CS2044 - Advanced Unix Tools & Scripting

Spring 2011

Hussam Abu-Libdeh
based on slides by David Slater

February 23, 2011

Agenda

Continue with the review.

Using Bash Efficiently

Now lets talk about how bash makes life easier.

Tab Completion

You can use the Tab key to auto-complete commands, parameters, and file and directory names. If there are multiple choices based on what you've typed so far, Bash will list them all!

Shell Expansion

In a bash shell, if we type:

```
$ echo This is a test
This is a test
```

But if we type

```
$ echo *
Lec1.pdf Lec1.dvi Lec1.tex Lec1.aux
```

What happened?

Shell Expansion

In a bash shell, if we type:

```
$ echo This is a test
This is a test
```

But if we type

```
$ echo *
Lec1.pdf Lec1.dvi Lec1.tex Lec1.aux
```

What happened?

The shell expanded `*` to all files in the current directory. This is an example of path expansion, one type of shell expansion.

Interpreting Special Characters

The following are special characters:

$ * < > & ? { } [ ]

- The shell interprets them in a special way unless we escape (\$) or place them in quotes "$".
- When we first invoke a command, the shell first translates it from a string of characters to a UNIX command that it understands.
- A shell's ability to interpret and expand commands is one of the powers of shell scripting.

Shell Expansions

The shell interprets $ in a special way.

- If var is a variable, then `$var` is the value stored in the variable var.
- If `cmd` is a command, then `$(cmd)` is translated to the result of the command `cmd`.

```
hussam@rumman:~$ echo $USER
hussam
hussam@rumman:~$ echo $(pwd)
/home/hussam
```

Shell Expansions

`*` `^` `?` `{` `}` `[` `]` Are all "wildcard" characters that the shell uses to match:

- Any string
- A single character
- A phrase
- A restricted set of characters

Shell Expansions

- `*` matches any string, including the null string (i.e. 0 or more characters).

Examples:

Input	Matched	Not Matched
`Lec*`	`Lecture1.pdf Lec.avi`	`ALecBaldwin/`
`L*ure*`	`Lecture2.pdf Lectures/`	`sure.txt`
`*.tex`	`Lecture1.tex Presentation.tex`	`tex/`

Shell Expansions

- ? matches a single character

Examples:

Input	Matched	Not Matched
`Lecture?.pdf`	`Lecture1.pdf Lecture2.pdf`	`Lecture11.pdf`
`ca?`	`cat can cap`	`ca cake`

Shell Expansions

- `[...]` matches any character inside the square brackets
  - Use a dash to indicate a range of characters
  - Can put commas between characters/ranges

Examples:

Input	Matched	Not Matched
`[SL]ec*`	`Lecture Section`	`Vector.tex`
`Day[1-4].pdf`	`Day1.pdf Day2.pdf`	`Day5.pdf`
`[A-Z,a-z][0-9].mp3`	`A9.mp3 z4.mp3`	`Bz2.mp3 9a.mp3`

Shell Expansions

- `[^...]` matches any character **not** inside the square brackets

Examples:

Input	Matched	Not Matched
`[^A-P]ec*`	`Section.pdf`	`Lecture.pdf`
`[^A-Za-z]*`	`90210 9Days.avi .bash_profile`	`vacation.jpg`

Shell Expansions

- **Brace Expansion:** `{...,...}` matches any phrase inside the comma-separated brackets

Examples:

Input	Matched
`{Hello,Goodbye}\ World`	`Hello World Goodbye World`

NOTE: brace expansion must have a list of patterns to choose from (i.e. at least two options)

Shell Expansions

And of course, we can use them together:

Input	Matched	Not Matched
`*i[a-z]e*`	`gift_ideas profile.doc ice`	`DriVer.exe`
`[bf][ao][ro].mp?`	`bar.mp3 foo.mpg far.mp4`	`foo.mpeg`

Quoting

If we want the shell to not interpret special characters we can use quotes:

- Single Quotes (’): No special characters are evaluated
- Double Quotes (“): Variable and command substitution is performed
- Back Quotes (`): Execute the command within the quotes

Example

```
hussam@rumman:~$ echo "$USER owes me $ 1.00"
hussam owes me $ 1.00
hussam@rumman:~$ echo '$USER owes me $ 1.00'
$USER owes me $ 1.00
hussam@rumman:~$ echo "I am $USER and today is
`date`"+
I am hussam and today is Tue Feb 22 16:23:30 EST 2011
```

Text processing tools

Some often used tools for reading and manipulating text:

- Printing files: `cat` , `more` , `less`
- Get the gist of a file: `head` , `tail`
- Reordering: `sort`
- Counting: `wc` , `uniq`
- Simple character translation: `tr`
- Pattern matching: `grep`
- File/stream editing: `sed`
- File/stream processing: `gawk` (`awk` on BSD / OS X)

`cat, more, less, head, tail`

- `cat file1 file2` - print the contents of file1 then file2 to stdout
- `more file1` - print the contents of file1 page by page to stdout
- `less file1` - just like `more` but with ability to scroll up/down
- `head -n 20 file` - prints the first 20 lines (default is 10)
- `tail file` - prints the last 10 lines of file

sort

Sorts the lines of a text file alphabetically.

- `sort -r u file`
  - sorts the file in reverse order and deletes duplicate lines.
- `sort -n -k 2 -t : file`
  - sorts the file numerically by using the second column, separated by a colon

Consider a file (`numbers.txt`) with the numbers 1, 5, 8, 11, 62 each on a separate line, then:

```
$ sort numbers.txt
1
11
5
62
8
```

```
$ sort numbers.txt -n
1
5
8
11
62
```

`wc`

- How many lines of code are in my new awesome program?
- How many words are in this document?
- Good for bragging rights

- `wc -l` : count the number of lines
- `wc -w` : count the number of words
- `wc -m` : count the number of characters
- `wc -c` : count the number of bytes

uniq

- `uniq file` - Discards all but one of successive identical lines
- `uniq -c file` - Prints the number of successive identical lines next to each line

tr

The Translate Command

`tr [options] <set1> [set2]`

- Translate or delete characters
- Sets are strings of characters
- By default, searches for strings matching set1 and replaces them with set2

Example:

`echo somefile | tr 'AEIOU' 'aeiou'` - changes all capital vowels to lower case vowels

Redirection revisited

- `tr` only receives input from *standard input* (stdin)
  - i.e. keyboard input
- What if we want to operate on files?
  1. Piping: `cat somefile | tr 'AEIOU' 'aeiou'`
  2. Input redirection: `tr 'AEIOU' 'aeiou' < somefile`

Redirection revisited

Input/Output Streams

- #0 : Standard input stream; STDIN (usually keyboard but can be redirected)
  - to redirect standard input, use the < operator
  - `command < file`
- #1 : Standard output stream; STDOUT (usually terminal console but can be redirected)
  - to redirect standard output, use the > operator
  - `command > file`
- #2 : Standard error stream; STDERR (depends on system setting, can also be redirected)
  - to redirect standard error, use the > operator and specify the stream by number (2)
  - `command 2> file`

Can combine two streams together by using `2>&1`

This says: send standard error to where standard output is going.

Useful for debugging/catching error messages.

Redirection revisited

Bash processes I/O redirection from left to right, allowing us to do fun things like this:

Example:

Let's delete everything but the numbers from test1.txt, then store them in test2.txt

- `tr -cd '0-9' < test1.txt > test2.txt`

Some Simple Examples

Example:

`echo *` prints everything in the directory, separated by spaces. Let's separate them by newlines instead:

- `echo * | tr ' ' '\n'` – replaces all spaces with newlines

Example:

Let's print a file in all uppercase:

- `tr 'a-z' 'A-Z' < test.txt` - prints the contents of text.txt in all caps

tr and sets

- If the two sets passed to tr are the same length then the first character in the first set is replaced by the first in the second and so on.
- If the second is shorter than the first, then they are matched and all extra are changed to the last character
- If the first is shorter, then only those corresponding characters in the second matter

Example:

```
echo "abcdefghijklmnopqrstuvwxyz" | tr 'a-z' 'a'
aaaaaaaaaaaaaaaaaaaaaaaaaa

echo "abcdefghjiklmnopqrstuvwxyz" | tr 'a-z' 'wxyz'
wxyzzzzzzzzzzzzzzzzzzzzzz
```

tr options

tr has some very useful options:

tr options

- `tr -d <set>` - delete all characters that are in the set
- `tr -c <set1> [set2]` - complements set1 before replacing it with set2

Example:

Lets print a file with all non-letters removed:

- `tr -cd 'a-zA-Z' < somefile` - removes non-letters from somefile

tr and character ranges

tr does not understand regular expressions (and really for the task it is designed for they don't make sense), but it **does** understand characte ranges and POSIX character sets such as `[:alpha:]`

Reminder:

- `[:alnum:]` - alphanumeric characters
- `[:alpha:]` - alphabetic characters
- `[:digit:]` - digits
- `[:punct:]` - punctuation characters
- `[:lower:]` - lowercase letters
- `[:upper:]` - uppercase letters
- `[:space:]` - whitespace characters

Substitution Cypher

We can use `tr` to do a simple substitution cypher. Create a text file called "cypher" with some reordering of the alphabet. Then to encode we would just do

- `tr 'a-z' 'cat cypher' < file > encodedfile`

and to decode we would do

- `tr 'cat cypher' 'a-z' < encodedfile > decodedfile`

Note the use of backticks around `cat cypher`. By doing this the shell expands this command and passes the output to `tr`.