially suited to tracking down files that should belong to the www group but don't:

```
$ find /www/ilug/htdocs/  -type f \! -group  www
```

The -nogroup argument means there is no group in the /etc/group file for the files in question.
This may arise if a group is removed from the /etc/group file sometime after it's been used.
To search for files by the numerical group ID use the -gid argument:

```
$ find -gid 100
```

Permissions

If you've ever had one or more shell scripts not work because their execute bits weren't set and want to sort things out for once and for all, then you should like this little example:

```
knoppix@ttyp1[bin]$ ls -l ~/bin/
total 8
-rwxr-xr-x    1 knoppix  knoppix        21 2004-01-20 21:42 wl
-rw-r--r--    1 knoppix  knoppix        21 2004-01-20 21:47 ww

knoppix@ttyp1[bin]$ find ~/bin/ -maxdepth 1 -perm 644 -type f \
                    -not -name .\*
/home/knoppix/bin/ww
```

Find locates the file that isn't set to execute, as we can see from the output of ls.

Types of files

The '-type' argument obviously specifies what type of file find is to go looking for (remember in Linux absolutely everything is represented as some type of file).
So far I've been using '-type f' which means search for normal files.

If we want to locate directories with '_of_' in their name we'd use:

```
$ find . -type d -name '*_of_*'
```

The list generated by this won't include symbolic links to directories.
To get a list including directories and symbolic links:

```
$ find . \( -type d -or -type l \) -name '*_of_*'
```

For a complete list of types check the man page.

Regular expressions

Thus far we've been using casual wildcards to specify certain groups of files. Find also support regular expressions, so we can use more advanced criteria with regards to locating files. The matching expression must apply to the entire path:

```
ken@gemmell:/home/library/code$ find . -regex '.*/mp[0-4].*'
./library/sql/mp3_genre_types.sql
```

The -regex test has a case insensitive counterpart, -iregex.

There is a little gotcha with using regular expressions: You must allow for the full path of the files found, even if find is to search the current directory:

```
$ cd /usr/share/doc/samba-doc/htmldocs/using_samba
$ find . -regex './ch0[1-2]_0[1-3].*'
./ch01_01.html
./ch01_02.html
./ch02_01.html
./ch02_02.html
./ch02_03.html
```

Limiting by filesytem

As an experiment, get a MS formatted floppy disk and mount it as root:

```
$ su -
# mount /floppy
# mount
/dev/sda2 on / type ext2 (rw,errors=remount-ro)
proc on /proc type proc (rw)
devpts on /dev/pts type devpts (rw,gid=5,mode=620)
/dev/fd0 on /floppy type msdos (rw,noexec,nosuid,nodev)
```

Now try

```
$ find / -fstype msdos -maxdepth 1
```

You should see only /floppy listed.
To get the reverse of this, ie a listing of directories that are not on msdos file-systems, use

```
$ find / -maxdepth 1 \( -fstype msdos \) -prune -or -print
```

This is a start on limiting the files found by system type.

Summary

I've covered the vast majority of ways to use the find utility, but not absolutely everything. If you've any questions please don't hesitate in emailing me.

About the author, Ken Guest.

USERS COMMENTS
Posted By Ken Guest 12:07, 30 June 2006
In general it's best to use $touch -t rather than $touch -d. The find tool on FreeBSD doesn't support the -d argument, so you would have to do: $ touch -t "200606261500" marker instead of $ touch -d "26 jun 2006 15:00" marker

Posted By Bruce Williamson 11:09, 2 February 2010
Howzit Ken? I just want to make a comment regarding the finder-keepers section, slash (/) implies a forward slash see http://www.wsu.edu/~brians/errors/backslash.html "The wildcard character is escaped with a slash" This should actually read "The wildcard character is escaped with a backslash" Cheers Bruce

Add a note