CS35L – Fall 2018

Slide set:	2.2
Slide topics:	Shell scripting, regex, streams
Assignment:	2

Variables

- Declared using =
  - var=“hello” #NO SPACES!!!
- Referenced using $
  - echo $var

```
#!/bin/sh
        message="HELLO WORLD!!!"
echo $message
```

POSIX Built-in Shell Variables

Variable	Meaning
#	Number of arguments given to current process.
@	Command-line arguments to current process. Inside double quotes, expands to individual arguments.
*	Command-line arguments to current process. Inside double quotes, expands to a single argument.
- (hyphen)	Options given to shell on invocation.
?	Exit status of previous command.
$	Process ID of shell process.
0 (zero)	The name of the shell program.
!	Process ID of last background command. Use this to save process ID numbers for later use with the *wait* command.
ENV	Used only by interactive shells upon invocation; the value of $ENV is parameter-expanded. The result should be a full pathname for a file to be read and executed at startup. This is an XSI requirement.
HOME	Home (login) directory.
IFS	Internal field separator; i.e., the list of characters that act as word separators. Normally set to space, tab, and newline.
LANG	Default name of current locale; overridden by the other LC_* variables.
LC_ALL	Name of current locale; overrides LANG and the other LC_* variables.
LC_COLLATE	Name of current locale for character collation (sorting) purposes.
LC_CTYPE	Name of current locale for character class determination during pattern matching.
LC_MESSAGES	Name of current language for output messages.
LINENO	Line number in script or function of the line that just ran.
NLSPATH	The location of message catalogs for messages in the language given by $LC_MESSAGES (XSI).
PATH	Search path for commands.
PPID	Process ID of parent process.
PS1	Primary command prompt string. Default is "$ ".
PS2	Prompt string for line continuations. Default is "> ".
PS4	Prompt string for execution tracing with set -x. Default is "+ ".
PWD	Current working directory.

Exit: Return value

Check exit status of last command that ran with $?

Value	Typical/Conventional Meaning
0	Command exited successfully.
> 0	Failure to execute command.
1-125	Command exited unsuccessfully. The meanings of particular exit values are defined by each individual command.
126	Command found, but file was not executable.
127	Command not found.
> 128	Command died due to receiving a signal

Accessing Arguments

- Positional parameters represent a shell script's command-line arguments
- For historical reasons, enclose the number in braces if it's greater than 9

```
#! /bin/sh
#test script
echo first arg is $1
#echo tenth arg is ${10}

./test hello
first arg is hello
```

Quotes-Exercise

```
# a=pwd
# echo ‘$a’
# echo “$a”
# echo `$a`
```

Quotes

- Three kinds of quotes
  - Single quotes ' '
    - Do not expand at all, literal meaning
      - Try temp='$hello$hello' ; echo $temp
  - Double quotes " "
    - Almost like single quotes but expand backticks and $
  - Backticks ` ` or $()
    - Expand as shell commands
      - Try temp=`ls` ; echo $temp

if Statements

- If statements use the **test** command or [ ]
- “man test” to see the expressions that can be done

```
#!/bin/bash
if [ 5 –gt 1 ]
then
        echo “5 greater than 1"
else
        echo “error"
fi
```

- Condition for less than or equal??

Conditional

- if…then…fi
- if…then…else…fi
- if…then…elif..then…fi
- case…esac

```
#!/bin/sh

a=10
b=20

if [ $a == $b ]
then
   echo "a is equal to b"
elif [ $a -gt $b ]
then
   echo "a is greater than b"
elif [ $a -lt $b ]
then
   echo "a is less than b"
else
   echo "None of the condition met"
fi
```

Unconditional

-
-

```
#!/bin/sh

FRUIT="kiwi"

case "$FRUIT" in
   "apple") echo "Apple pie is quite tasty."
   ;;
   "banana") echo "I like banana nut bread."
   ;;
   "kiwi") echo "New Zealand is famous for kiwi."
   ;;
esac
```

Loops

- **While loop**

```
#!/bin/sh
COUNT=6
while [ $COUNT -gt 0 ]
do
        echo “Value of count is: $COUNT”
        let COUNT=COUNT-1
done
```

  - The “let” command is used to do arithmetic

- **For loop**

```
#!/bin/sh
temp=`ls`
for f in $temp
do
                echo $f
done
```

  - f will refer to each word in `ls` output

Regular Expressions

Regular Expressions

- Notation that lets you search for text with a particular pattern:
  - For example: starts with the letter a, ends with three uppercase letters, etc.

- http://regexpal.com/ to test your regex expressions

Regular expressions

Character	BRE / ERE	Meaning in a pattern
\	Both	Usually, turn off the special meaning of the following character. Occasionally, enable a special meaning for the following character, such as for \(...\) and \{...\}.
.	Both	Match any single character except NULL. Individual programs may also disallow matching newline.
*	Both	Match any number (or none) of the single character that immediately precedes it. For EREs, the preceding character can instead be a regular expression. For example, since . (dot) means any character, .* means "match any number of any character." For BREs, * is not special if it's the first character of a regular expression.
^	Both	Match the following regular expression at the beginning of the line or string. BRE: special only at the beginning of a regular expression. ERE: special everywhere.

Regular Expressions (cont'd)

$	Both	Match the preceding regular expression at the end of the line or string. BRE: special only at the end of a regular expression. ERE: special everywhere.
[...]	Both	Termed a bracket expression, this matches any one of the enclosed characters. A hyphen (-) indicates a range of consecutive characters. (Caution: ranges are locale-sensitive, and thus not portable.) A circumflex (^) as the first character in the brackets reverses the sense: it matches any one character not in the list. A hyphen or close bracket (]) as the first character is treated as a member of the list. All other metacharacters are treated as members of the list (i.e., literally). Bracket expressions may contain collating symbols, equivalence classes, and character classes (described shortly).
\{*n*,*m*\}	BRE	Termed an *interval expression*, this matches a range of occurrences of the single character that immediately precedes it. \{*n*\} matches exactly n occurrences, \{*n*,\} matches at least n occurrences, and \{*n*,*m*\} matches any number of occurrences between n and m. n and m must be between 0 and RE_DUP_MAX (minimum value: 255), inclusive.
\(\)	BRE	Save the pattern enclosed between \(and \) in a special *holding space*. Up to nine subpatterns can be saved on a single pattern. The text matched by the subpatterns can be reused later in the same pattern, by the escape sequences \1 to \9. For example, **\(ab\).*\1** matches two occurrences of ab, with any number of characters in between.

Examples

Expression	Matches
tolstoy	The seven letters tolstoy, anywhere on a line
^tolstoy	The seven letters tolstoy, at the beginning of a line
tolstoy$	The seven letters tolstoy, at the end of a line
^tolstoy$	A line containing exactly the seven letters tolstoy, and nothing else
[Tt]olstoy	Either the seven letters Tolstoy, or the seven letters tolstoy, anywhere on a line
tol.toy	The three letters tol, any character, and the three letters toy, anywhere on a line
tol.*toy	The three letters tol, any sequence of zero or more characters, and the three letters toy, anywhere on a line (e.g., toltoy, tolstoy, tolWHOtoy, and so on)

Regular Expressions (cont'd)

\n	BRE	Replay the nth subpattern enclosed in \(and \) into the pattern at this point. n is a number from 1 to 9, with 1 starting on the left.
{n,m}	ERE	Just like the BRE \{*n*,*m*\} earlier, but without the backslashes in front of the braces.
+	ERE	Match one or more instances of the preceding regular expression.
?	ERE	Match zero or one instances of the preceding regular expression.
|	ERE	Match the regular expression specified before or after.
( )	ERE	Apply a match to the enclosed group of regular expressions.

Matching Multiple Characters with One Expression

*	Match zero or more of the preceding character
\{*n*\}	Exactly n occurrences of the preceding regular expression
\{*n*,\}	At least n occurrences of the preceding regular expression
\{*n*,*m*\}	Between n and m occurrences of the preceding regular expression

POSIX Bracket Expressions

Class	Matching characters	Class	Matching characters
[:alnum:]	Alphanumeric characters	**[:lower:]**	**Lowercase characters**
[:alpha:]	Alphabetic characters	[:print:]	Printable characters
[:blank:]	**Space and tab characters**	[:punct:]	Punctuation characters
[:cntrl:]	Control characters	[:space:]	Whitespace characters
[:digit:]	Numeric characters	**[:upper:]**	**Uppercase characters**
[:graph:]	Nonspace characters	[:xdigit:]	Hexadecimal digits

Searching for Text

- grep: Uses basic regular expressions (BRE)

  "meta-characters ?, +, {, |, (, and) lose their special meaning; instead use the backslashed versions" –`man grep`

- egrep (or grep -E): Uses extended regular expressions (ERE) – no backslashes needed
- fgrep (or grep -F): Matches fixed strings instead of regular expressions.

Simple grep

```
$ who                                                    Who is logged on
tolstoy tty1 Feb 26 10:53
tolstoy pts/0 Feb 29 10:59
tolstoy pts/1 Feb 29 10:59
tolstoy pts/2 Feb 29 11:00
tolstoy pts/3 Feb 29 11:00
tolstoy pts/4 Feb 29 11:00
austen pts/5 Feb 29 15:39 (mansfield-park.example.com)
austen pts/6 Feb 29 15:39 (mansfield-park.example.com)

$ who | grep -F austen                        Where is austen logged on?
austen pts/5 Feb 29 15:39 (mansfield-park.example.com)
austen pts/6 Feb 29 15:39 (mansfield-park.example.com)
```

sed (stream editor)

- Now you can extract, but what if you want to replace parts of text?
- Use sed!

`sed 's/`***`regExpr`***`/`***`replText`***`/[g]'`

- Example
  - Display the first directory in PATH
  - `echo $PATH | sed 's/:.*//'`

Remove everything after and including the first colon

Regular Expression Review

4 Basic Concepts

- Quantification
  - How many times of previous expression?
  - Most common quantifiers: ?(0 or 1), *(0 or more), +(1 or more)
- Grouping
  - Which subset of previous expression?
  - Grouping operator: ()
- Alternation
  - Which choices?
  - Operators: [] and |
    - Hello|World [A B C]
- Anchors
  - Where?
  - Characters: ^ (beginning) and $ (end)

RegEx Exercises

- Which of the following strings would match the regular expression: aab?b
  - A. aabb
  - B. aa\nbbb
  - C. aab

Answer: aabb

aab

RegEx Exercises

- Which regular expression would match the words “favorite” and “favourite”?
  - Answer: “favou?rite”

RegEx Exercises

- Which regular expression would match the words “Ggle”, “Gogle” and “Google”?
  - Answer: “Go*gle”

- Which one would match “Gogle”, “Google” and “Gooogle” but not “Ggle”?
  - Answer: “Go+gle”

RegEx Exercises

- Which regular expression would match any version of the word “Google” that has an even number of o’s?
  - Answer: “G(oo)+gle”
- Which regular expression would match any version of the word “Google” that has fewer than 7 O’s?
  - Answer: “Go{0,6}gle”

RegEx Exercises

- Which line(s) would this regular expression match? “^T.+e$”
  - A. The
  - B. Te
  - C. Three
  - D. Then

  Answer: The, Three (ERE)

RegEx Exercises

- Which regular expression(s) would match the words “Ted”, “Ned” and “Sed”?
  - A. (T|N|S)ed
  - B. [T N S]ed
  - C. .ed
  - D. [L-U]?ed
  - E. *ed

  Answer: A., B., C.,

  D., E. (ERE)

RegEx Exercises

- Which regular expression would match all subdirectories within a directory?
  - Answer: ls –l | grep “^d”

Lab 2

What is Lab 2 About?

Build a spelling checker for the Hawaiian language
(Get familiar with sort, comm and tr commands!)

- Steps:
    1. Download a copy of web page containing basic English-to-Hawaiian dictionary
    2. Extract only the Hawaiian words from the web page to build a simple Hawaiian dictionary. Save it to a file called hwords (**site scraping**)
    3. Automate site scraping: `buildwords` script (cat hwnwdseng.htm | buildwords > hwords)
    4. Modify the command in the lab assignment to act as a spelling checker for Hawaiian
    5. Use your spelling checker to check hwords and the lab web page for spelling mistakes

Useful Text Processing Tools

- wc: outputs a one-line report of lines, words, and bytes
- head: extract top of files
- tail: extracts bottom of files
- tr: translate or delete characters
- grep: print lines matching a pattern
- sort: sort lines of text files
- sed: filtering and transforming text

Lab Hints

- Run your script on seasnet servers before submitting to CCLE
- sed '/patternstart/,/patternstop/d‘
  - delete patternstart to patternstop, works across multiple lines
    will delete all lines starting with patternstart to patternstop
- The Hawaiian words html page uses \r and \n for new lines
  - od –c hwnwdseng.htm to see the ASCII characters
- You can delete blank white spaces such as tab or space using
  - tr -d '[:blank:]’
  - Use tr -s to squeeze multiple new lines into one
- sed 's/<[^>]*>//g' a.html to remove all HTML tags

Assignment 2 Details

- Submit 3 files:
  - Script “buildwords”
  - Simple text file “lab2.log”
    - 80 character limit per row
  - Script “sameln”
- Check everything on SEASnet!
  - Assignments graded on SEASnet servers (lnxsrv07)

Buildwords

- Hawaiian.html -> buildwords -> hwords
- Buildwords
  - Read from STDIN and perform work on input
  - Output to STDOUT
- Ex:
  - $ ./buildwords < hawaiian.html > hwords

Lab2.log

- .log is the same as .txt – no difference
- Ex:
  - 1. I used wget to download the webpage
  - 2. I ….
  - 3. Answer to #3 here
- Should read basically like a lab journal
- Keep things concise!

Lab Hints

- Run your script on seasnet servers before submitting to CCLE
- sed '/patternstart/,/patternstop/d‘
  - delete patternstart to patternstop, works across multiple lines
    will delete all lines starting with patternstart to patternstop
- The Hawaiian words html page uses \r and \n for new lines
  - od –c hwnwdseng.htm to see the ASCII characters
- You can delete blank white spaces such as tab or space using
  - tr -d '[:blank:]’
  - Use tr -s to squeeze multiple new lines into one
- sed 's/<[^>]*>//g' a.html to remove all HTML tags

Homework 2

- Write a script `sameln` that does the following:
  - User provides a directory name as an argument
  - Finds all regular files in directory(do not recurse) and ignores all other types (directories, symlinks, etc.)
  - If 2 or more files have the same content (cmp)
    - Keep the file whose name is alphabetically first or starts with a dot
    - Replace duplicates with hard links (ln)
    - File names may contain special characters!
  - Hint: see the cmp, ln, and test utilities.

Checking Hard Links

- Inode: data structure that stores information about files
  - File type
  - Permission
  - Owner
  - File Size, etc.
- Each inode is identified by a unique inode number within the file system
- Check a file's inode number: ls –i filename
- **How do you check if two files are hard-linked?**
  - Same inode number

Homework Hints

- Array itself
  - ${arrayName[@]}
- Iterate over an array in bash
  - for i in ${arrayName[@]}
- Length of array
  - ${#arrayName[@]}
- Length of element
  - ${#arrayName[1]}

Hardware prerequisite (Assignment 7)

BeagleBone Green Wireless

http://web.cs.ucla.edu/classes/winter18/cs35L/syllabus.html