WILDCARD AND PATTERN MATCHING EXAMPLES

Linux Wildcards

Wildcards - Pattern Matching - Shell Expansion

During your use of linux you will find it very handy to get a basic appreciation of wildcards, pattern matching and expansion techniques. These abbreviations are very powerful and and c considerable amount of time.

For example, if we needed to create multiple directories all beginning with the name "section" and then followed by a number, we could issue the command:
mkdir section01 section02 section03 section04.
As you can imagine this would be quite time consuming if there were a lot to create. However, we could save a lot of time by issuing the command:
mkdir section{01..10}. Here shell expansion comes to the rescue.

```
john@john-desktop:~/test_examples$ ls -l
total 0
john@john-desktop:~/test_examples$ mkdir section{01..10}
john@john-desktop:~/test_examples$ ls -l
total 40
drwxrwxr-x 2 john john 4096 Jan 20 13:44 section01
drwxrwxr-x 2 john john 4096 Jan 20 13:44 section02
drwxrwxr-x 2 john john 4096 Jan 20 13:44 section03
drwxrwxr-x 2 john john 4096 Jan 20 13:44 section04
drwxrwxr-x 2 john john 4096 Jan 20 13:44 section05
drwxrwxr-x 2 john john 4096 Jan 20 13:44 section06
drwxrwxr-x 2 john john 4096 Jan 20 13:44 section07
drwxrwxr-x 2 john john 4096 Jan 20 13:44 section08
drwxrwxr-x 2 john john 4096 Jan 20 13:44 section09
drwxrwxr-x 2 john john 4096 Jan 20 13:44 section10
john@john-desktop:~/test_examples$ rmdir section*
john@john-desktop:~/test_examples$ ls -l
total 0
```

As you can see from the above example, the command line can be very powerful. One simple command created ten directories and one command then removed all ten directories that r pattern of "section*".

Wildcards and Globbing

As you use Linux more and more, you will often come across scenarios where you need to perform an operation on multiple filesystem objects. As we saw earlier, it can be quite time cc have to specify every filesystem object. Well thankfully you do not, Linux has in built wildcard support also commonly known as "Globbing". For example if you had multiple files called t test03, test04 and test05, we could use a wildcard pattern in conjunction with the "rm" command to delete these. In the example below we can see two methods in use:

```
john@john-desktop:~/test_examples$ touch file{1..5}
john@john-desktop:~/test_examples$ ls -l
total 0
-rw-rw-r-- 1 john john 0 Jan 20 14:14 file1
-rw-rw-r-- 1 john john 0 Jan 20 14:14 file2
-rw-rw-r-- 1 john john 0 Jan 20 14:14 file3
-rw-rw-r-- 1 john john 0 Jan 20 14:14 file4
-rw-rw-r-- 1 john john 0 Jan 20 14:14 file5
john@john-desktop:~/test_examples$ rm file[1-5]
john@john-desktop:~/test_examples$ ls -l
total 0
john@john-desktop:~/test_examples$ touch file{1..5}
john@john-desktop:~/test_examples$ ls -l
total 0
```

```
-rw-rw-r-- 1 john john 0 Jan 20 14:14 file1
-rw-rw-r-- 1 john john 0 Jan 20 14:14 file2
-rw-rw-r-- 1 john john 0 Jan 20 14:14 file3
-rw-rw-r-- 1 john john 0 Jan 20 14:14 file4
-rw-rw-r-- 1 john john 0 Jan 20 14:14 file5
john@john-desktop:~/test_examples$ rm file*
john@john-desktop:~/test_examples$ ls -l
total 0
```

As you can see in the above example, any files that matched the pattern "**file[1-5]**" or "**file***" were deleted. In the above example the "**file[1-5]**" will match any files that begin "**file**" an
either a **1**, **2**, **3**, **4** or **5**. However, the pattern "**file***" would match any files that start with the name "file". The *wildcard option matches any characters, or even "no characters".

We can also list specific filesystem objects by adding a pattern to the "ls" command: "**ls -d /etc/pa***". You can add to the pattern to filter out even more results: "**ls -d /etc/pap***". In the
any filesystem objects that contained "/etc/pa" will be returned. In the amended example we modified the pattern to only include filesystem objects that contained the pattern "/etc/p

```
john@john-desktop:~$ ls -d /etc/pa*
/etc/pam.conf  /etc/pam.d  /etc/papersize  /etc/passwd  /etc/passwd-
john@john-desktop:~$ ls -d /etc/pap*
/etc/papersize
```

One important thing to note is that if you use the pattern matching and no pattern is matched, you will receive an error. The message will normally take the form of : **No such file or di**

```
john@john-desktop:~$ ls -d /etc/john*
ls: cannot access /etc/john*: No such file or directory
```

So why the error in the above example? Well if the specified pattern matches one or more files in the specified path, Bash would then replaces the specified pattern with a space separa
matching objects. If the pattern did not match any objects, then Bash will leave the wildcarded pattern and return an error indicating that it couldn't find /etc/john*.

The point to remember here is that glob patterns are only expanded if they match the objects in the filesystem.

Wild Card Expansion - "*" and "?" Special Characters

Below are examples of using wildcard expansion with the special characters "*" and "?".

Matching with a *

A * will match zero or more characters.

Example:

/etc/p* would match any files in /etc that begin with a "p" or a file on its own called "p"

/home/landoflinux/wi*1 matches any files in the directory /home/landoflinux that begin with "wi" and end in "1"

```
$ ls -l w*
-rw-rw-r-- 1 landoflinux landoflinux 0 Jan 20 21:12 wi1
-rw-rw-r-- 1 landoflinux landoflinux 0 Jan 20 21:12 wilcard
-rw-rw-r-- 1 landoflinux landoflinux 0 Jan 20 21:12 wildcard1
-rw-rw-r-- 1 landoflinux landoflinux 0 Jan 20 21:13 wildcard2
$ ls -l wi*1
-rw-rw-r-- 1 landoflinux landoflinux 0 Jan 20 21:12 wi1
-rw-rw-r-- 1 landoflinux landoflinux 0 Jan 20 21:12 wildcard1
$ ls -l wi*2
-rw-rw-r-- 1 landoflinux landoflinux 0 Jan 20 21:13 wildcard2
```

Matching with a ?

A ? will match any single character.

Example:

/home/landoflinux/wildcard? would match any file whose name consists of wildcard followed by a single character

```
$ ls -l w*

-rw-rw-r-- 1 landoflinux landoflinux 0 Jan 20 21:12 wi1
-rw-rw-r-- 1 landoflinux landoflinux 0 Jan 20 21:12 wilcard
```

```
-rw-rw-r-- 1 landoflinux landoflinux 0 Jan 20 21:12 wildcard1
-rw-rw-r-- 1 landoflinux landoflinux 0 Jan 20 21:13 wildcard2
$ ls -l wildcard?
-rw-rw-r-- 1 landoflinux landoflinux 0 Jan 20 21:12 wildcard1
-rw-rw-r-- 1 landoflinux landoflinux 0 Jan 20 21:13 wildcard2
```

Matching with []

To use this wildcard, you simply place the characters you wish to match inside the square brackets "[ ]". You can also add a dash "-" between the brackets to specify a range or a combin ranges.
wildcard[12] will match the file wildcard1 and wildcard2

```
$ ls -l wildcard[12]

-rw-rw-r-- 1 landoflinux landoflinux 0 Jan 20 21:12 wildcard1
-rw-rw-r-- 1 landoflinux landoflinux 0 Jan 20 21:13 wildcard2
```

[Ww]ildcard? will match any file that starts "Wildcard" or "wildcard" and ends with a single character. This option is very useful when dealing with upper and lowercase files. (Remembe sensitive the file Wildcard is different to the file wildcard).

```
$ ls -l [Ww]ildcard?
-rw-rw-r-- 1 landoflinux landoflinux 0 Jan 20 21:12 wildcard1
-rw-rw-r-- 1 landoflinux landoflinux 0 Jan 20 21:42 Wildcard1
-rw-rw-r-- 1 landoflinux landoflinux 0 Jan 20 21:13 wildcard2
-rw-rw-r-- 1 landoflinux landoflinux 0 Jan 20 21:42 Wildcard2
```

Excluding patterns with a [!]

As well as being able to match a range within the square brackets, you can also exclude matches within the brackets with the use of the exclamation mark "!".

ls -l [Ww]ildcard[!1] will match the files wildcard2 and Wildcard2 (see example below):

```
$ ls -l w*
-rw-rw-r-- 1 landoflinux landoflinux 0 Jan 20 21:12 wi1
-rw-rw-r-- 1 landoflinux landoflinux 0 Jan 20 21:12 wilcard
-rw-rw-r-- 1 landoflinux landoflinux 0 Jan 20 21:12 wildcard1
-rw-rw-r-- 1 landoflinux landoflinux 0 Jan 20 21:13 wildcard2
$ ls -l [Ww]ildcard[!1]
-rw-rw-r-- 1 landoflinux landoflinux 0 Jan 20 21:13 wildcard2
-rw-rw-r-- 1 landoflinux landoflinux 0 Jan 20 21:42 Wildcard2
```