NDG Introduction to Linux I - Chapter 4: File Globbing

Objectives

Key Terms

file globbing
The glob function searches for all the pathnames matching pattern according to the rules used by the shell.

Section 4.1 | Section 4.2 | Section 4.3 | Section 4.4

Content

4.1 Introduction

File globbing might sound like an unusual term, but the concept of globbing is probably a familiar one. Like a joker in a deck of cards, which you might be able to use as any card (a wildcard), glob characters are special characters that can be used to match file or path names.

For the details of how globbing works, you can view the man page on glob in section 7 of the manual pages by executing:

```
sysadmin@localhost:~$ man 7 glob
```

From that manual page:

- A string is a wildcard pattern if it contains one of the characters '?', '*', or '['. Globbing is the operation that expands a wildcard pattern into the list of path names matching the pattern.

The shell is what performs the expansion of the wildcards; therefore, the use of globbing is not limited to a particular command. Globbing is especially useful for file management since it can be used to match the path names to files that you want to manage.

4.2 Asterisk * Character

The asterisk* wildcard can be used to match any string, including the empty string.

```
sysadmin@localhost:~$ echo *

Desktop Documents Downloads Music Pictures Public Templates Videos
```

In the screen capture above, the echo command is used to see which files a glob pattern will match. In this case, it displays the name of every file in the user's current directory.

> Consider This
> An empty string is a string that has nothing in it. It is commonly referred to as "".

In order to limit the * wildcard, so that it matches fewer file names, it can be combined with other characters. For example, the D* pattern will only match file names that start with a D character:

```
sysadmin@localhost:~$ echo D*
Desktop Documents Downloads
```

The * character can appear anywhere in the pattern; it can even appear multiple times. It is possible to find a file that not only starts with a D, but also contains the letter n, by using the D*n* pattern:

```
sysadmin@localhost:~$ echo D*n*
Documents Downloads
```

4.3 Question Mark ? Character

The question mark ? character in a string will match exactly one character. For example, to see all of the files in the /usr/bin directory that only have one character in the file name, use the /usr/bin/? pattern:

```
sysadmin@localhost:~$ cd /usr/bin
sysadmin@localhost:/usr/bin$ echo /usr/bin/?
/usr/bin/[ /usr/bin/w
```

By adding more question mark ? characters, it is possible to match additional characters. In order to see which files had two or three characters as their file names, execute the following two echo commands:

```
sysadmin@localhost:/usr/bin$ echo ??
du ex hd id nl od pg pr sg tr ul vi wc xz
sysadmin@localhost:/usr/bin$ echo ???
a2p apt awk cal cmp col cut dig env eqn fmt gpg lcf ldd man pdb pic ptx rcp rev
rsh s2p scp see seq ssh sum tac tbl tee tic toe top tty ucf who xxd yes zip
```

The incorporation of other characters to patterns using the question mark ? character, including other wildcard characters, allow for more specific searches. For example, the w?? pattern will match files that started with the letter w, and are exactly three characters long:

```
sysadmin@localhost:/usr/bin$ echo w??
who
```

While the w??* pattern will match files that started with the letter w, but are at least three characters long:

```
sysadmin@localhost:/usr/bin$ echo w??*
w.procps wall watch wget whatis whereis which who whoami write
```

4.4 Brackets [ ] Characters

By using the square bracket []characters, a set of characters can be enclosed that will be used to match exactly one character. You can think of the square brackets as being like a question mark ? wildcard, but limited to the characters that are listed inside of the square brackets.

For example, using a pattern like [abcd]?? will match file names that start with an a, b, c, or d character and that are three characters long. Alternatively, letters that are consecutive can also be expressed as a range like the [a-d]?? pattern:

```
sysadmin@localhost:/usr/bin$ echo [a-d]??
a2p apt awk cal cmp col cut dig
```

Consider This

When the hyphen - character is used with square brackets, it is referred to as a range. For example: the [a-z] pattern would match any single character that is within the range of characters from a to z.

This range is defined by the ASCII text table. Simply use a lower value character from the ASCII table as the first character and a higher value character as the second.

To see the numeric value assigned to each character in the ASCII text table, either do a simple search for "ASCII text table" on the internet or view the table with the ascii command:

```
sysadmin@localhost:~$ ascii
    touch

    touch

sysadmin@localhost:/usr/bin$ cd ~/Documents
sysadmin@localhost:~/Documents$ touch '*' '**' '***' '?' '??' '???' '[][][]'
sysadmin@localhost:~/Documents$ ls
'*' School alpha-third.txt letters.txt people.csv
'**' Work alpha.txt linux.txt profile.txt
```

```
'***' '[][][]' animals.txt longfile.txt red.txt
'?' adjectives.txt food.txt newhome.txt spelling.txt
'??' alpha-first.txt hello.sh numbers.txt words
'???' alpha-second.txt hidden.txt os.csv
```

When placed inside the square brackets, the wildcard characters lose their wildcard meaning and only match themselves. So, a pattern like [*]* would match a file name that begins with an * character. A pattern of [?]* would match a file name that begins with a ? character.

```
sysadmin@localhost:~/Documents$ echo [*]*
* ** ***
sysadmin@localhost:~/Documents$ echo [?]*
? ?? ???
```

Even using square brackets inside of the square brackets will match a square bracket, so [[]]* would match a file name that starts with the square brackets:

```
sysadmin@localhost:~/Documents$ echo [[]]*
[][][]
```

Warning

Normally, you want to avoid using special characters like *, ?, [and] within file names because these characters, when used in a file name on the command line, can end up matching other file names unintentionally.

Just as with the other wildcard characters, the square brackets can appear multiple times in a pattern. For example, to match a file in the /usr/bin directory that contains at least two digits in the file name, use the /etc/*[0-9][0-9]* pattern:

```
sysadmin@localhost:~/Documents$ cd /usr/bin
sysadmin@localhost:/usr/bin$ echo *[0-9][0-9]*
base64 i386 linux32 linux64 perl5.18.2 pstree.x11 sha224sum sha256sum sha384sum sha512sum x86_64
```

The square brackets also support negating the set of characters (also called complementation). If the first character inside of the square brackets is either an exclamation ! character or a caret ^ character, then that first character has a special meaning of not the following characters, meaning match a single character that is not within the set of characters that follows it.

In other words, the [!a-v]* pattern would match a file name that does not begin with a letter a through v:

```
sysadmin@localhost:/usr/bin$ echo [!a-v]*
2to3 2to3-2.7 2to3-3.4 [ w w.procps wall watch wc wget whatis whereis which who
whoami write x86_64 xargs xauth xgettext xsubpp xxd xz xzcat xzcmp xzdiff xzegrep xzfgrep xzgrep xzless xzmore yes
zdump zip zipcloak zipdetails zipgrep zipinfo zipnote zipsplit zsoelim
```