CS35L – Winter 2019

Slide set:	2.1
Slide topics:	Shell scripting, regex, streams
Assignment:	2

HARDWARE PREREQUISITE (ASSIGNMENT 7)

SEEED STUDIO BEAGLEBONE GREEN WIRELESS DEVELOPMENT BOARD
HTTPS://WEB.CS.UCLA.EDU/CLASSES/WINTER19/CS35L/SYLLABUS.HTML

Unix Environment

- Defined by **Environment Variables**
- **Environment Variable -** named object that contains data used by one or more applications
- Simply put - it is a variable with a name and a value
- Examples:
  HOME: path to user's home directory
  PATH: list of directories to search in for command to execute
- Change value:
  export VARIABLE=…

Locale

A locale
- Set of parameters that define a user's cultural preferences
  - Language
  - Country
  - Other area-specific things

locale command prints information about the current locale environment to standard output

LC_* Environment Variables

locale gets its data from the LC_* environment variables
Examples:
- LC_TIME

    Date and time formats

- LC_NUMERIC

    Non-monetary numeric formats

- LC_COLLATE

    Order for comparing and sorting

The 'C' Locale

- The default locale

- An environment of "least surprise"

- Behaves like Unix systems before locales

Locale Settings Can Affect Program Behavior!!

Default sort order for the sort command depends:

- LC_COLLATE='C': sorting is in ASCII order
- LC_COLLATE='en_US': sorting is case insensitive except when the two strings are otherwise equal and one has an uppercase letter earlier than the other.

Other locales have other sort orders!

Shell Scripting

sort, comm, and tr

sort: sorts **lines** of **text** files
- Usage: sort [OPTION]…[FILE]…
- Sort order depends on locale
- C locale: ASCII sorting

comm: compare two **sorted** files **line by line**
- Usage: comm [OPTION]…FILE1 FILE2
- Comparison depends on locale

tr: translate **or** delete characters
- Usage: tr [OPTION]…SET1 [SET2]
- Ex: echo **"12345" | tr "12" "ab"**
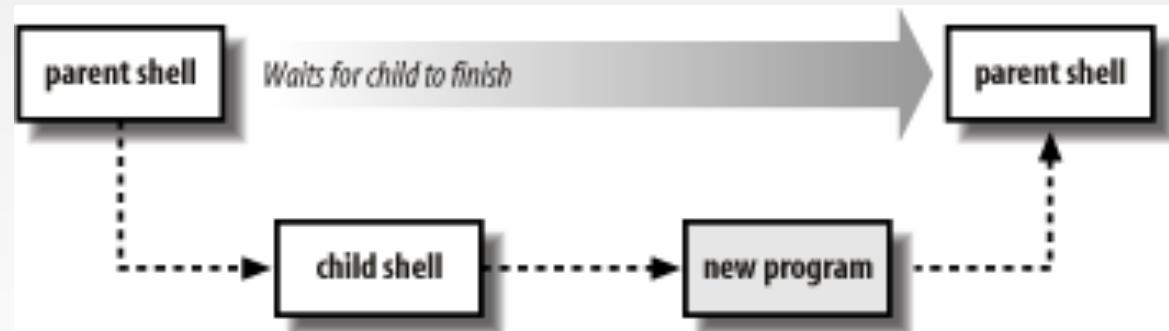
The Shell and OS

- 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

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

Execute shell scripts

$ touch script.sh

$ ./script.sh

-bash: ./script.sh: Permission denied

$ ls –al

-rw-r--r--   1 user1 csgrad   0 Apr 6 11:19 script.sh

$ chmod +x script.sh

$ ./script.sh

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

```
#!/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-Exercise

a=pwd
echo '$a'
echo "$a"
echo `$a`

Q) What are the outputs?

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 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**

  ```
  #!/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
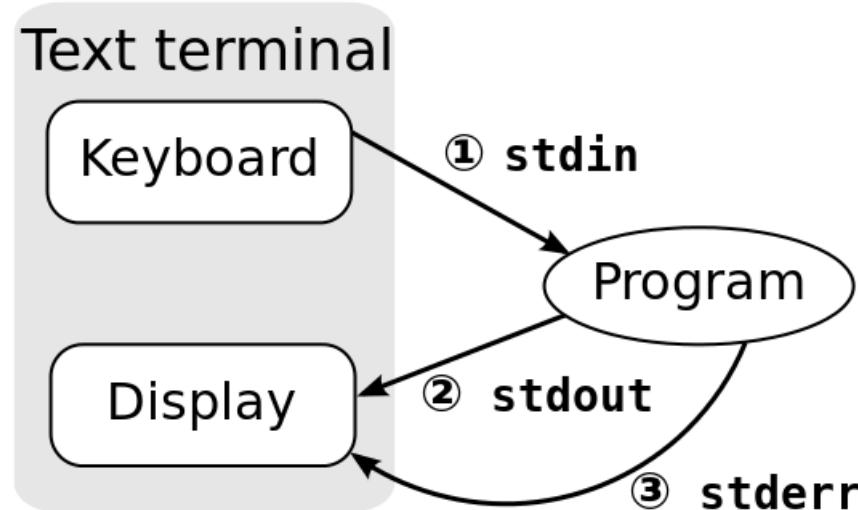
- **For loop**

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

  - f will refer to each word in ls output

Standard Streams

- Every program has these 3 streams to interact with the world

  - stdin (0): contains data going into a program

  - stdout (1): where a program writes its output data

  - stderr (2): where a program writes its error msgs

Redirection and Pipelines

- *program < file* redirects *file to program's stdin*:
  cat <file

- *program > file* redirects *program*'s stdout to *file2*:
  cat <file >file2

- *program 2> file* redirects *program*'s stderr to *file2*:
  cat <file 2>file2

- *program >> file* **appends** program's stdout to *file*

- *program1 | program2* assigns stdout of *program1* as the stdin of *program2; text 'flows' through the pipeline*
  cat <file | sort >file2

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 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.

Examples

Expression	Matches
tolstoy	
^tolstoy	
tolstoy$	
^tolstoy$	
[Tt]olstoy	
tol.toy	
tol.*toy	

1. The seven letters tolstoy, anywhere on a line

1. The seven letters tolstoy, at the beginning of a line

1. The seven letters tolstoy, at the end of a line

1. A line containing exactly the seven letters tolstoy, and nothing else

1. Either the seven letters Tolstoy, or the seven letters tolstoy, anywhere on a line

1. The three letters tol, any character, and the three letters toy, anywhere on a line

1. 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