

CST8177 – Linux II

Regular Expressions

Topics

- ▶ Our standard `.bashrc` and `.bash_profile` (or `.profile`)
- ▶ Our standard script header
- ▶ Regular Expressions

.bashrc

```
[ -z "${PS1-}" ] && return
if [ "${_FIRST_SHELL-}" = "" ] ; then
    export _FIRST_SHELL=$$
    export PATH="$PATH:$HOME/bin"
    export LC_ALL=en_CA.UTF-8
    export LANG=en_CA.UTF-8
    # here we put things that
    # should be done once
fi
# here we put things that need to be
# done for every interactive shell
```

.bash_profile

Contains just one line:

```
source ./ .bashrc
```

Standard script header

```
#!/bin/sh -u  
PATH=/bin:/usr/bin ; export PATH    # add /sbin and /usr/sbin if needed  
umask 022                          # use 077 for secure scripts
```

Matching Patterns

- ▶ There are two different pattern matching facilities that we use in Unix/Linux:
 1. filename globbing patterns match existing pathnames in the current filesystem only
 2. regular expressions match substrings in arbitrary input text

- ▶ We need to pay close attention to which of the two situations we're in, because some of the same special characters have different meanings!

File Name Globbing

- ▶ Globbing is used for
 - globbing patterns in command lines
 - patterns used with the `find` command
- ▶ shell command line (the shell will match the patterns against the file system):
 - `ls *.txt`
 - `echo ??????.txt`
 - `vi [ab]*.txt`
- ▶ `find` command (we double quote the pattern so the `find` command sees the pattern, not the shell):
 - `find ~ -name "*.txt"`
 - in this case, the `find` command matches the pattern against the file system

Regular Expressions

- ▶ **IMPORTANT:** regular expressions use some of the same special characters as filename matching on the previous slide but they mean different things!
- ▶ Before we look at regular expressions, let's take a look at some expressions you're already comfortable with: algebraic expressions
- ▶ Larger algebraic expressions are formed by putting smaller expressions together

Algebraic Expressions

Expression	Meaning	Comment
a	a	a simple expression
b	b	another simple expression
ab	$a \times b$	ab is a larger expression formed from two smaller ones concatenating two expressions together means to multiply them
b^2	$b \times b$	we might have represented this with b^2 , using $^$ as an exponentiation operator
ab^2	$a \times (b \times b)$	why not $(a \times b) \times (a \times b)$?
$(ab)^2$	$(a \times b) \times (a \times b)$	

Basic Regular Expressions

Expression	Meaning	Comment
a	match single 'a'	a simple expression
b	match single 'b'	another simple expression
ab	match strings consisting of single 'a' followed by single 'b'	"ab" is a larger expression formed from two smaller ones concatenating two regular expressions together means "followed immediately by" and we'll say "followed by"
b*	match zero or more 'b' characters	a big difference in meaning from the '*' in globbing! This is the regular expression repetition operator.
ab*	'a' followed by zero or more 'b' characters	why not repeating ('a' followed by 'b'), zero or more times? Hint: think of "ab ² " in algebra.
\(ab\)*	('a' followed by 'b'), zero or more times	We can use parenthesis, but in Basic Regular Expressions, we use \ <code>(</code> and \ <code>)</code>

Basic Regular Expressions (con't)

Expression	Matches	Ex.	Example Matches	Comment
non-special character	itself	x	"x"	like globbing
one expression followed by another	first followed by second	xy	"xy"	like globbing
.	any single character	.	"x" or "y" or "!" or "." or "*" ...etc	like the '?' in globbing
expression followed by *	zero or more matches of the expression	x*	"" or "x" or "xx" or "xxx" ...etc	NOT like the * in globbing, although .* behaves like * in globbing
character classes	a SINGLE character from the list	[abc]	"a" or "b" or "c"	like globbing

Basic Regular Expressions (con't)

Expression	Matches	Ex.	Example Matches	Comment
<code>^</code>	beginning of a line of text	<code>^x</code>	"x" if it's the first character on the line	anchors the match to the beginning of a line
<code>\$</code>	end of a line of text	<code>x\$</code>	"x" if it's the last character on the line	anchors the match to the end of a line
<code>^</code> (but not first)	<code>^</code>	<code>a^b</code>	"a^b"	<code>^</code> has no special meaning unless its first
<code>\$</code> (but not last)	<code>\$</code>	<code>a\$b</code>	"a\$b"	<code>\$</code> has no special meaning unless its last

Basic Regular Expressions (con't)

Expression	Matches	Ex.	Example Matches	Comment
special character inside [and]	as if the character is not special	<code>[\\]</code>	<code>"\"</code>	conditions: <code>']'</code> must be first, <code>'^'</code> must not be first, and <code>'-'</code> must be last
<code>\</code> followed by a special character	that character with its special meaning removed	<code>\.</code>	<code>"."</code>	like globbing
<code>\</code> followed by non-special character	the non-special character	<code>\a</code>	<code>"a"</code>	<code>\</code> before a non-special character is ignored

Exploring Regular Expressions

- ▶ testing regular expressions with grep on stdin
 - run `grep --color=auto 'expr'`
 - use single quotes to protect your *expr* from the shell
 - grep will wait for you to repeatedly enter your test strings (type `^D` to finish)
 - grep will print any string that matches your *expr*, so each matched string will appear twice (once when you type it, and once when grep prints it)
 - the part of the string that matched will be colored
 - unmatched strings will appear only once where you typed them

Basic Regular Expressions (cont'd)

- ▶ Regular expressions can be used in awk, grep, vi, sed, more, less, and others
- ▶ For now, we'll use grep on the command line
- ▶ We will get into the habit of putting our regex in single quotes on the command line to protect the regex from the shell
- ▶ Special characters for basic regular expressions: `\`, `[`, `]`, `.`, `*`, `^`, `$`
- ▶ can match single quote by using double quotes, as in: `grep "I said, \"don't\""`
- ▶ alternatively: `grep 'I said, "don\''t'''`

Regular Expressions

- ▶ Appendix A in the Sobell Text book is a source of information
- ▶ You can read under `REGULAR EXPRESSIONS` in the man page for the `grep` command – this tells you what you need to know
- ▶ The `grep` man page is normally available on Unix systems, so you can use it to refresh your memory, even years from now

Regular Expressions to test

- ▶ examples (try these)
 - `grep 'ab'` #any string with **a** followed by **b**
 - `grep 'aa*b'` #one or more **a** followed by **b**
 - `grep 'a.*b'` #**a**, then one or more anything, then **b**
 - `grep 'a.*b'` #**a** then zero or more anything, then **b**
 - `grep 'a.b'` # **a** then exactly one anything, then **b**
 - `grep '^a'` # **a** must be the first character
 - `grep '^a.*b$'` # **a** must be first, **b** must be last
- ▶ Try other examples: have fun!