Wildcards!

Taming the file system.

Introduction

In the section on File Manipulation we learnt about a few commands to do interesting things. The problem was that they all operated on a single file at a time, not very efficient. Now I'm going to introduce a means to play about with a set of files at once.

So what are they?

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.

Here is the basic set of wildcards:

- ***** - represents zero or more characters
- **?** - represents a single character
- **[]** - represents a range of characters

As a basic first example we will introduce the *****. In the example below we will list every entry beginning with a b.

Terminal

```
1.  user@bash: pwd
```

```
/home/ryan/linuxtutorialwork
user@bash:
user@bash: ls
barry.txt blah.txt bob example.png firstfile foo1 foo2
foo3 frog.png secondfile thirdfile video.mpeg
user@bash:
user@bash: ls b*
barry.txt blah.txt bob
user@bash:
```

Under the Hood

The mechanism here is actually kinda interesting. On first glance you may assume that the command above (ls) receives the argument **b*** then proceeds to translate that into the required matches. It is actually bash (The program that provides the command line interface) that does the translation for us. When we offer it this command it sees that we have used wildcards and so, before running the command (in this case ls) it replaces the pattern with every file or directory (ie path) that matches that pattern. We issue the command:

- ls b*

Then the system translates this into:

- ls barry.txt blah.txt bob

and then executes the program. The program never sees the wildcards and has no idea that we used them. This is funky as it means we can use them on the command line whenever we want. We are not limited to only certain programs or situations.

Some more examples

Some more examples to illustrate their behaviour. 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.

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.

```
                                    Terminal
user@bash: ls /home/ryan/linuxtutorialwork/*.txt
/home/ryan/linuxtutorialwork/barry.txt /home/ryan/linuxtutorialwork/blah.txt
user@bash:
```

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.

```
Terminal
1.  user@bash: ls ?i*
2.  firstfile video.mpeg
3.  user@bash:
```

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.

```
Terminal
1.  user@bash: ls *.???
2.  barry.txt blah.txt example.png frog.png
3.  user@bash:
```

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.

```
Terminal
1.  user@bash: ls [sv]*
2.  secondfile video.mpeg
3.  user@bash:
```

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:

```
Terminal
1.  user@bash: ls *[0-9]*
2.  foo1 foo2 foo3
3.  user@bash:
```

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

```
Terminal
1.  user@bash: ls [^a-k]*
2.  secondfile thirdfile video.mpeg
```

```
3. user@bash:
```


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.

Find the file type of every file in a directory.

```
Terminal
1. user@bash: file /home/ryan/*
2. bin: directory
3. Documents: directory
4. frog.png: PNG image data
5. public_html: directory
6. user@bash:
```

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

```
Terminal
1. user@bash: mv public_html/*.??g public_html/images/
2. user@bash:
```

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.

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

4. `user@bash:`

Summary

Stuff We Learnt

No new commands introduced in this section.

Important Concepts

Anywhere in any path
Wildcards may be used at any part of a path.

Wherever a path is used
Because wildcard substitution is done by the system, not the command, they may be used wherever a path is used.

Activities

Let's play with some patterns.

- A good directory to play with is /etc which is a directory containing config files for the system. As a normal user you may view the files but you can't make any changes so we can't do any harm. Do a listing of that directory to see what's there. Then pick various subsets of files and see if you can create a pattern to select only those files.
- Do a listing of /etc with only files that contain an extension.
- What about only a 3 letter extension?
- How about files whose name contains an uppercase letter? (hint: [[:upper:]] may be useful here)
- Can you list files whose name is 4 characters long?

❮ VI - Text Editor (./vi.php) Permissions ❯ (./permissions.php)

Home

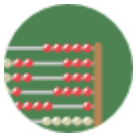
(/)

Linux Tutorial

(/linuxtutorial/)

HTML Tutorial

(/html-tutorial/)



Binary Tutorial

(/binary-tutorial/)

Education is the kindling of a flame,
not the filling of a vessel.
- *Socrates*

Contact (/contact.php) | Disclaimer (/disclaimer.php)

Bash Scripting Tutorial

(/bash-scripting-tutorial/)

CSS Tutorial

(/css-tutorial/)

Regular Expressions

(/regular-expressions-tutorial/)

Programming Challenges

(/programming-challenges/)

Problem Solving

(/problem-solving-skills/)

Boolean Algebra Tutorial

(/boolean-algebra-tutorial/)

Basic Design Tutorial

(/graphic-design-tutorial/)

Solve the Cube

(/rubiks-cube-tutorial/)

Software Design and Development

(/software-design-and-development/)

micro:bit Tutorial

(/microbit-tutorial/)