

---

## Experiment 11

**Aim: Searching:** Search file or directory in directory structure using find and locate command with various options, wildcards \*, ?, [], !

### Find Command in Linux

The Linux Find Command is one of the most important and frequently used command command-line utility in Unix-like operating systems. Find command is used to search and locate the list of files and directories based on conditions you specify for files that match the arguments. Find can be used in a variety of conditions like you can find files by permissions, users, groups, file type, date, size, and other possible criteria.

### The Easiest Way To Find A File

The command used to search for files is called **find**.

Here is the basic syntax of the **find** command:

```
find
```

The starting point is the folder where you want to start searching from. To start searching the whole drive you would type the following:

```
find /
```

If, however, you want to start searching for the folder you are currently in then you can use the following syntax:

```
find .
```

When you search by name across the whole drive, use the following syntax:

```
find / -name filename
```

- The first part of the find command is obviously the word find.
- The second part is where to start searching from.
- The next part is an expression which determines what to find.
- Finally the last part is the name of the thing to find.

### Search Location Shortcuts

The first argument after **find** is the location you wish to search. Although you may specify a specific directory, most people use a metacharacter to serve as a substitute. The three metacharacters that work with this command include:

- **Period:** specifies the current and all nested folders
- **Forward Slash:** specifies the entire filesystem
- **Tilde:** specifies the active user's home directory

There are five parts from basic to advance usage of the find command.

1. Part I: Basic Find Commands for Finding Files with Names
2. Part II: Find Files Based on their Permissions
3. Part III: Search Files Based On Owners and Groups
4. Part IV: Find Files and Directories Based on Date and Time
5. Part V: Find Files and Directories Based on Size
6. Part VI: Find Multiple Filenames in Linux

## Part I – Basic Find Commands for Finding Files with Names

### 1. Find Files Using Name in Current Directory

Find all the files whose name is tecmint.txt in a current working directory.

```
# find . -name tecmint.txt
```

```
./tecmint.txt
```

### 2. Find Files Under Home Directory

Find all the files under /home directory with name tecmint.txt.

```
# find /home -name tecmint.txt
```

```
/home/tecmint.txt
```

### 3. Find Files Using Name and Ignoring Case

Find all the files whose name is tecmint.txt and contains both capital and small letters in /home directory.

```
# find /home -iname tecmint.txt
```

```
./tecmint.txt
```

```
./Tecmint.txt
```

### 4. Find Directories Using Name

Find all directories whose name is Tecmint in / directory.

```
# find / -type d -name Tecmint
```

```
/Tecmint
```

#### 5. Find PHP Files Using Name

Find all php files whose name is tecmint.php in a current working directory.

```
# find . -type f -name tecmint.php
```

```
./tecmint.php
```

#### 6. Find all PHP Files in Directory

```
# find . -type f -name "*.php"
```

```
./tecmint.php
```

```
./login.php
```

```
./index.php
```

### Part II – Find Files Based on their Permissions

#### 7. Find Files With 777 Permissions

```
# find . -type f -perm 0777 -print
```

#### 8. Find Files Without 777 Permissions

```
# find / -type f ! -perm 777
```

#### 9. Find Read Only Files

```
# find / -perm /u=r
```

```
$ find /etc -maxdepth 1 -perm /u=r
```

```
/etc
```

```
/etc/thunderbird
```

```
/etc/brltty
```

```
/etc/dkms  
/etc/phpmyadmin  
... output truncated ...
```

#### 10. Find Executable Files

```
# find / -perm /a=x
```

```
$ find /bin -maxdepth 2 -perm /a=x  
/bin  
/bin/preseed_command  
/bin/mount  
/bin/zfgrep  
/bin/tempfile  
... output truncated ...
```

#### 11. Find Files with 777 Permissions and Chmod to 644

Find all 777 permission files and use chmod command to set permissions to 644.

```
# find / -type f -perm 0777 -print -exec chmod 644 {} \;
```

#### 12. Find Directories with 777 Permissions and Chmod to 755

Find all 777 permission directories and use chmod command to set permissions to 755.

```
# find / -type d -perm 777 -print -exec chmod 755 {} \;
```

#### 13. Find and remove single File

To find a single file called tecmint.txt and remove it.

```
# find . -type f -name "tecmint.txt" -exec rm -f {} \;
```

#### 14. Find and remove Multiple File

To find and remove multiple files such as .mp3 or .txt, then use.

```
# find . -type f -name "*.txt" -exec rm -f {} \;
```

OR

```
# find . -type f -name "*.mp3" -exec rm -f {} \;
```

#### 15. Find all Empty Files

To find all empty files under certain path.

```
# find /tmp -type f -empty
```

#### 16. Find all Empty Directories

To file all empty directories under certain path.

```
# find /tmp -type d -empty
```

#### 17. File all Hidden Files

```
# find /tmp -type f -name ".*"
```

### Part III – Search Files Based On Owners and Groups

#### 18. Find Single File Based on User

To find all or single file called tecmint.txt under / root directory of owner root.

```
# find / -user root -name tecmint.txt
```

#### 19. Find all Files Based on User

To find all files that belongs to user Tecmint under /home directory.

```
# find /home -user tecmint
```

#### 20. Find all Files Based on Group

To find all files that belongs to group Developer under /home directory.

```
# find /home -group developer
```

#### 21. Find Particular Files of User

To find all .txt files of user Tecmint under /home directory.

```
# find /home -user tecmint -iname "*.txt"
```

### Part IV – Find Files and Directories Based on Date and Time

#### 22. Find Last 50 Days Modified Files

To find all the files which are modified 50 days back.

```
# find / -mtime 50
```

#### 23. Find Last 50 Days Accessed Files

To find all the files which are accessed 50 days back.

```
# find / -atime 50
```

#### 24. Find Last 50-100 Days Modified Files

To find all the files which are modified more than 50 days back and less than 100 days.

```
# find / -mtime +50 -mtime -100
```

#### 25. Find Changed Files in Last 1 Hour

To find all the files which are changed in last 1 hour.

```
# find / -cmin -60
```

#### 26. Find Modified Files in Last 1 Hour

To find all the files which are modified in last 1 hour.

```
# find / -mmin -60
```

27. Find Accessed Files in Last 1 Hour

To find all the files which are accessed in last 1 hour.

```
# find / -amin -60
```

Part V – Find Files and Directories Based on Size28. Find 50MB Files

To find all 50MB files, use.

```
# find / -size 50M
```

29. Find Size between 50MB – 100MB

To find all the files which are greater than 50MB and less than 100MB.

```
# find / -size +50M -size -100M
```

30. Find and Delete 100MB Files

To find all 100MB files and delete them using one single command.

```
# find / -size +100M -exec rm -rf {} \;
```

35. Find Specific Files and Delete

Find all .mp3 files with more than 10MB and delete them using one single command.

```
# find / -type f -name *.mp3 -size +10M -exec rm {} \;
```

**36. How to Send Output from the Find Command to a File**

The main problem with the find command is that it can sometimes return too many results to look at in one go. you can output the lines to a file as follows:

```
find / -name *.mp3 -fprint nameoffiletoprintto
```

**How to Find and Execute a Command against a File**

To search for and edit a file at the same time:

```
find / -name filename -exec nano '{}' \;
```

The above command searches for a file called filename and then run the nano editor for the file that it finds.

### **Let see some practical exercises on using find command.**

#### **1. How to run the last executed find command?**

```
!find
```

This will execute the last find command. It also displays the last find command executed along with the result on the terminal.

#### **2. How to find the files whose name is not "sum.java"?**

```
find -not -name "sum.java"
```

```
./SUM.java  
./bkp  
./multiply.java
```

This is like inverting the match. It prints all the files except the given file "sum.java".

#### **Invert match**

It is also possible to search for files that do not match a given name or pattern. This is helpful when we know which files to exclude from the search.

```
$ find ./test -not -name "*.php"  
./test  
./test/abc.txt  
./test/subdir
```

So in the above example we found all files that do not have the extension of php, either non-php files. The find command also supports the exclamation mark in place of not.

```
find ./test ! -name "*.php"
```

#### **3. How to limit the file searches to specific directories?**



## Limit depth of directory traversal

The find command by default travels down the entire directory tree recursively, which is time and resource consuming. However the depth of directory traversal can be specified. For example we don't want to go more than 2 or 3 levels down in the sub directories. This is done using the maxdepth option.

```
$ find ./test -maxdepth 2 -name "*.php"
./test/subdir/how.php
./test/cool.php

$ find ./test -maxdepth 1 -name *.php
./test/cool.php
```

The second example uses maxdepth of 1, which means it will not go lower than 1 level deep, either only in the current directory.

This is very useful when we want to do a limited search only in the current directory or max 1 level deep sub directories and not the entire directory tree which would take more time.

Just like maxdepth there is an option called mindepth which does what the name suggests, that is, it will go atleast N level deep before searching for the files.

```
find -name "sum.java"
./tmp/sum.java
./bkp/var/tmp/files/sum.java
./bkp/var/tmp/sum.java
./bkp/var/sum.java
./bkp/sum.java
./sum.java
```

You can see here the find command displayed all the files with name "sum.java" in the current directory and sub-directories.

### 4. How to find for a file in the current directory only?

```
find -maxdepth 1 -name "sum.java"
./sum.java
```

This will find for the file "sum.java" in the current directory only

5. How to print the files in the current directory and one level down to the current directory?

```
find -maxdepth 2 -name "sum.java"
/tmp/sum.java
./bkp/sum.java
./sum.java
```

6. How to print the files in the current directory and two levels down to the current directory?

```
find -maxdepth 3 -name "sum.java"
/tmp/sum.java
./bkp/var/sum.java
./bkp/sum.java
./sum.java
```

7. How to print the files in the subdirectories between level 1 and 4?

```
find -mindepth 2 -maxdepth 5 -name "sum.java"
/tmp/sum.java
./bkp/var/tmp/files/sum.java
./bkp/var/tmp/sum.java
./bkp/var/sum.java
./bkp/sum.java
```

8. How to find the empty files in a directory?

```
find . -maxdepth 1 -empty
./empty_file
```

9. How to find the largest file in the current directory and sub directories

```
find . -type f -exec ls -s {} \; | sort -n -r | head -1
```

The find command "find . -type f -exec ls -s {} \;" will list all the files along with the size of the file. Then the sort command will sort the files based on the size. The head command will pick only the first line from the output of sort.

**10.** How to find the smallest file in the current directory and sub directories

```
find . -type f -exec ls -s {} \; | sort -n -r | tail -1
```

Another method using find is

```
find . -type f -exec ls -s {} \; | sort -n | head -1
```

**11.** How to find files based on the file type?

**a.** Finding socket files

```
find . -type s
```

**b.** Finding directories

```
find . -type d
```

**c.** Finding hidden directories

```
find -type d -name ".*"
```

**d.** Finding regular files

```
find . -type f
```

**e.** Finding hidden files

```
find . -type f -name ".*"
```

**12.** How to find the files which are modified after the modification of a give file.

```
find -newer "sum.java"
```

This will display all the files which are modified after the file "sum.java"

**13.** Display the files which are accessed after the modification of a give file.

```
find -anewer "sum.java"
```

**14.** Display the files which are changed after the modification of a give file.

```
find -cnewer "sum.java"
```

**15.** How to find the files which are modified 30 minutes back

```
find . -not -mmin -30
```

**16.** How to find the files which are modified 1 day back.

```
find . -not -mtime -1
```

**17.** How to find the files which are created between two files.

```
find . -cnewer f1 -and ! -cnewer f2
```

So far we have just find the files and displayed on the terminal. Now we will see how to perform some operations on the files.

**18.** Find the files which have the name "java" in it and then display only the files which have "class" word in them?

```
find -name "*java*" -exec grep -H class {} \;
```

**19.** How to remove files which contain the name "java".

```
find -name "*java*" -exec rm -r {} \;
```



This will delete all the files which have the word “java” in the file name in the current directory and sub-directories.

## Part VI – Find multiple filenames in Linux

The simplest and general syntax of the find utility is as follows:

```
# find directory options [ expression ]
```

Let us proceed to look at some examples of find command in Linux.

1. Assuming that you want to find all files in the current directory with **.sh** and **.txt** file extensions, you can do this by running the command below:

```
# find . -type f \( -name "*.sh" -o -name "*.txt" \)
```

```
aaronkilik@tecMint ~/bin $ find . -type f \( -name "*.sh" -o -name "*.txt" \)
./examples.txt
./script.sh
./test.sh
./list.txt
aaronkilik@tecMint ~/bin $
```

Interpretation of the command above:

1. **.** means the current directory
2. **-type** option is used to specify file type and here, we are searching for regular files as represented by **f**
3. **-name** option is used to specify a search pattern in this case, the file extensions
4. **-o** means “OR”

It is recommended that you enclose the file extensions in a bracket, and also use the **\** (back slash) escape character as in the command.

2. To find three filenames with **.sh**, **.txt** and **.c** extensions, issues the command below:

```
# find . -type f \( -name "*.sh" -o -name "*.txt" -o -name "*.c" \)
```

```
aaronkilik@tecMint ~/bin $ find . -type f \( -name "*.sh" -o -name "*.txt" -o -name "*.c" \)
./examples.txt
./script.sh
./test.sh
./list.txt
./file.c
./header.c
./lost.c
aaronkilik@tecMint ~/bin $
```



3. Here is another example where we search for files with **.png**, **.jpg**, **.deb** and **.pdf** extensions:

```
# find /home/aaronkilik/Documents/ -type f \( -name "*.png" -o -name "*.jpg" -o -name "*.deb" -o -name "*.pdf" \)
```

```
aaronkilik@tecMint ~ $ find /home/aaronkilik/Documents/ -type f \( -name "*.png" -o -name
-o -name "*.deb" -o -name "*.pdf" \)
/home/aaronkilik/Documents/sudo.png
/home/aaronkilik/Documents/festival3.jpg
/home/aaronkilik/Documents/true.jpg
/home/aaronkilik/Documents/keyboard.jpg
/home/aaronkilik/Documents/programmers.jpg
/home/aaronkilik/Documents/own_business.jpg
/home/aaronkilik/Documents/HIM.jpg
/home/aaronkilik/Documents/life.jpg
/home/aaronkilik/Documents/cabling.png
/home/aaronkilik/Documents/Tecmint-News/ServerMania-Cloud-Hosting.jpg
/home/aaronkilik/Documents/f8M93fM.jpg
/home/aaronkilik/Documents/linux.jpg
/home/aaronkilik/Documents/children.jpg
/home/aaronkilik/Documents/dp.jpg
/home/aaronkilik/Documents/festival2.jpg
/home/aaronkilik/Documents/keyboard-shortcuts.jpg
/home/aaronkilik/Documents/festival.jpg
/home/aaronkilik/Documents/Tecmint.com/How to Upgrade To Linux Mint 18 /check-mintupgraded
/home/aaronkilik/Documents/Tecmint.com/How to Upgrade To Linux Mint 18 /download-complete
/home/aaronkilik/Documents/Tecmint.com/How to Upgrade To Linux Mint 18 /upgrade3.png
/home/aaronkilik/Documents/Tecmint.com/How to Upgrade To Linux Mint 18 /download-mintupgra
ges.png
/home/aaronkilik/Documents/Tecmint.com/How to Upgrade To Linux Mint 18 /check-unlimited-so
png
/home/aaronkilik/Documents/Tecmint.com/How to Upgrade To Linux Mint 18 /update-manager.png
/home/aaronkilik/Documents/Tecmint.com/How to Upgrade To Linux Mint 18 /package-configurat
```

When you critically observe all the commands above, the little trick is using the **-o** option in the find command, it enables you to add more filenames to the search array, and also knowing the filenames or file extensions you are searching for.

### Combine multiple search criterias

It is possible to use multiple criterias when specifying name and inverting. For example

```
$ find ./test -name 'abc*' ! -name '*.php'
./test/abc.txt
./test/abc
```

The above find command looks for files that begin with abc in their names and do not have a php extension. This is an example of how powerful search expressions can be build with the find command.

### **OR operator**

When using multiple name criterias, the find command would combine them with AND operator, which means that only those files which satisfy all criterias will be matched. However if we need to perform an OR based matching then the find command has the "o" switch.

```
$ find -name '*.php' -o -name '*.txt'  
./abc.txt  
./subdir/how.php  
./abc.php  
./cool.php
```

The above command search for files ending in either the php extension or the txt extension.

### **Search multiple directories together**

So lets say you want to search inside 2 separate directories. Again, the command is very simple

```
$ find ./test ./dir2 -type f -name "abc*"  
./test/abc.txt  
./dir2/abcdefg.txt
```

Check, that it listed files from 2 separate directories.

Did you know you could search your home directory by using the ~ symbol?

```
$ find ~ -name "hidden.php"
```

### **Some advanced operations**

The find command not only finds files based on a certain criteria, it can also act upon those files using any linux command. For example, we might want to delete some files.

### **Here are some quick examples**

**List out the found files**

Lets say we found files using find command, and now want to list them out as the ls command would have done. This is very easy.

```
$ find . -exec ls -ld {} \;  
drwxrwxr-x 4 enlightened enlightened 4096 Aug 11 19:01 .  
-rw-rw-r-- 1 enlightened enlightened 0 Aug 11 16:25 ./abc.txt  
drwxrwxr-x 2 enlightened enlightened 4096 Aug 11 16:48 ./abc  
drwxrwxr-x 2 enlightened enlightened 4096 Aug 11 16:26 ./subdir  
-rw-rw-r-- 1 enlightened enlightened 0 Aug 11 16:26 ./subdir/how.php  
-rw-rw-r-- 1 enlightened enlightened 29 Aug 11 19:13 ./abc.php  
-rw-rw-r-- 1 enlightened enlightened 0 Aug 11 16:25 ./cool.php
```

**Delete all matching files or directories**

The following command will remove all text files in the tmp directory.

```
$ find /tmp -type f -name "*.txt" -exec rm -f {} \;
```

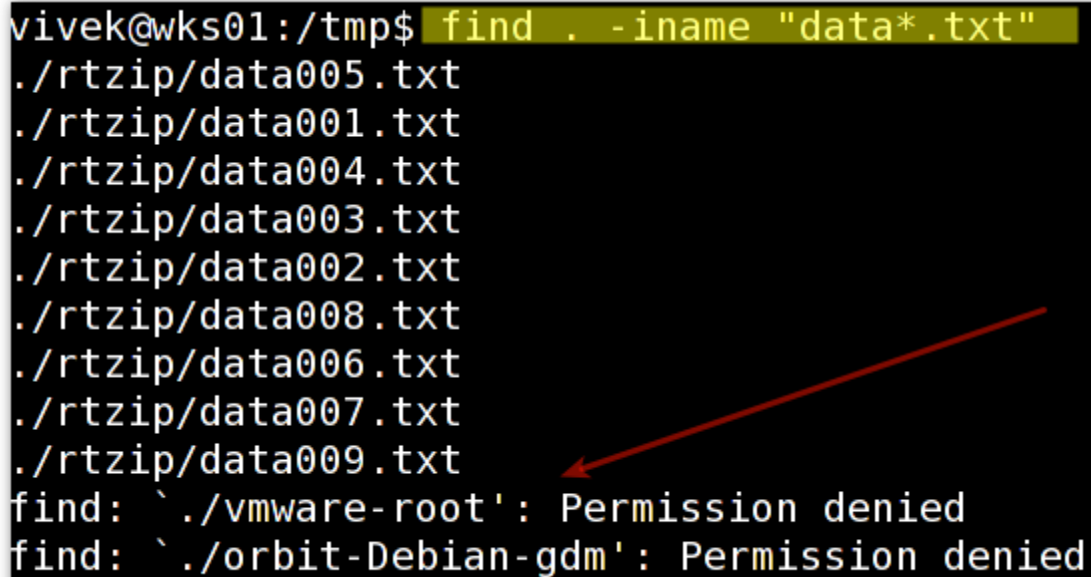
The same operating can be carried out with directories, just put type d, instead of type f.

Lets take another example where we want to delete files larger than 100MB

```
$ find /home/bob/dir -type f -name *.log -size +10M -exec rm -f {} \;
```



### Dealing with “Permission denied error messages” on Linux



```
vivek@wks01:/tmp$ find . -iname "data*.txt"
./rtzip/data005.txt
./rtzip/data001.txt
./rtzip/data004.txt
./rtzip/data003.txt
./rtzip/data002.txt
./rtzip/data008.txt
./rtzip/data006.txt
./rtzip/data007.txt
./rtzip/data009.txt
find: `./vmware-root': Permission denied
find: `./orbit-Debian-gdm': Permission denied
```

A terminal window showing the execution of the command `find . -iname "data*.txt"`. The output lists several files in the `./rtzip/` directory. Below these, two error messages are displayed: `find: `./vmware-root': Permission denied` and `find: `./orbit-Debian-gdm': Permission denied`. A red arrow points from the first error message to the second one.

Find will show an error message for each directory/file on which you don't have read permission. To avoid those messages, append `2>/dev/null` at the end of each command.

### Locate Command

One of the most common operations when working on Linux is to search for files and directories. On Linux systems, there are several commands which allow you to search for files with `find` and `locate` being the most used ones.

The `locate` command is the quickest and simplest way to search for files and directories by their names.

The `locate` command is often the simplest and quickest way to find the locations of files and directories on Linux and other Unix-like operating systems.

### Installing locate (locate command not found)

Depending on the distribution and on how the system was provisioned the `locate` package may or may not be pre-installed on your Linux system.

To check whether the `locate` utility is installed on your system, open up your terminal, type `locate`, and press Enter. If the package is installed the system will display `locate: no pattern to search for specified`, otherwise you will see something like `locate command not found`.

If locate is not installed you can easily install it using the package manager of your distro.

### **Install locate on Ubuntu and Debian**

```
sudo apt update
```

```
sudo apt install mlocate
```

### **How Locate Command Works? – updatedb and updatedb.conf**

The locate command searches for a given pattern through a database file that is generated by the updatedb command. The found results are displayed on the screen, one per line.

During the installation of the mlocate package, a cron job is created that runs the updatedb command every 24 hours. This ensures the database is regularly updated. For more information about the cron job check the /etc/cron.daily/mlocate file.

The database can be manually updated by running updatedb command as root or user with sudo privileges:

```
sudo updatedb
```

The update process will take some time, depending on the number of files and directories and the speed of your system.

Files created after the database update will not be shown in the locate results.

Compared to the more powerful find command that searches the file system, locate operates much faster but lacks many features and can search only by the file name.

**Q1. When we say that locate searches very quickly, then the first question that comes into mind is that what makes locate so fast?**

Well, locate does not search the files on disk rather it searches for file paths in a database.

The database is a file that contains information about the files and their path on your system. The locate database file is located at:

```
/var/lib/mlocate/mlocate.db
```

**Q2. The next logical question is, what keeps this mlocate database updated?**

Well, there is another utility known as `updatedb`. When you execute `updatedb`, it scans the whole system and updates the `mlocate.db` database file.

So one limitation of the `'locate'` command is its dependency on the database which can be updated by another utility `'updatedb'`. Hence, in order to get the latest and reliable results from `'locate'` command the database on which it works should be updated at regular intervals.

We can also configure the `'updatedb'` utility as per our needs. This can be achieved by updating the `updatedb.conf`. This is a configuration file that `updatedb` reads before updating the database. `updatedb.conf` is located under `/etc/` :

```
# cat /etc/updatedb.conf
```

### **How to Use the locate Command**

The syntax for the `locate` command is as follows:

```
locate [OPTION] PATTERN...
```

When used without any options, `locate` displays every *absolute pathname* for which the user has access permission that contains any of the names of files and/or directories that are provided to it as arguments (i.e., input data).

The absolute pathname, also referred to as the *absolute path* or the *full path*, is the hierarchy of directories from the *root directory* to the designated file or directory. The root directory is the directory at the very top of the filesystem (i.e., hierarchy of files) that contains all other directories and files on the system and which is designated by a forward slash (`/`). It is important that the absolute pathname is returned both because it tells the exact locations on the system and because it makes it possible to indicate the locations of files or directories that have the same name but different absolute paths.

Thus, for example, the following would list the absolute paths of all files named *file1* and all directories named *dir1* for which the user had access permission:

```
locate file1 dir1
```

It would also list any other absolute pathnames that contained these strings (i.e., sequences of characters), for example `/home/john/file123` or `/usr/local/mydir1/index.html`.

For example, to search for a file named `.bashrc` you would type:

```
locate .bashrc
```

The output will include the names all files containing the string `.bashrc` in their names:

```
/etc/bash.bashrc
/etc/skel/.bashrc
/home/linuxize/.bashrc
/usr/share/base-files/dot.bashrc
/usr/share/doc/adduser/examples/adduser.local.conf.examples/bash.bashrc
/usr/share/doc/adduser/examples/adduser.local.conf.examples/skel/dot.bashrc
```

The `/root/.bashrc` file will not be shown because we ran the command as a normal user that doesn't have access permissions to the `/root` directory.

Similarly, after a file or directory has been removed, you need to make sure that the locate database has been updated, as otherwise, the command will keep showing the file in its output when searched.

- To limit the search results use the `-n` option followed by the number of results you want to be displayed. For example, the following command will search for all `.py` files and display only 10 results:

```
locate -n 10 *.py
```

- By default, locate performs case-sensitive searches. The `-i` (`--ignore-case`) option tells locate to ignore case and run case-insensitive search.

```
locate -i readme.md
/home/linuxize/p1/readme.md
/home/linuxize/p2/README.md
/home/linuxize/p3/ReadMe.md
```

- To display the count of all matching entries, use the `-c` (`--count`) option. The following command would return the number of all files containing `.bashrc` in their names:

```
locate -c .bashrc
6
```

**\*\*By default, locate doesn't check whether the found files still exist on the file system. If you deleted a file after the latest database update if the file matches the search pattern it will be included in the search results.**

- To display only the names of the files that exist at the time locate is run use the `-e` (`--existing`) option.



If it is desired to perform a more sophisticated search, including searching by attributes other than name (e.g., by size, creation date or location), the *find* command should be used.

As we already discussed earlier in this article, if a file is removed from the system, then until you update the locate database again, the command will keep showing that filename in output. For this specific case, however, you can skip updating the database, and still have correct results in output using the **-e** command line option.

For example, I removed the 'filetosearch.txt' file from my system. This was confirmed by the find command, which was no longer able to search the file:

```
HTF@HowtoForge:~$ find /home/himanshu/Downloads/ -name "*tosea
rch*"
HTF@HowtoForge:~$
```

But when I performed the same operation using locate, it was still showing the file in the output:

```
HTF@HowtoForge:~$ locate "*tosearch*"
/home/himanshu/Downloads/filetosearch.txt
HTF@HowtoForge:~$
```

And we know why - because locate's database wasn't updated after the file was deleted. However, using the **-e** option did the trick:

```
HTF@HowtoForge:~$ locate -e "*tosearch*"
HTF@HowtoForge:~$
```

Here's what the locate man page says about this option: "Print only entries that refer to files existing at the time locate is run."

### • How to separate output entries with ASCII NULL

By default, the output entries that the locate command produces are separated using newline (\n) character. But if you want, you can change the separator, and have the ASCII NULL instead of a newline. This can be done using the **-0** command line option.

For example, I've executed the same command we used in the last section above, but added the **-0** command-line option:

```
HTF@HowtoForge:~$ locate -l -0 "*tosearch*"
/home/himanshu/Downloads/NEWFILETOSEARCH.txt/home/himanshu/Dow
nloads/newfiletosearch.txthTF@HowtoForge:~$
```

So you can see that the newline separator is no longer there - it has been replaced with NULL.

### • How to view information about the locate database

In case you want to know which database locate is using, as well as other statistics about the database, use the **-S** command-line option.



```
HTF@HowtoForge:~$ locate -S
Database /var/lib/mlocate/mlocate.db:
 46,122 directories
 4,57,970 files
 2,69,56,478 bytes in file names
 1,13,67,408 bytes used to store database
HTF@HowtoForge:~$
```

### • How to search for an exact filename using locate

By default, when you search for a filename using locate, then the name you pass - say NAME - is implicitly replaced by \*NAME\*. For example, if I search for a filename 'testfile', then all names matching \*testfile\* are produced in the output:

```
HTF@HowtoForge:~$ locate testfile
/home/himanshu/.local/share/gedit/plugins/gedit-file-search/stfiles
/home/himanshu/.local/share/gedit/plugins/gedit-file-search/stfiles/U+0130_LATIN_CAPITAL_LETTER_I_WITH_DOT_ABOVE.txt
/home/himanshu/.local/share/gedit/plugins/gedit-file-search/stfiles/another'test"file
/home/himanshu/.local/share/gedit/plugins/gedit-file-search/stfiles/evil_binary_file_1.bin
/home/himanshu/.local/share/gedit/plugins/gedit-file-search/stfiles/file with spaces
/home/himanshu/.local/share/gedit/plugins/gedit-file-search/stfiles/file_for_whole_word_option.txt
/home/himanshu/.local/share/gedit/plugins/gedit-file-search/stfiles/testfile:with_evil_name_1
/home/himanshu/.local/share/gedit/plugins/gedit-file-search/stfiles/text_with_binary.txt
/home/himanshu/.local/share/gedit/plugins/gedit-file-search/stfiles/utf8-extract-1.bin
/home/himanshu/Downloads/grep-complete-match-testfile.png
/home/himanshu/Downloads/grep-testfile.png
/home/himanshu/Downloads/testfile
```

But what if the requirement is to search files with names exactly matching 'testfile'? Well, in this case, you'll have to use regular expressions, which can be enabled using the **-r** command-line option. So, here's how you can search for just 'testfile' using regular expressions:

```
locate -r /testfile$
```

```
HTF@HowtoForge:~$ locate -r /testfile$
/home/himanshu/Downloads/testfile
HTF@HowtoForge:~$
```

**Example 1:** Search a file with specific name.

```
$ locate sample.txt
```

It will search for sample.txt in particular directory.

**Example 2:** Limit Search Queries to a Specific Number.

```
$ locate "*.html" -n 20
```

It will show 20 results for the searching of file ending with *.html*.

**Example 3:** Display The Number of Matching Entries.

```
$ locate -c [.txt]*
```

It will count files ending with *.txt*.

**Example 4:** Ignore Case Sensitive Locate Outputs. This command is configured to process queries in a case sensitive manner. It means *SAMPLE.TXT* will show a different result than *sample.txt*.

```
$ locate -i *SAMPLE.txt*
```

**Example 5:** Separate Output Entries Without New Line.

```
$ locate -i -0 *sample.txt*
```

Default separator for locate command is the newline (`\n`) character. But if someone want to use a different separator like the ASCII NUL, the he/she can do so using the `-0` command line option.

## Locate with wildcards

The specificity of locate can be increased by using it together with *wildcards* or other *regular expressions*. Wildcards are [characters](#) that can be used to substitute for any other character or characters. For example, the star character ( `*` ) is a wildcard that can represent any single character or any string containing any number of characters. Regular expressions are a string that describes or matches a set of strings, according to certain syntax rules. For example, the following command uses the star wildcard to display all files on the system that have the *.png* [filename extension](#):

```
locate "*.png"
```

The `-q` option is used to suppress error messages, such as those that might be returned in the event that the user does not have permission to access designated files or directories. Thus, the following would suppress any error messages in the above example:

```
locate "*.png" -q
```

It is often the case that a large number of results will be returned for any query. The `-n` option followed by an integer limits the results to a specific number. For example, the following command would display only 15 results in a search for files that have an *.html* extension:

```
locate -n 15 "*.html"
```



An alternative is to use a [pipe](#) (represented by the vertical bar character) to [redirect](#) the output of locate from the display screen to a *pager* such as *more* or *less*, which presents only one screenful of output at a time, for example,

```
locate "*.html" | less
```

## **Wildcards**

Wildcards (also referred to as meta characters) are symbols or special characters that represent other characters. You can use them with any command such as ls command or rm command to list or remove files matching a given criteria, receptively.

Wildcards are a set of building blocks that allow you to create a pattern defining a set of files or directories. As you would remember, whenever we refer to a file or directory on the command line we are actually referring to a path. Whenever we refer to a path we may also use wildcards in that path to turn it into a set of files or directories.

These wildcards are interpreted by the shell and the results are returned to the command you run. There are three main wildcards in Linux:

- An asterisk (\*) – matches one or more occurrences of any character, including no character.
- Question mark (?) – represents or matches a single occurrence of any character.
- Bracketed characters ([ ]) – matches any occurrence of character enclosed in the square brackets. It is possible to use different types of characters (alphanumeric characters): numbers, letters, other special characters etc.

A *wildcard* is a character that can be used as a substitute for any of a class of characters in a search, thereby greatly increasing the flexibility and efficiency of searches.

Wildcards are commonly used in *shell* commands in Linux and other Unix-like operating systems. A shell is a program that provides a text-only user interface and whose main function is to execute commands typed in by users and display their results.

Wildcards are also used in *regular expressions* and programming languages. Regular expressions are a pattern matching system that uses *strings* (i.e., sequences of characters) constructed according to pre-defined syntax rules to find desired strings in text.

The term *wildcard* or *wild card* was originally used in card games to describe a card that can be assigned any value that its holder desires. However, its usage has spread so that it is now used to describe an unknown or unpredictable factor in a variety of fields.

### **Star Wildcard**



Three types of wildcards are used with Linux commands. The most frequently employed and usually the most useful is the *star wildcard*, which is the same as an asterisk (\*). The star wildcard has the broadest meaning of any of the wildcards, as it can represent zero characters, all single characters or any string.

As an example, the *file* command provides information about any filesystem object (i.e., file, directory or link) that is provided to it as an *argument* (i.e., input). Because the star wildcard represents every string, it can be used as the argument for file to return information about every object in the specified directory. Thus, the following would display information about every object in the *current directory* (i.e., the directory in which the user is currently working):

```
file *
```

If there are no matches, an error message is returned, such as *\*: can't stat `\*': (No such file or directory)*. In the case of this example, the only way that there would be no matches is if the directory were empty.

Wildcards can be combined with other characters to represent parts of strings. For example, to represent any filesystem object that has a *.jpg* filename extension, *\*.jpg* would be used. Likewise, *a\** would represent all objects that begin with a *lower case* (i.e., small) letter *a*.

As another example, the following would tell the *ls* command (which is used to list files) to provide the names of all files in the current directory that have an *.html* or a *.txt* extension:

```
ls *.html *.txt
```

Likewise, the following would tell the *rm* command (which is used to remove files and directories) to delete all files in the current directory that have the string *xxx* in their name:

```
rm *xxx*
```

### Question Mark Wildcard

The question mark (?) is used as a wildcard character in shell commands to represent exactly one character, which can be any single character. Thus, two question marks in succession would represent any two characters in succession, and three question marks in succession would represent any string consisting of three characters.

Thus, for example, the following would return data on all objects in the current directory whose names, inclusive of any extensions, are exactly three characters in length:

```
file ???
```

And the following would provide data on all objects whose names are one, two or three characters in length:

```
file ? ?? ???
```

As is the case with the star wildcard, the question mark wildcard can be used in combination with other characters. For example, the following would provide information about all objects in the current directory that begin with the letter *a* and are five characters in length:

```
file a????
```

The question mark wildcard can also be used in combination with other wildcards when separated by some other character. For example, the following would return a list of all files in the current directory that have a three-character filename extension:

```
ls *.???
```

### **Square Brackets Wildcard**

The third type of wildcard in shell commands is a pair of square brackets, which can represent any of the characters enclosed in the brackets. Thus, for example, the following would provide information about all objects in the current directory that have an *x*, *y* and/or *z* in them:

```
file *[xyz]*
```

And the following would list all files that had an extension that begins with *x*, *y* or *z*:

```
ls *. [xyz]*
```

The same results can be achieved by merely using the star and question mark wildcards. However, it is clearly more efficient to use the bracket wildcard.

When a hyphen is used between two characters in the square brackets wildcard, it indicates a range inclusive of those two characters. For example, the following would provide information about all of the objects in the current directory that begin with any letter from *a* through *f*:

```
file [a-f]*
```

And the following would provide information about every object in the current directory whose name includes at least one numeral:

```
file *[0-9]*
```



The use of the square brackets to indicate a range can be combined with its use to indicate a list. Thus, for example, the following would provide information about all filesystem objects whose names begin with any letter from *a* through *c* or begin with *s* or *t*:

```
file [a-cst]*
```

Likewise, multiple sets of ranges can be specified. Thus, for instance, the following would return information about all objects whose names begin with the first three or the final three lower case letters of the alphabet:

```
file [a-cx-z]*
```

Sometimes it can be useful to have a succession of square bracket wildcards. For example, the following would display all filenames in the current directory that consist of *jones* followed by a three-digit number:

```
ls jones[0-9][0-9][0-9]
```

You need to carefully choose which wildcard to use to match correct filenames: it is also possible to combine all of them in one operation as explained in the examples below.

## How to Match Filenames Using Wildcards in Linux

For the purpose of wildcards, following files to demonstrate each example.

```
createbackup.sh list.sh lspace.sh speaker.sh  
listopen.sh lost.sh rename-files.sh topprocs.sh
```

1. This command matches all files with names starting with *l* (which is the prefix) and ending with one or more occurrences of any character.

```
$ ls -l l*  
aaronkilik@tecmint ~/bin $ ls -l l*  
-rw-r--r-- 1 aaronkilik aaronkilik 0 Oct 4 10:44 listopen.sh  
-rw-r--r-- 1 aaronkilik aaronkilik 12 Oct 4 11:11 list.sh  
-rw-r--r-- 1 aaronkilik aaronkilik 0 Oct 4 10:45 lost.sh  
-rw-r--r-- 1 aaronkilik aaronkilik 0 Oct 4 10:44 lspace.sh  
aaronkilik@tecmint ~/bin $
```

2. This example shows another use of *\** to copy all filenames prefixed with *users-0* and ending with one or more occurrences of any character.

```
$ mkdir -p users-info
$ ls users-0*
$ mv -v users-0* users-info/ # Option -v flag enables verbose output
```

```
aaronkilik@tecmint ~ $ mkdir -p users-info
aaronkilik@tecmint ~ $
aaronkilik@tecmint ~ $ ls users-0*
users-01.list users-02.list users-03.txt users-06.list
users-01.txt users-02.txt users-05.list
aaronkilik@tecmint ~ $
aaronkilik@tecmint ~ $ mv -v users-0* users-info/
'users-01.list' -> 'users-info/users-01.list'
'users-01.txt' -> 'users-info/users-01.txt'
'users-02.list' -> 'users-info/users-02.list'
'users-02.txt' -> 'users-info/users-02.txt'
'users-03.txt' -> 'users-info/users-03.txt'
'users-05.list' -> 'users-info/users-05.list'
'users-06.list' -> 'users-info/users-06.list'
aaronkilik@tecmint ~ $
```

3. The following command matches all files with names beginning with l followed by any single character and ending with st.sh (which is the suffix).

```
$ ls l?st.sh
```

```
aaronkilik@tecmint ~/bin $ ls -l l?st.sh
-rw-r--r-- 1 aaronkilik aaronkilik 12 Oct  4 11:11 list.sh
-rw-r--r-- 1 aaronkilik aaronkilik  0 Oct  4 10:45 lost.sh
aaronkilik@tecmint ~/bin $
```

4. The command below matches all files with names starting with l followed by any of the characters in the square bracket but ending with st.sh.

```
$ ls l[abdcio]st.sh
```

```
aaronkilik@tecmint ~/bin $ ls -l l[abdcio]st.sh
-rw-r--r-- 1 aaronkilik aaronkilik 12 Oct  4 11:11 list.sh
-rw-r--r-- 1 aaronkilik aaronkilik  0 Oct  4 10:45 lost.sh
aaronkilik@tecmint ~/bin $
```

## How to Combine Wildcards to Match Filenames in Linux

You can combine wildcards to build a complex filename matching criteria as described in the following examples.

5. This command will match all filenames prefixed with any two characters followed by st but ending with one or more occurrence of any character.

```
$ ls  
$ ls ??st*
```

```
aaronkili@tecmint ~/bin $ ls  
createbackup.sh  list.sh  lspace.sh      speaker.sh  
listopen.sh      lost.sh  rename-files.sh topprocs.sh  
aaronkili@tecmint ~/bin $  
aaronkili@tecmint ~/bin $ ls ??st*  
listopen.sh  list.sh  lost.sh  
aaronkili@tecmint ~/bin $  
aaronkili@tecmint ~/bin $
```

6. This example matches filenames starting with any of these characters [clst] and ending with one or more occurrence of any character.

```
$ ls  
$ ls [clst]*
```

```
aaronkili@tecmint ~/bin $ ls  
createbackup.sh  list.sh  lspace.sh      speaker.sh  
listopen.sh      lost.sh  rename-files.sh topprocs.sh  
aaronkili@tecmint ~/bin $  
aaronkili@tecmint ~/bin $ ls [clst]*  
createbackup.sh  list.sh  lspace.sh  topprocs.sh  
listopen.sh      lost.sh  speaker.sh  
aaronkili@tecmint ~/bin $
```

7. In this examples, only filenames starting with any of these characters [clst] followed by one of these [io] and then any single character, followed by a t and lastly, one or more occurrence of any character will be listed.

```
$ ls  
$ ls [clst][io]?t*
```



```
aaronkilik@tecmint ~/bin $ ls
createbackup.sh  list.sh  lspace.sh  speaker.sh
listopen.sh      lost.sh  rename-files.sh  topprocs.sh
aaronkilik@tecmint ~/bin $
aaronkilik@tecmint ~/bin $ ls [clst][io]?t*
listopen.sh  list.sh  lost.sh
aaronkilik@tecmint ~/bin $
```

8. Here, filenames prefixed with one or more occurrence of any character, followed by the letters tar and ending with one or more occurrence of any character will be removed.

```
$ ls
$ rm *tar*
$ ls
```

```
aaronkilik@tecmint ~/bin $ ls
createbackup.sh  list.sh  lspace.sh  scripts.tar.bz2  speaker.sh
listopen.sh      lost.sh  rename-files.sh  scripts.tar.gz  topprocs.sh
aaronkilik@tecmint ~/bin $
aaronkilik@tecmint ~/bin $ rm *tar*
aaronkilik@tecmint ~/bin $
aaronkilik@tecmint ~/bin $ ls
createbackup.sh  list.sh  lspace.sh  speaker.sh
listopen.sh      lost.sh  rename-files.sh  topprocs.sh
aaronkilik@tecmint ~/bin $
aaronkilik@tecmint ~/bin $
```

## How to Match Characters Set in Linux

9. Now let's look at how to specify a set of characters. Consider the filenames below containing system user's information.

```
$ ls
users-111.list users-1AA.list users-22A.list users-2aB.txt users-2ba.txt
users-111.txt users-1AA.txt users-22A.txt users-2AB.txt users-2bA.txt
users-11A.txt users-1AB.list users-2aA.txt users-2ba.list
users-12A.txt users-1AB.txt users-2AB.list users-2bA.list
```

This command will match all files whose name starts with users-i, followed by a number, a lower case letter or number, then a number and ends with one or more occurrences of any character.

```
$ ls users-[0-9][a-z0-9][0-9]*
```

The next command matches filenames beginning with users-i, followed by a number, a lower or upper case letter or number, then a number and ends with one or more occurrences of any character.

```
$ ls users-[0-9][a-zA-Z0-9][0-9]*
```

This command that follows will match all filenames beginning with users-i, followed by a number, a lower or upper case letter or number, then a lower or upper case letter and ends with one or more occurrences of any character.

```
$ ls users-[0-9][a-zA-Z0-9][a-zA-Z]*
```

```
aaronkilik@tecmint ~/users-info $ ls
users-111.list  users-1AA.list  users-22A.list  users-2aB.txt  users-2ba.txt
users-111.txt  users-1AA.txt  users-22A.txt  users-2AB.txt  users-2bA.txt
users-11A.txt  users-1AB.list  users-2aA.txt  users-2ba.list
users-12A.txt  users-1AB.txt  users-2AB.list  users-2bA.list
aaronkilik@tecmint ~/users-info $
aaronkilik@tecmint ~/users-info $ ls users-[0-9][a-z0-9][0-9]*
users-111.list  users-111.txt
aaronkilik@tecmint ~/users-info $
aaronkilik@tecmint ~/users-info $ ls users-[0-9][a-zA-Z0-9][0-9]*
users-111.list  users-111.txt
aaronkilik@tecmint ~/users-info $ ls users-[0-9][a-zA-Z0-9][a-zA-Z]*
users-11A.txt  users-1AB.list  users-2aA.txt  users-2ba.list
users-12A.txt  users-1AB.txt  users-2AB.list  users-2bA.list
users-1AA.list  users-22A.list  users-2aB.txt  users-2ba.txt
users-1AA.txt  users-22A.txt  users-2AB.txt  users-2bA.txt
aaronkilik@tecmint ~/users-info $
```

## How to Negate a Set of Characters in Linux

10. You can as well negate a set of characters using the ! Symbol. The following command lists all filenames starting with users-i, followed by a number, any valid file naming character apart

from a number, then a lower or upper case letter and ends with one or more occurrences of any character.

```
$ ls users-[0-9][!0-9][a-zA-Z]*
```

## Some more examples

For all the examples below, assume we are in the directory linuxtutorialwork and that it contains the files as listed above. Also note that I'm using ls in these examples simply because it is a convenient way to illustrate their usage. Wildcards may be used with any command.

1. \$ pwd

```
/home/ryan/linuxtutorialwork
```

2. \$ ls

```
barry.txt blah.txt bob example.png firstfile foo1 foo2 foo3 frog.png secondfile  
thirdfile video.mpeg
```

Every file with an extension of txt at the end. In this example we have used an absolute path. Wildcards work just the same if the path is absolute or relative.

1. ls /home/ryan/linuxtutorialwork/\*.txt
2. /home/ryan/linuxtutorialwork/barry.txt /home/ryan/linuxtutorialwork/blah.txt
- 3.

Now let's introduce the ? operator. In this example we are looking for each file whose second letter is i. As you can see, the pattern can be built up using several wildcards.

1. ls ?i\*
2. firstfile video.mpeg
- 3.

Or how about every file with a three letter extension. Note that video.mpeg is not matched as the path name must match the given pattern exactly.

1. ls \*.???
2. barry.txt blah.txt example.png frog.png
- 3.



And finally the range operator ( [ ] ). Unlike the previous 2 wildcards which specified any character, the range operator allows you to limit to a subset of characters. In this example we are looking for every file whose name either begins with a s or v.

1. `ls [sv]*`
2. `secondfile video.mpeg`
- 3.

With ranges we may also include a set by using a hyphen. So for example if we wanted to find every file whose name includes a digit in it we could do the following:

1. `ls *[0-9]*`
2. `foo1 foo2 foo3`
- 3.

We may also reverse a range using the caret ( ^ ) which means look for any character which is not one of the following.

1. `ls [^a-k]*`
2. `secondfile thirdfile video.mpeg`
- 3.

## **Some Real World Examples**

The examples above illustrate how the wildcards work but you may be wondering what use they actually are. People use them everywhere and as you progress I'm sure you'll find many ways in which you can use them to make your life easier. Here are a few examples to give you a taste of what is possible. Remember, these are just a small sample of what is possible, and they can be used whenever you specify a path on the command line. With a little creative thinking you'll find they can be used in all manner of situations.

1. Find the file type of every file in a directory.

```
$ file /home/ryan/*
bin: directory
Documents: directory
frog.png: PNG image data
public_html: directory
```

2. Move all files of type either jpg or png (image files) into another directory.

```
mv public_html/*.??g public_html/images/
```

3. Find out the size and modification time of the `.bash_history` file in every users home directory. (`.bash_history` is a file in a typical users home directory that keeps a history of commands the user has entered on the command line. Remember how the `.` means it is a hidden file?) As you can see in this example, we may use wildcards at any point in the path.

```
ls -lh /home/*/.bash_history
-rw----- 1 harry users 2.7K Jan 4 07:32 /home/harry/.bash_history
-rw----- 1 ryan users 3.1K Jun 12 21:16 /home/ryan/.bash_history
```