CST8177 – Linux II

More Scripting and Regular Expressions

Todd Kelley
kelleyt@algonquincollege.com

Today's Topics

- lynda.com
- stty (pending from last week)
- .bashrc versus .bash_profile
- More shell scripting
- Regular Expression examples

Lynda.com

- Some students are already comfortable with the command line
- For those who aren't, yet another tutorial source that might help is Lynda.com
- All Algonquin students have free access to Lynda.com
- Unix for Mac OSX users:

http://www.lynda.com/Mac-OS-X-10-6-tutorials/Unix-for-Mac-OS-X-Users/78546-2.html

.bashrc versus .bash_profile

- .bash_profile is loaded once by a login shell
- .bashrc is loaded by non-login shells
- There are cases where there never is a login shell
  - ssh remote-server.com <some_command>
- So the method we'll use in this course:
  - `.bash_profile` does nothing except load .bashrc
  - `.bashrc` keeps track of things that should be done only once

.bashrc

```
[ -z "$PS1" ] && return
if [ "${_FIRST_SHELL-}" = "" ] ; then
    export _FIRST_SHELL=$$
    PATH=$PATH:$HOME/bin
    # 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:

```
[ -f $HOME/.bashrc ] && . $HOME/.bashrc
```

Or equivalently, these three lines instead

```
if [ -f $HOME/.bashrc ]; then
    . $HOME/.bashrc
fi
```

Shell scripting

- For the impatient, you can read ahead

http://elearning.algonquincollege.com/coursemat/alleni/idallen/cst8177/13w/notes/000_script_style.html

- From now on, at the top of all our shell scripts, we put

```
#!/bin/sh -u
# UTF-8 (international) script header
PATH=/bin:/usr/bin ; export PATH
umask 022
unset LC_ALL                             # unset the over-ride
variable
LC_COLLATE=en_US.utf8 ; export LC_COLLATE # sort by character
set
LC_CTYPE=en_US.utf8 ; export LC_CTYPE # handle multi-byte chars
LANG=en_US.utf8 ; export LANG          # legacy version of LC_CTYPE
```

Internationalization (i18n)

- http://teaching.idallen.com/cst8177/13w/notes/000_character_sets.html

- Not all computer users use the same alphabet
- When we write a shell script, we need to ensure that it handles text properly in the presence of i18n
- In the beginning, there was ascii, a 7 bit code of 128 characters
- Now there's Unicode, a table that is meant to assign an integer to every character in the world
- UTF-8 is an implementation of that table, encoding the 7-bit ascii characters in a single byte with high order bit of 0
- The 128 single-byte UTF-8 characters are the same as true ascii bytes (both have a high order bit of 0)
- UTF-8 characters that are not ascii occupy more than one byte
- Locale settings determine how characters are interpreted and treated, whether as ascii or UTF-8, their ordering, and so on

What is locale

- A locale is the definition of the subset of a user's environment that depends on language and cultural conventions.
- It is made up from one or more categories. Each category is identified by its name and controls specific aspects of the behavior of components of the system.
- Category names correspond to the following environment variable names (we deal with just the first two in our shell scripts):
  - *LC_CTYPE*: Character classification and case conversion.
  - *LC_COLLATE*: Collation order.
  - *LC_MONETARY*: Monetary formatting.
  - *LC_NUMERIC*: Numeric, non-monetary formatting.
  - *LC_TIME*: Date and time formats.
  - *LC_MESSAGES*: Formats of informative and diagnostic messages and interactive responses.

Regular Expressions (again)

- Three kinds of matching
  1. Filename globbing
     - used on shell command line, and shell matches these patterns to filenames that exist
     - used with the `find` command
  2. Regular expressions, used with
     - vi
     - sed
     - awk
     - grep
  3. Extended regular expressions
     - egrep or grep -E (not emphasized in this course)
     - perl regular expressions (not in this course)

Testing Regular Expressions

- testing regular expressons with grep on stdin
  - run grep 'expr' on the standard input
  - use the 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)
  - unmatched strings will appear only once where you typed them
  - type ^D to finish

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
- Let's try some in vi and awk