CS 35L

Software Construction Laboratory

Lecture 2.1

15th January, 2019

Logistics

- 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

- Types of Processes
- Diff command
- Emacs
  - Standard Editor Operations
  - Emacs Tricks
- UNIX Wildcards
  - * - matches zero or more characters
  - ? – matches exactly one character
  - [] – matches characters enclosed by them

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

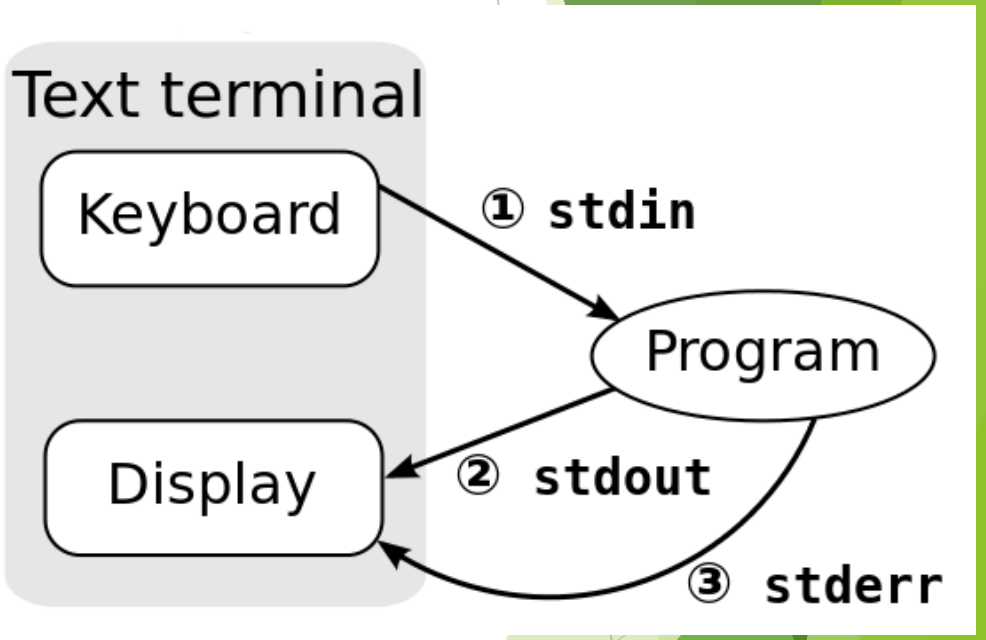
- The default locale
- An environment of “least surprise”
- Behaves like Unix systems before locales

Locale Settings

- Locale Settings affect default command execution
- Default sort order for the sort command depends on:
  - 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!

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 & Pipelines

- program < file redirects file to programs’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

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"

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

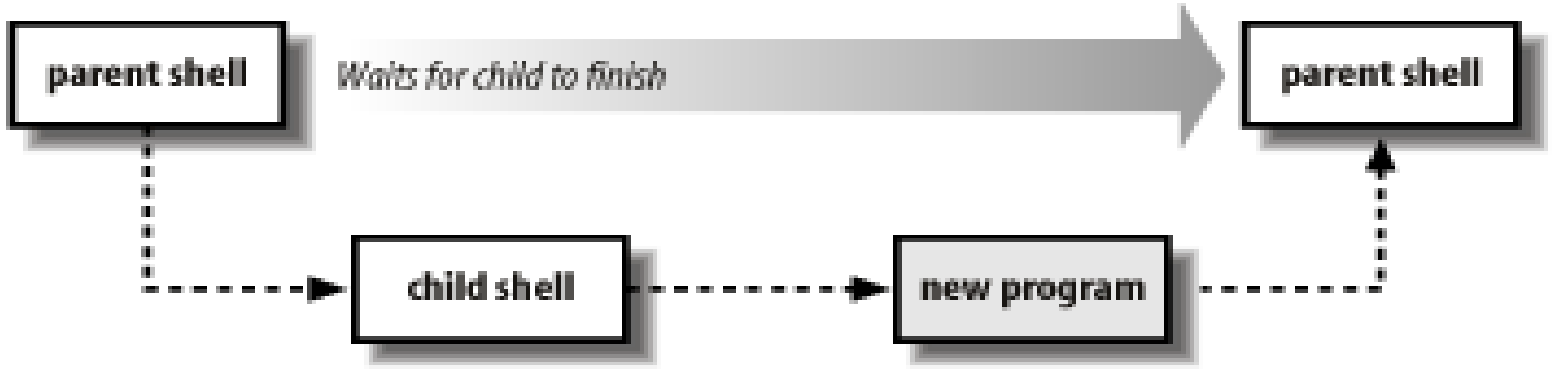
- Examples: Python, JavaScript, Shell Scripting
- No compilation required. Directly interpreted!
- Interpreter reads program, translates into internal form and executes
- Runs slower than a high level language
- Applications:
  - Automation, Extracting information from a data set, Less code intensive

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

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 and $
- 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

```
#!/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
```

```
#!/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

Questions?