How to use the which command in Linux

The which command in Linux determines the executable binary, which will execute when you issue a command to the shell. If you have different versions of the same program on your computer, you can use which to find out which shell will use.

The `which` command in Linux determines the executable binary, which will execute when you issue a command to the shell. If you have different versions of the same program on your computer, you can use `which` to find out which shell will use.

Instructions for using which command in Linux

1. Binary and path
2. See the links
3. Which command
4. See results
5. Check multiple commands at the same time

Binary and path

When you try to run a program or command from a Terminal window, the shell (usually Bash on modern distributions) has to find that command and launch it. A number of commands, such as `cd` , `history` and `pwd` , are integrated into the shell, so Bash does not take much effort to find them.

But how does Bash locate external commands, programs and other independent binaries? Bash uses paths, exactly a set of paths, each leading to a directory. It then searches each directory to come up with an executable file that matches the command or program you're trying to run. When you find a suitable file, Bash will launch it and give up searching.

You can use `echo` to check the **$ PATH** environment **variable** and see the directories in the path. To do so, enter the following, then press `Enter` :

```
echo $PATH
```

The output list separates each path with a colon (`:` . On the computer the article uses, Bash will search the directories in the following order:

1. / usr / local / sbin
2. / usr / local / bin
3. / usr / sbin
4. / usr / bin
5. / sbin
6. / bin
7. / user / games
8. / usr / local / games
9. / snap / bin

There are many directories called **/ sbin** and **/ bin** in the file system, which can lead to some confusion.

See the links

Suppose there is an updated version of a program called **htg.** It has a name in the current directory and you can run it by typing the following command:

```
./htg
```

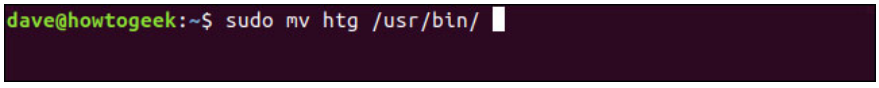
This is not really a program, it just prints the version number and then closes. The new version is **1.2.138.**

To run a program in the current working directory, you must enter '**./**' before the program name, so Bash knows where to find it.

Because the article wants to run this particular program from any directory, the example will move the executable program to the **/ usr / bin directory.** Bash will find that program in the path and run it.

For example, there is no need to execute in the current directory, so there is no need to type '**./**' before the program name, as shown below:

```
sudo mv htg /usr/bin
```



Now, let the program run by typing:

```
htg
```

The command runs, but not the new, updated program, but instead, the older version, **1.2.105.**

`which` command

The problem the article demonstrated above is why the `which` command was created.

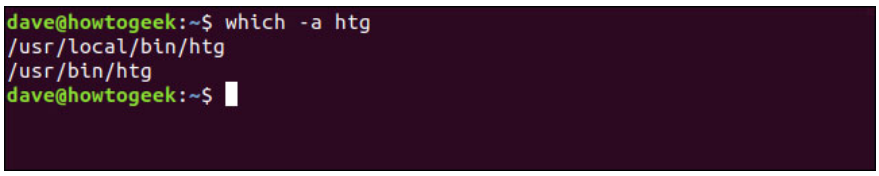
In this example, we will use `which` and pass the name of the program we are testing as a command line parameter:

```
which htg
```

The `which` command reports it found a version of **htg** in the **/ usr / local / bin directory.** Because that location appears in the path before the directory where we moved the updated **htg** , Bash will use the previous version of the program.

However, if we use the **-a** (all) option as shown below, it will continue searching even if a match is found:

```
which -a htg
```



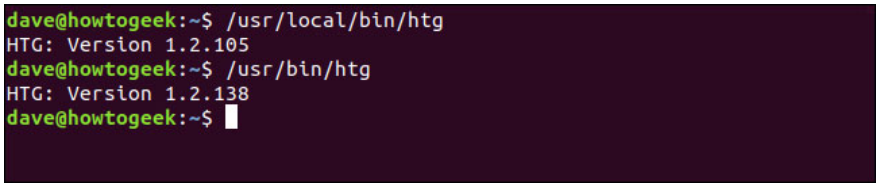
It then lists all matching results of any directory in the path.

Therefore, there is a problem, which is that the previous version of the program in the directory is also included in the patch. And that directory is being searched before the directory where we have placed the new version of the program.

To verify, you can enter the following and run each version of the program:

```
/usr/local/bin/htg
```

```
/usr/bin/htg
```



This explains the cause of the problem, and the solution is very simple.

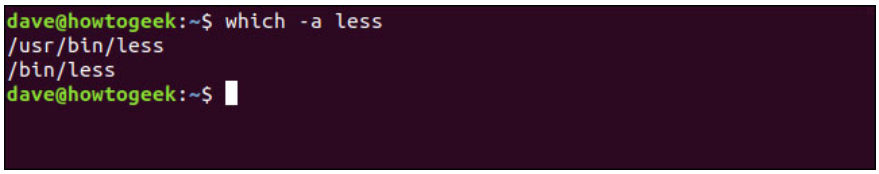
Actually, you have the options: Delete the old version in / **use** / **local** / **bin directory** or move it from / **usr** / **bin** to / **usr** / **local** / **bin.**

See results

Two results do not mean two binary files.

Let's look at an example in which the article will use the `which` command with the **-a** (all) option and find the versions of the `less` program:

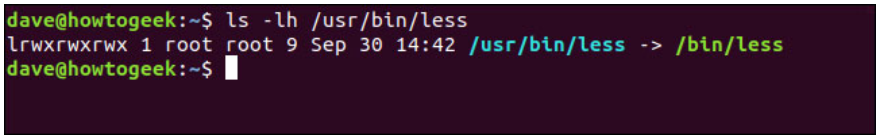
```
which -a less
```



The `which` statement reports two locations containing a version of the `less` program, but is that true? It's strange that there are two different versions (or the same version in multiple locations) of the `less` program installed on Linux computers. Therefore, we will not accept the output from `which` . Instead, dig a little deeper.

You can use the `ls` , `-l` (long listing) and `-h` (human-readable) options to see what happens next:

```
ls -lh /usr/bin/less
```

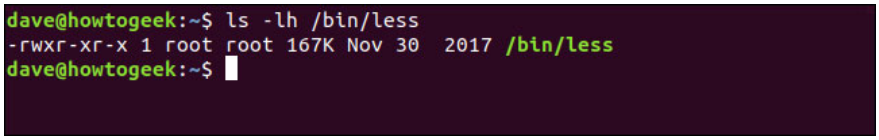


The reported file size is 9 bytes! This is definitely not a full copy of `less` .

The first character of the list is an **l** . A normal file will have hyphens (**-**) as the first character. **'l'** is the symbol of the symbolic link. If you miss that detail, the -> icon also indicates this is a symbolic link (you can think of it as a kind of shortcut). This indicates a copy of `less` in / **bin.**

Please try again with the `less` version in / **bin:**

```
ls -lh /bin/less
```



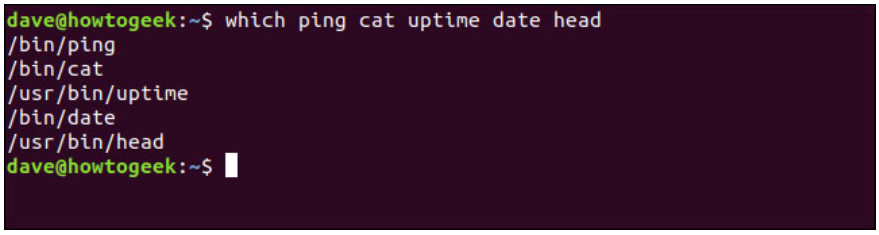
This entry is obviously a real binary executable. The first character of the list is a hyphen (`-`), which means it is a regular file and the file size is 167KB. Therefore, only a `less` copy is installed, but there is a symbolic link to it from another directory, which Bash also finds when searching for the path.

Check multiple commands at the same time

You can pass multiple programs and commands to `which` and it will check them in order.

For example, if you type:

```
which ping cat uptime date head
```



`which` command will work according to the list of programs and commands you have provided, and then list the results for each program.

In addition to digging into the Linux file system out of curiosity, `which` most useful when you expect a set of behaviors from a command or program, but get something else.

You can use `which` of these cases to verify that the Bash command is starting is the command you want to use.