

*May the source be with you,
but remember the KISS principle ;-)*

Softpanorama: (slightly skeptical) Open Source Software Educational Society



Web www.softpanorama.org

Unix Find Command Tutorial

- [Introduction](#)
- [Using -exec option with find](#)
- [Feeding find output to pipes with xargs](#)
- [SUID/SGUID games, abandoned and other abnormal files](#)
- [Summary](#)
- [Webliography](#)

Introduction

Unix find is a pretty tricky command that can often fool even experienced UNIX professionals with ten or more years of sysadmins work under the belt.

The idea is extremely simple: this is a utility for *searching files using the directory information* and in this sense it is more like **ls**. But it is more obscure than ls and has a special unique mini-language for specifying query, the language that long outlived its usefulness but nobody has a courage to replace it with a standard scripting language.

For obscure historical reasons **find** has a notation that is completely different from all other UNIX commands: it has full-word options rather than single-letter options. For example, instead of a typical Unix-style option `-n` to match filenames **find** uses option `-name`.

In general you need to specify the starting point for a search through the file system as the first argument, followed by any actions desired. The list of popular examples can be found in Reference section of this page.

It is simply impossible to remember all the details of this language unless you are need to use complex queries each day multiple times and that's why this page was created.

There are some interesting options in Posix find (as described in [Solaris man page](#))

[Unix Tutorial](#)

Solutions for Your Small Business Business Begins Here.
www.business.com

[Porting UNIX to Linux?](#)

Free whitepaper is packed with advice on moving X/Motif apps.
www.ics.com

Web by Google

- **-fstype type** True if the filesystem to which the file belongs is of type *type*. For example on Solaris mounted local filesystems have type ufs (Solaris 10 added zfs). For AIX local filesystem is **jfs** or **jfs2** (journaled file system). If you want to traverse NFS filesystems you can use **nfs** (network file system).
- **"-atime/-ctime/-mtime"** the last time a file's "access time", "file status" and "modification time", measured in days or minutes. The time can be compared to another file with "-newer/-anewer/-cnewer".
 - Example: find everything in your home directory modified in the last 24 hours:
 - **find \$HOME -mtime 0**
 - Example: find everything in your home directory modified in the last 7 days:
 - **find \$HOME -mtime -7**
 - Example: find everything in your home directory that have NOT been modified in the last year:
 - **find \$HOME -mtime +365**
 - Example: find everything in your home that has been modified more recently than "abc.txt":
 - **find \$HOME -newer ~joeuser/lastbatch.txt**
- **-local** True if the file system type is not a remote file system type as defined in the `/etc/dfs/fstypes` file. **nfs** is used as the default remote filesystem type if the `/etc/dfs/fstypes` file is not present. The **-local** option descends the hierarchy of non-local directories. See EXAMPLES for an example of how to search for local files without descending.
- **-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.

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 sign, then it means *"more than n days ago", or older than n,*
- **-n:** if it has the minus sign, then it means *less than n days ago (-n), or younger than 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 **AND**ed
- **o** to have multiple conditions **OR**ed
- **!** to negate a condition
- **expression** to satisfy any complex condition

What is really important is that it is possible (and even easy) 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); this list can be piped to the script for postprocessing
- **exec command** executes the specified command. This is more suitable for doing simple things. For more complex things post processing of output is a safer option as you have some additional context to make the particular decision.

The most common reason for using the **find** command is to utilize its capability to recursively process the subdirectories. 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
```

You can also execute the **find** command with multiple conditions. 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 -a -mtime 0 -print
```

Option	Meaning	Example
-atime n -atime +n -atime -n	True if file was accessed n days ago (n), accessed more then n days ago(+n) or less than n days ago (-n)	
-ctime n	True if the file was created n days ago.	<code>find . -ctime +30 -type f -exec rm {} !;</code>
-exec command	Execute command.	<code>find . -mtime -2 -type f -exec mv {} ../Spam_collector \;</code>
-mtime n	True if file was modified n days ago.	<code>find . -mtime -2 -type f -exec mv {} ../Spam_collector \;</code>
-name pattern	True if filename matches pattern.	
-print	Print names of files found.	
-type c	True if file is of type c	<code>find . -mtime -2 -type f -exec mv {} ../Spam_collector \;</code>
-user name	True if file is owned by user name.	

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
```

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

```
find . -name "*.html: -print
```

You 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, I can use -mtime with the argument -7 (include the hyphen):

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

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

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

To find those html files that I haven't touched for at least 7 days, I use +7:

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

1. You 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`, I can use the following command:

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

2. 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

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. You need to remember two tricks:

1. the command that you want to execute need to contain a special (obscure) argument `{}`, which will be replaced by the matched filename,
2. 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 `{}` 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.

How to apply a unix command to a set of file (-exec).

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

This command will search in the current directory and all sub directories. All files named rc.conf will be processed by the chmod -o+r command. The argument '{}' inserts each found file into the chmod command line. The \; argument indicates the exec command line has ended.

The end results of this command is all rc.conf files have the other permissions set to read access (if the operator is the owner of the file).

The find command is commonly used to remove core files that are more than a few days old. These core files are copies of the actual memory image of a running program when the program dies unexpectedly. They can be huge, so occasionally trimming them is wise:

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

There's no output from this command because I 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 you 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.

Feeding find output to pipes with xargs

One of the biggest limitations of the -exec command is that it can only run the specified command on one file at a time. The xargs command solves this problem by enabling users to run a single command on many files at one time. In general, it is much faster to run one command on many files, because this cuts down on the number of commands that need to be started.

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 -l -i '/bin/ksh' {} \;
```

But there is more elegant and more Unix-like way of accomplishing the same task 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 -l -i '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 when grep shows the matching line. Removing the -l flag results in more meaningful (and potentially more useful output):

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

When used in combination, find, grep, and xargs are a potent team to help find files lost or misplaced anywhere in the UNIX file system. I encourage you to experiment further with these important commands to find ways they can help you work with UNIX. You can use time to find the difference in speed with -exec option:

```
time find /usr/src -name "*.html" -exec grep -l foo '{}' ';' | wc -l
```

```
time find /usr/src -name "*.html" | xargs grep -l foo | wc -l
```

xargs works considerably faster. The difference becomes even greater when more complex commands are run and the list of files is longer.

```
find /mnt/zip -name "*prefs copy" -print | xargs rm
```

This won't work because I have a filename with spaces. If I add -print0, I can do it with no problems:

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

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

Some people wonder why there is a -p option. xargs runs the specified command on the filenames from its standard input, so interactive commands such as cp -i, mv -i, and rm -i don't work right. The -p option solves that problem. In the preceding example, the -p option would have made the command safe because I could answer yes or no to each file. Thus, the command I typed was the following:

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

Many users frequently ask why **xargs** should be used when shell command substitution achieves the same results. Take a look at this example:

```
grep -l foo `find /usr/src/linux -name "*.html"`
```

The drawback with commands such as this is that if the set of files returned by **find** is longer than the system's command-line length limit, the command will fail. The **xargs** approach gets around this problem because **xargs** runs the command as many times as is required, instead of just once.

SUID/SGUID games, finding abandoned and other abnormal files

Suid root refers to a special attribute called set user id. This allows the program to do functions not normally allowed for users to do themselves. Low level networking routines, controlling graphical display functions, changing passwords, and logging in are all examples of programs that rely on executing their functions as a user that is not restricted by standard file permissions. While many programs need this functionality, the program must be bug free in only allowing the user to do the function the program was designed for. Every suid root program represents a potential security problem.

The first step in controlling suid root programs is to have a baseline, the list of all suid program in the system. This can be achieved quite easily by using find:

```
find / -type f -perm +6000 -exec ls -l {} \; > suid.list
(note: this will find both set user id and set group id programs)
```

Above command is using GNU find. Solaris find command is slightly different:

```
find / -type f \( -perm -4000 -o -perm -2000 \) -exec ls -l {} \;
```

This command will find all the suid programs on a system and pipes the commands to a file called suid.list. The next step in controlling suid root programs is to analyze which programs should not be suid root or can be removed without impeding system functionality. An obvious example of something that should not be suid root is **/usr/X11R6/bin/SuperProbe**. This is a program merely used for testing purposes.

```
'chmod -s /usr/X11R6/bin/SuperProbe'
```

Other programs that are unneeded to be suid root include anything in the svgalib hierarchy. This library itself is buggy and

nothing that depends on it should be `sudo` root in a secure system.

Here is a [suid.list](#) from a much cleaner system, though perhaps a little too overzealous. The functionality that does not exist with this setup is ping and traceroute. Everything else is possible and keeping up with changes in utilities is easy with such a small list to look at.

The main theory that is common with `sudo` root programs is only letting what needs to be available to users be `sudo` root. Know what each program does and the reason it needs to be `sudo` root. Also know the versions of `sudo` root programs and upgrade them as new versions come out, making sure to keep track of your vendor's OS fixes.

Summary

The `find` command has a lot of esoteric options, and to get the full power out of `find`, `xargs`, and `grep`, you need to experiment. You can specify:

- 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)
- Matching criteria
 - -atime n file was accessed n days ago
 - -mtime n file was modified n days ago
 - -size n file is exactly n 512-byte blocks
 - -type c file type (e.g., f=plain, d=dir)
 - -name nam file name (e.g., `*.c')
 - -user usr file's owner is usr
 - -perm p file's access mode is p
- Actions
 - -print display pathname
 - -exec cmd execute command ({} expands to file)

There are several options for matching criteria:

```
-atime n      File was accessed "n" days ago
-mtime n      File was modified "n" days ago
-size n       File is "n" 512-byte blocks big
-type c       Specifies file type: f=plain text, d=directory
-fstype typ   Specifies filesystem type: 4.2 or nfs
-name nam     The filename is "nam"
-user usr     The file's owner is "usr"
-group grp    The file's group owner is "grp"
-perm p       The file's access mode is "p" (integer)
```

You can use plus + and minus - modifiers with the "atime", "mtime", and "size" criteria to increase their usefulness. Some examples:

```
-mtime +7     Matches files modified more than 7 days ago
-atime -2     Matches files accessed less than 2 days ago
-size +100    Matches files larger than 100 blocks (50K)
```

Multiple options are joined by AND by default. OR may be specified with the `-o` flag and the use of grouped parentheses. To match all files modified more than 7 days ago and accessed more than 30 days ago, use:

```
\( -mtime +7 -o -atime +30 \)
```

NOT may be specified with an exclamation point. To match all files ending in .txt except the file "notme.txt", use:

```
\! -name notme.txt -name \*.txt
```

You can specify the following actions for the list of files that the "find" command locates:

```
-print      Display pathnames of matching files
-exec cmd   Execute command "cmd" on a file
-ok cmd     Prompt before executing command "cmd" on a file
-mount      (System V) Restrict to filesystem of starting dir.
-xdev       (BSD) Restrict to filesystem of starting dir.
-prune      (BSD) Don't descend into subdirectories
```

Executed commands must end with \; or ';' and may use {} as a placeholder for each file found by "find". For example, for a long listing of each file found, use:

```
-exec ls -l {} \;
```

Matching criteria and actions may be placed in any order, and are evaluated from left to right.

Webliography

- [Softpanorama Unix Find Page](#)
- [Find - Wikipedia, the free encyclopedia](#)
- [Advanced techniques for using the UNIX find command](#)
- [Finding Files - Table of Contents](#) by David MacKenzie Edition 1.1, for GNU **find** version 4.1 November 1994
- [Find tutorial](#) Copyright 2001 Bruce Barnett and General Electric Company
- [O'Reilly Network / Finding Things in Unix](#)
- One of the most useful utilities to be found on any Unix system is the **find** command. In the next two articles, I'd like to work you through the syntax of this command and provide you with some practical examples of its usage.

[ONLamp.com: Find: Part Two \[Mar. 14, 2002\]](#)

- About.com: [Power Commands- Find](#)
- [How do I use the solaris find command - ECN Knowledge Base @ Purdue](#)
- [The Shell Corner Using the Unix find Command](#)
- [Some examples of using Unix find command.](#)
- [find examples](#)
- [Linux and UNIX find command help](#)
- [Learn Unix The find command](#)

Need Sun Support?

Superior Sun Support Plans Multiple Support Options! 24-7
www.Terix.com

Online IT Courses

Check Out Our Online UNIX Program Move Ahead in Life-Request Info Now
www.UMassOnline.net

Introduction to unix

Download Free IT White Papers about Unix, Windows & Other Platforms
www.FindWhitePapers.com

Unix Resume

Don't just search for a job find
www.Dice.com

by Google

Copyright © 1996-2007 by Dr. Nikolai Bezroukov. www.softpanorama.org was created as a service to the UN Sustainable Development Networking Programme ([SDNP](#)) in the author free time. [Submit comments](#) This

document is an industrial compilation designed and **created exclusively for educational use** and is placed under the copyright of the [Open Content License\(OPL\)](#). Original materials copyright belong to respective owners. **Quotes are made for educational purposes only in compliance with the fair use doctrine.**

Standard disclaimer: *The statements, views and opinions presented on this web page are those of the author and are not endorsed by, nor do they necessarily reflect, the opinions of the author present and former employers, SDNP or any other organization the author may be associated with. We do not warrant the correctness of the information provided or its fitness for any purpose.*

Created: May 16, 1997; Last modified: May 21, 2007