A/UX: Bourne Shell Metacharacters (6/93)

Article Created: 21 October 1988
Article Reviewed/Updated: 24 June 1993

TOPIC ---

This article discusses Bourne shell metacharacters.

DISCUSSION --

A metacharacter is a character that performs a special function in the shell, like expanding a file name or redirecting output. In fact, the metacharacters in this article fall into two categories: wildcard characters for file name expansion and redirection of input/output and process.

File name expansion characters include

- Asterisk *
- Brackets [ ]
- Brackets-Hyphen [-]
- Question mark ?

Input/output and process redirection characters include

- Ampersand &
- Back quote `
- Backslash \
- Braces { }
- Double greater than > >
- Greater than >
- Less than <
- Parentheses ( )
- Semicolon ;
- Slash /
- Vertical bar |

The article is divided into two sections (one for each type of character). All metacharacters are listed alphabetically within their sections.

FILE NAME EXPANSION METACHARACTERS

The Asterisk

The asterisk (*) substitutes in a file or directory name for zero or more characters, except a leading period (like ".ash").

Example: The command

```
ls *ash
```

returns file names ending with the letters "ash" like "ash", "bash", "cash", "mash", and "splash". However, it won't return strings that have characters after the "ash" like "bash.tmp" or "bashful.

Brackets

Brackets ([ ]) cause the shell to look for a match for each character between the brackets, one at a time. It does not match the period character.

Example: ls [bm]ash returns file names "bash" and "mash", but not "cash" or "bash.tmp".

A variation is brackets with a hyphen separating two characters. It causes the shell to match any character within the range of these characters in a file or directory name.

Example: ls [a-c]ash returns file names that have letters in the range from "a" through "c" as the first character of their name and that are followed by the letters "ash". Thus, "ash", "bash", and "cash" match, but not "dash" or "mash".

You can place a bracket metasequence anywhere in a string.

Question Mark

The question mark (?) substitutes for any ONE character in the same position in a file or directory name. The question mark does not expand file names, nor does not match a leading period.

Example: "ls ?ash" returns only file names that have one character followed by the characters "ash". Thus, "bash", and "cash" match, but not "ash" or "splash".

REDIRECTION METACHARACTERS

Ampersand

The ampersand (&), placed at the end of a line, causes the task it follows to run in the background. When using the ampersand, the shell returns the task's PID (Process ID Number). Note: In the Bourne shell, you cannot retrieve a task from the background.

Example: cat /etc/passwd &

Back Quotes

The back quotes (` `) contain any UNIX command. When the shell executes the line containing the back-quoted command, it replaces the command and with the command's output in the string. Without the back quotes, the shell would treat a command like an ordinary string.

For example,

```
echo today is date
```

returns "echo today is date", whereas

```
echo today is `date`
```

returns "today is " followed by the system date.

Backslash

The backslash (\), preceding a metacharacter, causes the shell to interpret the character as a regular character rather than as a metacharacter. Typically, this can stop a variable from returning its value.

For example, type in the following three lines of code. And notice that the third line does not return the variable's value.

```
x=hi
echo $x
echo \$x
```

Dollar Sign

The dollar sign ($) causes the shell to evaluate or display the value of the variable the dollar sign precedes. The following to lines output "hi", the contents of variable x.

```
x=hi
echo $x
```

Double Quotes

The double quotes (" ") serve three purposes. First, it lets you put tabs and spaces in a string as you assign the string to a variable. Second, when you display the contents of a variable, surrounding the variable name in double quotes preserves tabs and multiple contiguous spaces and tabs. Try this example:

```
y=me and you
```

The shell refuses the command when it hits the first space. Try this (including multiple spaces after "and"):

```
     y="me and    you"
     echo $y
     echo "$y"
```

The third purpose for double quotes is that they can "seal in" command characters, so that these characters appear as literals. For example, typing

```
     echo today's
```

causes the system to display a greater than sign. It's waiting for further input, input that must conclude with a single quote. The command

```
     echo "today's"
```

just prints the word.

Greater Than and Double-Greater-Than

Both these commands take data from a source command and send that data to the target file as if it were standard output. If the file doesn't exist the command creates one. The difference is that greater-than always *writes* to the target file, thus erasing any data already in the file. The double-greater-than character always *appends* the data to the file.

To see these characters at work, try this command-line sequence:

First, check to see if you have a file named "doc." If you do, use a different file name.

```
     ls doc
     date > doc
     cat doc
     who am i >> doc
     cat doc
     cal 11 1993 > doc
     cat doc
```

Less Than

The less than (<) causes a file to be treated as a standard input for the command. The example writes the file names in the current directory into doc, It then hands the contents of doc to the sort command for a reverse sort to the screen

```
     ls > doc
     sort -r < doc
```

Parentheses

The parentheses -- ( ) -- group several commands for execution in a subshell with command output passed as standard input to the next command

on a pipeline. Notice the difference when you use parentheses and when you don't in this example:

```
date; who am i > doc
cat doc
(date; who am i) > doc
cat doc
```

Without parentheses, the shell executes the date command without sending its output to doc. The shell pipes the who am i output, because it is immediately adjacent to the greater than.

Semicolon

The semicolon (;) causes commands on the current line to be executed sequentially. Note the example for parentheses and the example below:

```
date ; ls
```

Single Quotes

Put single quotes (') around special characters when you do not want the shell to interpret them.

Note: if a back quote occurs within the command, it must be escaped with a "\" (backslash). Otherwise, the usual quoting conventions apply within the command.

Vertical Bar

The vertical bar (|) causes the standard output for the first command and to be treated as the standard input for the second command. That is, the vertical bar combines commands into a pipeline, passing data from one command to another without an intervening file. The example passes the listing for the current directory directly to the sort command for a reverse sort printed to the screen:

```
ls | sort -r
```

Article Change History:
24 Jun 1993 - Revised for technical accuracy.

Keywords: <None>

==========

19960215 11:05:19.00