CS 35L

Software Construction Laboratory

Lecture 2.1

8th October, 2019

Logistics

- Assignment 2 Deadline
  - Deadline - Monday, 14th October, 11:55pm
- If you are looking for PTE’s or wanting to switch labs, continue to write your name on the sheet of paper
- Assignment 10
  - Will create a sheet for presentations from Week 3
- Hardware requirement for Week 8
  - Seeed Studio BeagleBone Green Wireless Development Board
  - Buy individual boards

Review - Previous Lab

- Locale Command
  - The C Locale
- Standard Streams
  - 0 – Standard Input Stream
  - 1 – Standard Output Stream
  - 2 – Standard Error Stream
- Redirection and Pipeline
  - > , >>, < , 2> - Redirection Operators
  - | - Pipe Operator
- Sort, Comm and Tr commands

Shell Scripting – What is a shell?

- The shell is a user interface to the OS
- Accepts commands as text, interprets them, uses OS API to carry out what the user wants – open files, start programs...
- Common shells
  - bash, sh, csh, ksh

Compiled Languages v/s Scripting Languages

Compiled Languages

- Examples: C,C++,Java
- First Compiled
- Source code to object code; then executed
- Run faster
- Applications:
  - Typically run inside a parent program like scripts, more compatible during integration, can be compiled and used on any platform (eg. Java)

Scripting Languages

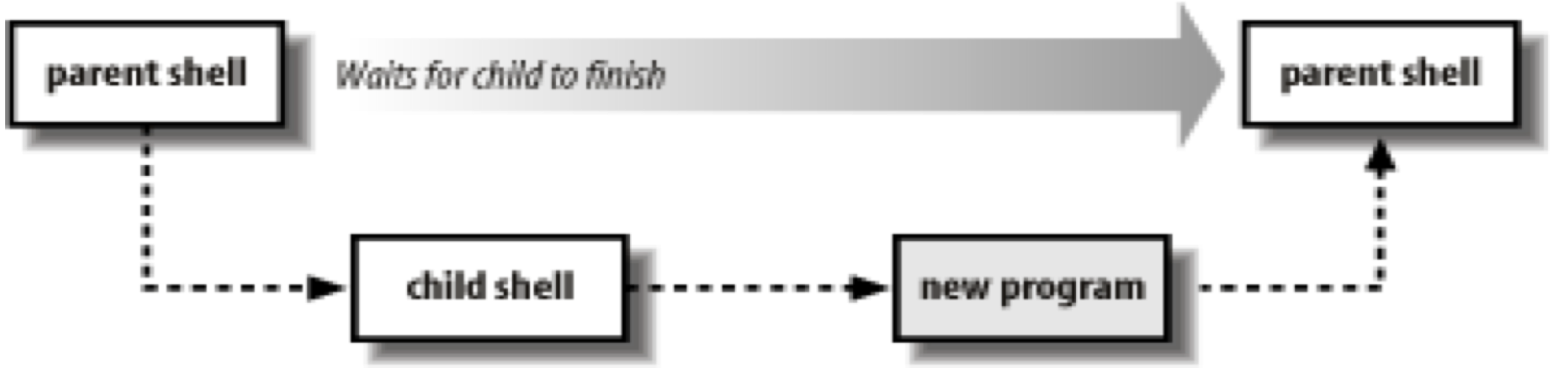
- Examples: Python, JavaScript, Shell Scripting
- No compilation required. Directly interpreted!
- Interpreter reads program, translates into internal form and executes
- Runs slower than a high level language
- Applications:
  - Automation, Extracting information from a data set, Less code intensive

Shell Script

- A computer program designed to be run on a shell (UNIX/Linux)
- All shell commands can be executed inside a script
- Why use a shell script?
  - Simplicity
  - Portability
  - Ease of development

Scripts: First Line

- A shell script file is just a file with shell commands
- When shell script is executed a new child "shell" process is spawned to run it
- The first line is used to state which child "shell" to use
  - #! /bin/sh
  - #! /bin/bash



Sample Shell Script

- Write a Shell script to print Hello World

Simple Execution Tracing

- Shell prints out each command as it is executed
- Execution tracing within a script:
  - set –x: to turn it on
  - set +x: to turn it off

Output using echo or printf

- echo writes arguments to stdout, can’t output escape characters (without –e)

  ```
  $ echo “Hello\nworld”
  Hello\nworld
  $ echo –e “Hello\nworld”
  Hello
  world
  ```

- printf can output data with complex formatting, just like C printf()

  ```
  $ printf "%.3e\n" 46553132.14562253
  4.655e+07
  ```

Variables

- Declared using =
  - var="hello"   ***#NO SPACES!!!***
- Referenced using $
  - echo $var
- Example:

  -
    ```
    #!/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
  - #! /bin/sh
  - #test script
  - echo “first arg is $1”

  - ./test hello
  - first arg is hello

Quotes behaviour - Exercise

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

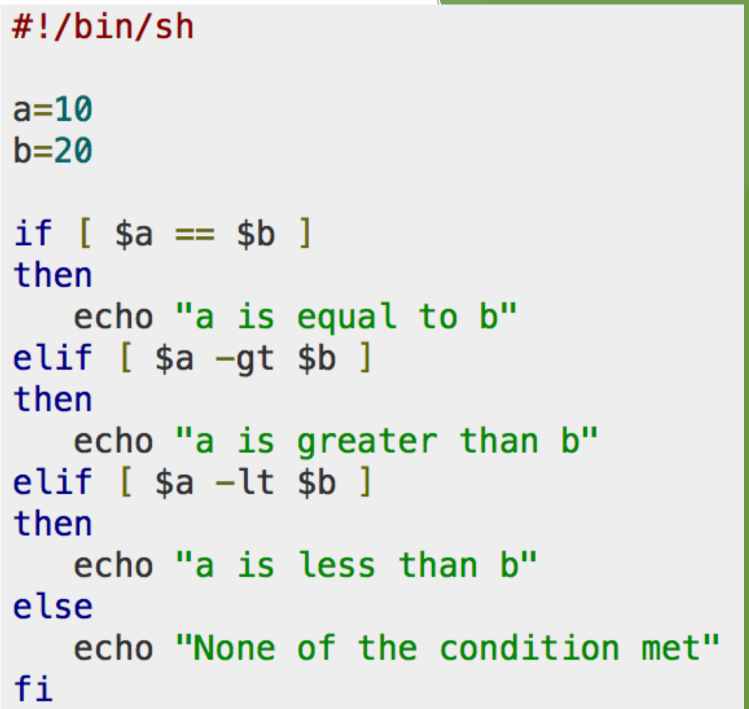
Q) What are the outputs?

Quotes Behaviour

- 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 ` ` or $()
  - Expand as shell commands
  - Try temp=`ls` ; echo $temp

Conditional and Unconditional Statements

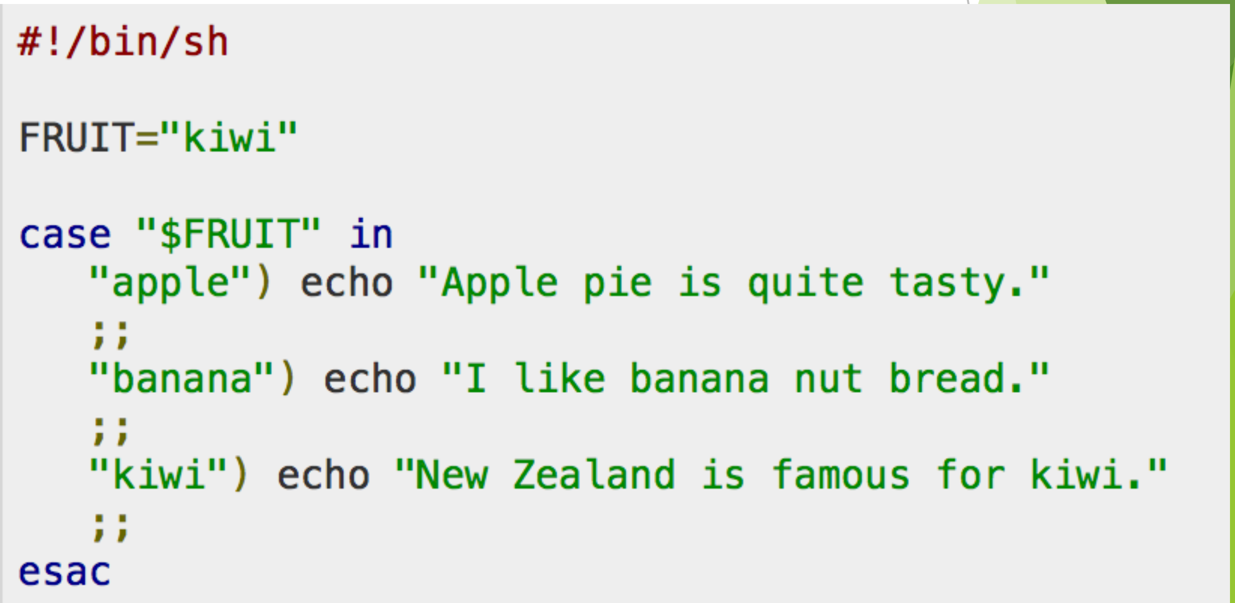
- **Conditional**
  - if…then…fi
  - if…then…else…fi
  - if…then…elif..then…fi
  - case…esac
- **Unconditional**
  - break
  - continue



```sh
#!/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
```



```sh
#!/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 – Example:

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

Note the (( )) to do arithmetic operations

Loops

- For Loop – Example:

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

Note: f will refer to each word in `ls` *output*

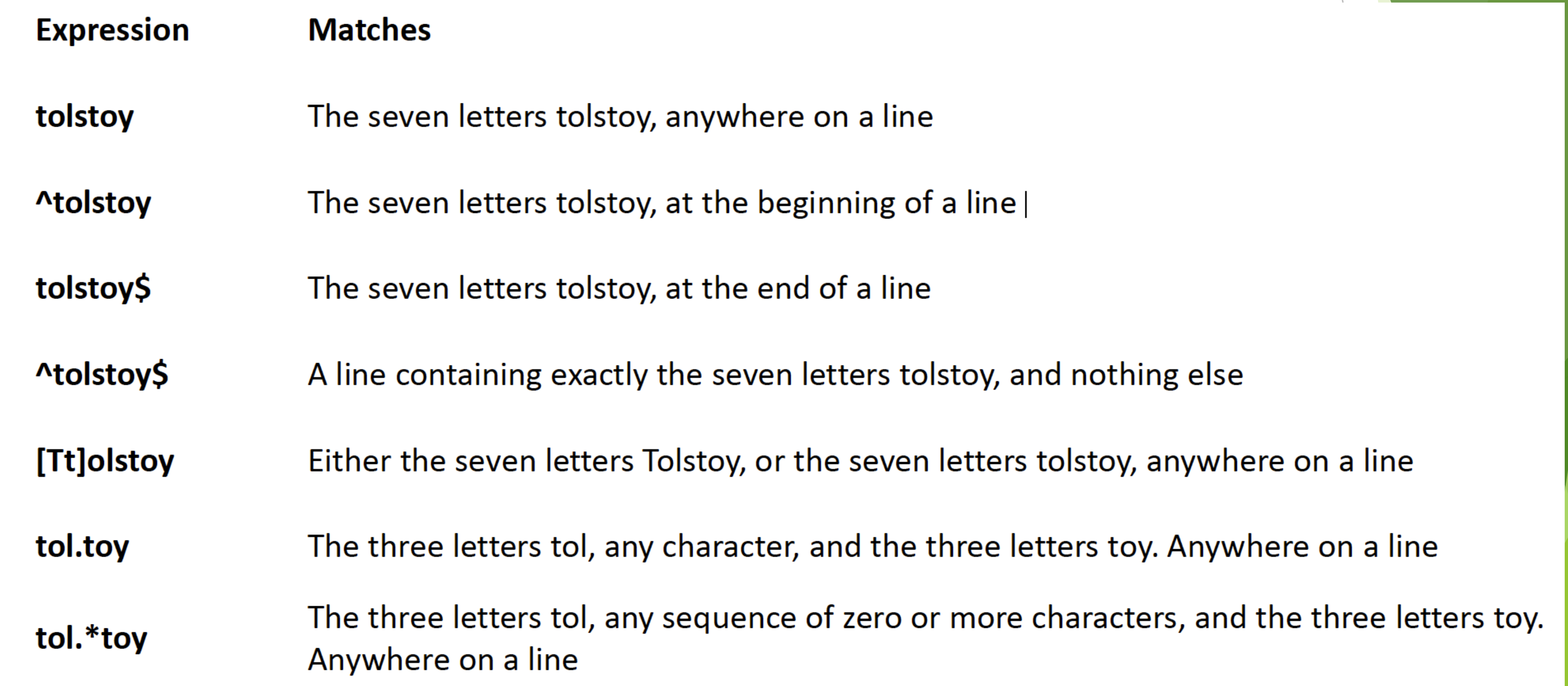
Regular Expressions (regex)

- A regex is a special text string for describing a certain search pattern
- Quantification
  - How many times of previous expression?
  - Most common quantifiers: ?(0 or 1), *(0 or more), +(1 or more)
- Alternation
  - Which choices?
  - Operators: [] and |
  - E.g Hello|World , [A B C]
- Anchors
  - Where?
  - Characters: ^(beginning) and $(end)

regex contd…

- ^ start of line
- $ end of line
- \ turn off special meaning of next character
- [] match any of enclosed characters, use – for range
- [^] match any characters except those enclosed in []
- . match a single character of any value
- * match 0 or more occurrences of preceding character/expression
- + match 1 or more occurrences of preceding character/expression

regex contd…



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

Basic Regular Expressions (BRE) vs Extended Regular Expressions (ERE)

- In basic regular expressions the meta-characters ‘?’, ‘+’, ‘{’, ‘|’, ‘(’, and ‘)’ lose their special meaning; instead use the backslashed versions ‘\?’, ‘\+’, ‘\{’, ‘\|’, ‘\(’, and ‘\)’ for their special meaning.
- In extended regular expressions, the meta characters, ‘?’, ‘+’, ‘{’, ‘|’, ‘(’, and ‘)’ retain their special meaning. They can be literally used by escaping them:‘\?’, ‘\+’, ‘\{’, ‘\|’, ‘\(’, and ‘\)’.
- *man grep* for more information

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 sub patterns can be saved on a single pattern. The text matched by the sub patterns 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.

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.

Regular Expressions (cont’d)

*	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	[:ascii:]	**ASCII Characters**

Regex Exercises

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

Regex Exercises

- Which regular expression would match the words “favorite” and “favourite”?

Regex Exercises

- Which regular expression would match the words “Ggle”, “Gogle” and “Google”?

- Which one would match “Gogle”, “Google” and “Gooogle” but not “Ggle”?

Regex Exercises

- Which regular expression would match any version of the word “Google” that has an even number of o’s?

- Which regular expression would match any version of the word “Google” that has fewer than 7 O’s?

Regex Exercises

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

Regex Exercises

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

Regex Exercises

- Which regular expression would match all subdirectories within a directory?

Assignment 2 - Laboratory

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

Assignment 2 - Laboratory

- Build a spelling checker for the Hawaiian language
  - Get familiar with sort, comm and tr commands!
- Steps:
  - Download a copy of web page containing basic English-to-Hawaiian dictionary
  - Extract only the Hawaiian words from the web page to build a simple Hawaiian dictionary. Save it to a file called hwords (site scraping)
  - Automate site scraping: buildwords script (cat hwnwdseng.htm | buildwords > hwords)
  - Modify the command in the lab assignment to act as a spelling checker for Hawaiian
  - 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

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

Buildwords

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

Questions?