CS2044 - Advanced Unix Tools & Scripting

Spring 2011

Hussam Abu-Libdeh
Today's slides are from David Slater

February 25, 2011

Random Bash Tip of the Day

The more you use BASH the more you see what options you use all the time. For instance `ls -l` to see permissions, or `rm -i` to insure you don't accidentally delete a file. Wouldn't it be nice to be able to make shortcuts for these things?

Alias:

`alias name=command`

- The alias allows you to rename or type something simple instead of typing a long command
- You can set an alias for your current session at the command prompt
- To set an alias more permenantly add it to your `.bashrc` or `.bash_profile` file.

Alias Examples

Examples

```
alias ls='ls --color=auto'
alias dc=cd
alias rm='rm -i'
alias ll='ls -l'  alias canhaz='sudo apt-get install'
```

- Quotes are necessary if the string being aliased is more than one word
- To see what aliases are active simply type `alias`
- Note: If you are poking around in .bashrc you should know that # is the UNIX comment character. So any line that starts with # is commented out.

A Bit of Translation

What does this print?

```
echo '1337 5p34k !5 n07 5p0k3n 4m0n9 2341 h4ck325' |
tr '01234579!'  'olreastgi'
```

A Bit of Translation

What does this print?

```
echo '1337 5p34k !5 n07 5p0k3n 4m0n9 2341 h4ck325' |
tr '01234579!'  'olreastgi'
```

```
leet speak is not spoken among real hackers
```

Variables

- Variables are denoted by `$varname`, and set by

  `varname=value`

- No spaces!

  Example:

```
$ world=Earth
$ echo "Yo $world"
Yo Earth
```

- There are a ton of built-in variables ($USER $SHELL ... etc)

grep

The purpose of `grep` is to print the lines that match a particular pattern.

`grep password file` - prints all lines that contain the word `password`.

How many lines contain the word monster in Frankenstein?

```
grep '[Mm]onster' Frankenstein.txt | wc -l
33
```

Grep options

- `grep -i` - ignores case
- `grep -A 20 -B 10` - prints the 10 lines before and 20 lines after each match
- `grep -v` - inverts the match
- `grep -o` - shows only the matched substring
- `grep -n` - displays the line number

A simple script

```
#!/bin/bash
tr ' ' '\n' $1 | grep '^[[:upper:]]' | wc -l
```

- `$1` refers to the first argument passed to the script (more on this next time)
- `'^[[:upper:]]'` is our first example of a regular expression. Here `^` refers to the beginning of a line and `[[:upper:]]` is any uppercase letter.

Regular Expression

`grep` like many operations takes in a **regular expression** as its argument. Pattern matching with regular expressions is more sophisticated than shell expansion and also uses different syntax.

More precisely, a regular expression is a set of strings; these strings **match** the expression.

When we use regular expressions, it is (usually) best to enclose them in single quotes to stop the shell from expanding it before passing it to `grep` or other tools.

Regular Expression Rules

- `.` - matches any single character
- `*` - matches 0 or more occurences of the character immediately proceeding
- `\?` - matches 0 or 1 occurrences of the character immediately preceding
- `\+` - matches 1 or more occurrences of the character immediately preceding

`$ grep 't.a'` - prints lines with things like `tea, taa, steap`
`$ grep 't*a` - prints lines with things like `aste, taste, ttaste, tttaste`

Regular Expression Rules

- `[c1,c2]` - matches either character (comma optional)
- `^` - matches the beginning of a line
- `$` - matches the end of a line
- `e\{i,j\}` matches between `i` and `j` occurrences of strings that match `e`.

`grep -o '[0-9]\{3\}-\{0,1\}[0-9]\{2\}-\{0,1\}[0-9]\{4\}'` prints all social security numbers in a file (both `111-11-1111` and `111111111`)

Regular Expression Rules

- `c1\|c2` matches the expression c1 or the expression c2.
- `\(txt\)` groups for applying postfix operators
- `\<` matches the beginning of a word
- `\>` matches the end of a word

`grep '\(top\)\{3\}'` searches for `toptoptop`.

`grep 'top\{3\}'` searches for `toppp`.

`grep '[0-5]\{2\}\|[6-9]\{2\}'` searches for things like 12, 15, 68, 97, but not 19, 61.

A word about extended regular expressions

With extended regular expressions you do not need to escape special characters such as ?, +, () and . To use extended regular expressions with grep use the variant egrep.

Extended regular expressions tend to be cleaner and easier to read:

`grep '\(woo\+t\)\{2,3\}'` becomes `egrep '(woo+t){2,3}'`.

There is also `fgrep` which does not understand any regular expressions (fastest).

Why we quote regular expressions

Suppose we have a directory with the following files in it:

```
num, num2, test
```

Now suppose we want to search the file test for the regular expression `nu*`. If we don't quote,

```
grep nu* test
```

gets expanded to

```
grep num num2 test
```

, which searches num2 and test for the string num.

Regular Expression Examples

How would you match any word that begins with `c` and ends with `d`?

Regular Expression Examples

How would you match any word that begins with c and ends with d?

```
grep '\<c[A-Za-z]*d\>'
```

```
grep '\<c.*d\>'
```

Regular Expression Examples

How would you match any word that begins with c and ends with d?

```
grep '\<c[A-Za-z]*d\>'
```

```
grep '\<c.*d\>'
```

If we just want 5 letter words beginning with c and ending with d:

```
grep -o '\<c...d\>' /usr/share/dict/words | uniq
```

```
caged
caked
caned
caped
cared
⋮
```

Great for crosswords!

More Grep Examples

```
$ grep '^smug'
```

More Grep Examples

```
$ grep '^smug'
```

searches for lines that begin with smug

More Grep Examples

```
$ grep '^smug'
```

searches for lines that begin with smug

```
$ grep '[[:digit:]]\{5\}'
```

More Grep Examples

```
$ grep '^smug'
```

searches for lines that begin with smug

```
$ grep '[[:digit:]]\{5\}'
```

searches for zip codes

More Grep Examples

```
$ grep '^smug'
```

searches for lines that begin with smug

```
$ grep '[[:digit:]]\{5\}'
```

searches for zip codes

```
$ grep '^$'
```

More Grep Examples

```
$ grep '^smug'
```

searches for lines that begin with smug

```
$ grep '[[:digit:]]\{5\}'
```

searches for zip codes

```
$ grep '^$'
```

searches for blank lines

More Grep Examples

```
$ grep '^smug'
```

searches for lines that begin with smug

```
$ grep '[[:digit:]]\{5\}'
```

searches for zip codes

```
$ grep '^$'
```

searches for blank lines

```
$ grep 'B[oO][bB]$'
```

More Grep Examples

```
$ grep '^smug'
```

searches for lines that begin with smug

```
$ grep '[[:digit:]]\{5\}'
```

searches for zip codes

```
$ grep '^$'
```

searches for blank lines

```
$ grep 'B[oO][bB]$'
```

searches for lines that end in either BOB, Bob, BOb, or BoB.

Convenient Shortcuts for Sets of Characters

- `[[:alnum:]]` - `[A-Za-z0-9]`
- `[[:alpha:]]` - `[A-Za-z]`
- `[[:punct:]]` - Punctuation and symbols
- `[[:digit:]]` - `[0-9]`
- `[[:xdigit:]]` - hexadecimal digits [0-9A-Fa-f]

Examples:
`grep '[a-t[:digit:]]'` looks for any letter from a to t or any digit.
`[[:digit:][:alpha:]]` is equivalent to `[[:alnum:]]`

sed

sed is a *stream editor*. We will only cover the basics, as it is a completely programming language!

- `sed 's/regexp/txt/g'` - substitution

```
sed 's/not guilty/guilty/g' filename
```

What happens if we don't have the g?

sed

sed is a *stream editor*. We will only cover the basics, as it is a completely programming language!

- `sed 's/regexp/txt/g'` - substitution

```
sed 's/not guilty/guilty/g' filename
```

What happens if we don't have the g?

Without the g, it will only do one substitution per line.

- `sed '/regexp/d'` - deletion

```
sed '/[Dd]avid/d' filename > filename2
```

deletes all **lines** that contain either David or david and saves the file as `filename2`.

Why Sed?

Sed is designed to be useful especially if:

- your files are too large for comfortable interactive editing
- your sequence of editing commands is too complicated to be comfortably typed in iteractive mode
- you want to globally apply your edits in one pass through

sed understands regular expressions!

The power of sed is that it treats everything between the first pair of /'s as a regular expression. So we could do

```
sed 's/[[:alpha:]]\{1,3\}[[:digits:]]*@cornell\.edu/cornell email
removed/g' file
```

to print a file with all cornell email addresses removed.

use -r to use extended regular expressions.

sed is greedy!

sed matches a given regular expression to the **longeset** string as soon as possible:

```
$ echo filename1
a b col d e f lapse h i j k lapse m n
$ sed 's/col.*lapse/collapse/g' filename1
```

sed is greedy!

sed matches a given regular expression to the **longeset** string as soon as possible:

```
$ echo filename1
a b col d e f lapse h i j k lapse m n
$ sed 's/col.*lapse/collapse/g' filename1
a b collapse m n

$ echo filename2
a b c col d e f col g h i lapse
sed 's/col.*lapse/collapse/g' filename2
```

sed is greedy!

sed matches a given regular expression to the **longeset** string as soon as possible:

```
$ echo filename1
a b col d e f lapse h i j k lapse m n
$ sed 's/col.*lapse/collapse/g' filename1
a b collapse m n

$ echo filename2
a b c col d e f col g h i lapse
sed 's/col.*lapse/collapse/g' filename2
a b c collapse
```

sed can save the string

Another Example:

```
sed 's/^\([A-Z][A-Za-z]*\), \([A-Z][A-Za-z]*)/\2 \1/' filename
```

- Searches for an expression at the beginning of the line of the form `e1, e2` where `e1` and `e2` are "words" starting with capital letters.
- Placing an expression inside ( ) tells the editor to save whatever string matches the expression
- Since ( ) are special characters we escape them
- We access the saved strings as `\1, \2`.
- This script for example could convert a database file from

`Lastname, Firstname` to `Firstname Lastname`

More sed

You can specify which lines to check by numbers or with regular expressions:

`sed '1,20s/john/John/g' filename` - checks lines 1 to 20

`sed '/^The/s/john/John/g' filename` - checks lines that start with The

& cooresponds to the pattern found:

```
sed  's/[a-z]\+/\(&\)/g' filename
```

replaces words with words in quotes For more on sed check out

http://www.grymoire.com/Unix/Sed.html

Even more sed

How could we use sed to remove a specific regular expression?

Even more sed

How could we use sed to remove a specific regular expression? `sed 's/regexp/ /g' file`

Example:

```
sed 's/[[:alnum:]]/ /g' Frankenstein.txt
```

Other Sed Options

Besides substitution and deletion we can use sed to

- append lines
- change lines
- insert
- print lines

among other things

Appending a Line

append

`/address/a\ text` - append a line with text following each line matched by address

`/address/i\text` - append a line with text preceeding each line matched by address

Example: `sed '/[Mm]onster/a\ A monster is on the line above look out!'`

More Sed

Change

`/address/c\ text` replaces a line that contains address with text `/address1/,/address2/c\ text` replaces all lines between address1 and address2 (including them) with text **yes change can work with multiple lines**

```
sed '/[Mm]onsters\?/c\ This line deleted because monsters scare me'
Frankenstein.txt
```

```
sed '/^From /,/^$/c\ <Mail Header Removed>' / mail
```

Print

Print

`sed '/address1/,/address2/p'` prints all lines between address1 and address2
Most usefull when used with the flag -n that suppresseds normal output

`sed -n /debt/p' databasefile` - prints all lines that contain the word debt

`sed -n /begin{equation}/,/end{equation}/p' texfile` - prints all equations in your tex file

sed scripting

sed is a complete programming language and we can write sed scripts.

- **Recall:** Any file that begins with #! is a script file

Example

- Create a new text file named trim.sed

```
#! /usr/bin/sed -f
s/^ *//
s/ *$//
```

You can run this script from the shell like any other program:

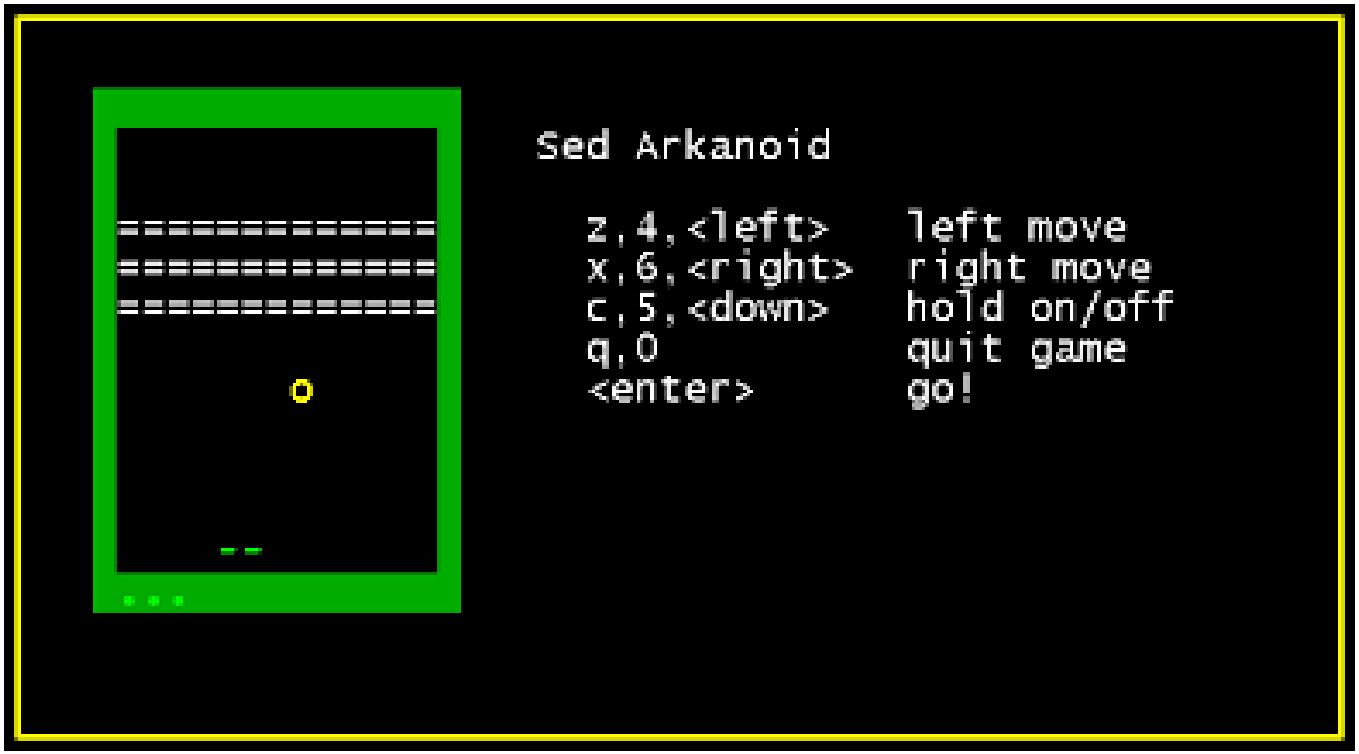
- `echo " this is a test " | ./trim.sed`

```
this is a test
```

We now have a script that trims leading and trailing whitespace!

Sed Arkanoid

Sed is a complete programming language. In fact people have written entire games as sed scripts.



http:aurelio.net/soft/sedarkanoid/