

CSCI 274 - Intro to Linux OS

Week 5 - Wildcard, Globbing, Bash scripting and Control Signals

Mona Wade (dwade@mines.edu)

Overview

1. Wildcard
2. Globing
3. Bash Scripting
4. Control Signals

Review: Quoting

If you're simply enclosing a few words of text, it really **doesn't** matter which one you use, as they will both work exactly the same.

The difference between single and double quotes becomes more important when you're dealing with **variables on the command line**.

```
root@geekmini:/# test="this is a test"
root@geekmini:/# echo $test
this is a test
root@geekmini:/# echo "$test"
this is a test
root@geekmini:/# echo '$test'
$test
root@geekmini:/#
```

Review: Quoting

Double Quotes

- Use when you want to enclose variables or use shell expansion inside a string.
- All characters within are interpreted as regular characters except for `$` or ``` which will be expanded on the shell.

Single Quotes

- All characters within single quotes are interpreted as a string character.

Globbering

An asterisk (*) – matches one or more occurrences of any character, including no character.

Question mark (?) – represents or matches a single occurrence of any character.

Bracketed characters ([]) – matches any occurrence of character enclosed in the square brackets. It is possible to use different types of characters (alphanumeric characters): numbers, letters, other special characters etc.

Curly brackets ({ })- terms are separated by commas and each term must be the name of something or a wildcard.

You can as well negate a set of characters using the ! symbol.

Bash Scripts

A Bash script is a plain text file which contains a series of commands. These commands are a mixture of commands we would normally type ourselves on the command line (such as `ls` or `cp` for example) and commands we could type on the command line but generally wouldn't (you'll discover these over the next few pages). An important point to remember though is:

Anything you can run normally on the command line can be put into a script and it will do exactly the same thing. Similarly, anything you can put into a script can also be run normally on the command line and it will do exactly the same thing.

Bash Scripts

Running (aka executing) a Bash script is fairly easy. Script must have the **execute permission** set. If you forget to grant this permission before running the script you'll just get an error message telling you as such and no harm will be done.

Terminal

```
1. user@bash: ./myscript.sh
2. bash: ./myscript.sh: Permission denied
3. user@bash: ls -l myscript.sh
4. -rw-r--r-- 18 ryan users 4096 Feb 17 09:12 myscript.sh
5. user@bash: chmod 755 myscript.sh
6. user@bash: ls -l myscript.sh
7. -rwxr-xr-x 18 ryan users 4096 Feb 17 09:12 myscript.sh
8. user@bash: ./myscript.sh
9. Hello World!
10. user@bash:
```

Control Signals

Ctrl + C = Interrupt/Kill whatever you are running (SIGINT)

Ctrl + L = Clear the screen

Ctrl + s = Stop output to the screen (for long running verbose commands; then use PgUp/PgDn for navigation)

Ctrl + q = Allow output to the screen (if previously stopped using command above)

Ctrl + D = Send an EOF marker, unless disabled by an option, this will close the current shell (EXIT)

Ctrl + Z = Send the signal SIGTSTP to the current task, which suspends it; to return to it later enter fg 'process name' (foreground).