Wildcards

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.

```
pwd
/home/ryan/linuxtutorialwork

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

ls b*
barry.txt blah.txt bob
```

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.

```
ls /home/ryan/linuxtutorialwork/*.txt

/home/ryan/linuxtutorialwork/barry.txt
/home/ryan/linuxtutorialwork/blah.txt
```

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.

```
ls ?i*

firstfile video.mpeg
```

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.

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

```
barry.txt blah.txt example.png frog.png
```

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.

```
ls [sv]*

secondfile video.mpeg
```

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:

```
ls *[0-9]*

foo1 foo2 foo3
```

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

```
ls [^a-k]*

secondfile thirdfile video.mpeg
```