CS 35L

LAB 8, Session 2

TA: Sucharitha Prabhakar

EMAIL ID: prabhakarsucharitha@gmail.com

Outline

- Pipelines and redirection
- Shell script
- Interpreted languages

Details

mkdir lab2_1 (Make a directory for each lab session)

cd lab2_1

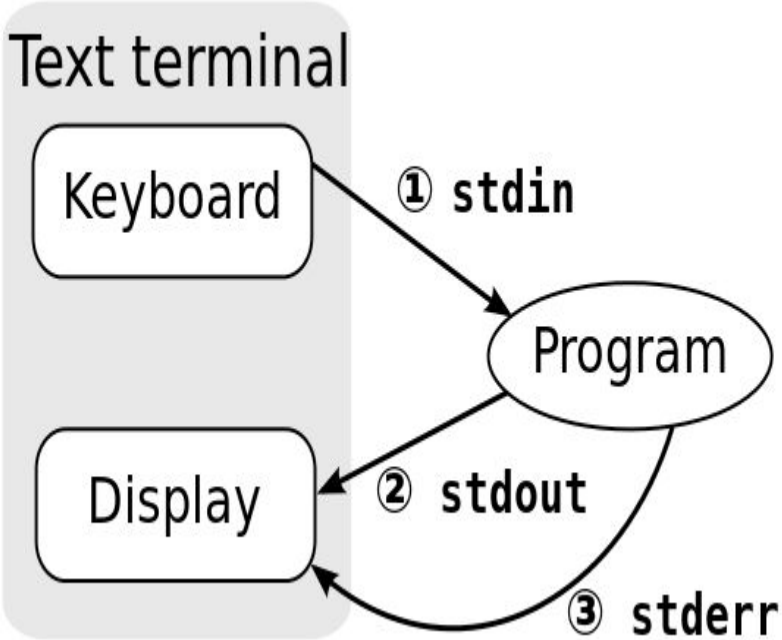
touch lab.log (optional)

touch lab.txt

touch hw.txt

Redirection and Pipeline

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 - Bourne shell

- > Redirect standard output
- 2> Redirect standard error
- 2>&1 Redirect standard error to standard output
- < Redirect standard input
- | Pipe standard output to another command
- >> Append to standard output
- 2>&1| Pipe standard output and standard error to another command

Redirect standard output

```
ls -l

ls -l > file1.txt

cat file1.txt

rm file1.txt
```

Redirect standard input

echo "Hello"

echo "Hello" > myfile.txt (output redirection)

cat < myfile.txt (input redirection)

echo “Appending another Hello” >> myfile.txt

cat < myfile.txt

Pipeline

find . -type f (Find all files in current directory and subdirectories)

wc -l (prints new line counts)

find . -type f | wc -l (show total number of files in current directory and subdirectories)

Shell Script

Shell?

The shell is a user interface to access OS’s services like file management, process management etc

Accepts commands as text, interprets them, uses kernel API to carry out what the user wants – open files, start programs etc and displays output

Types of UNIX shells

Two major types of shells:

1. The Bourne shell - the default prompt is the $ character. Subcategories - Bourne shell (sh), Korn shell (ksh), Bourne Again shell (bash), POSIX shell (sh)
2. The C shell - the default prompt is the % character. Subcategories - C shell (csh), TENEX/TOPS C shell (tcsh)

Compiled vs. Interpreted

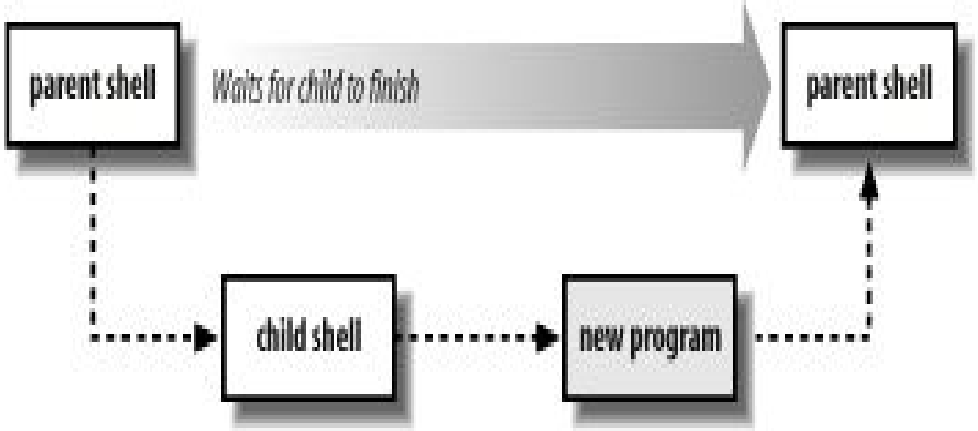
Compiled

- Programs are translated from their original source code into machine code that is executed by hardware
- Efficient and fast
- Require recompiling
- Work at low level, dealing with bytes, integers, floating points, etc.
- Ex: C/C++

Interpreted

- Interpreter program (the shell) reads commands, carries out actions commanded as it goes
- Much slower execution
- Portable
- High-level, easier to learn
- Ex: PHP, Ruby, bash

Shell script executions



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

Simple shell script

1. emacs hello.sh

In the file,

2. #!/bin/bash

echo “Hello”

3. Save and exit (C-x C-s, C-x C-c)

Executing a shell script

```
bash ./hello.sh

ls -l hello.sh

chmod +x hello.sh

ls -l hello.sh

bash ./hello.sh
```

Output using echo and printf

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

Example: echo “Hello\nworld” vs echo –e “Hello\nworld”

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

Example: printf "%.3e\n" 46553132.14562253

Quotes

Three kinds of quotes

- Single quotes ' '
  - Do not expand at all, literal meaning
- Double quotes " "
  - Almost like single quotes but expand $, ‘ ‘ and ‘\’
- Backticks ` `
  - Expand as shell commands

Refer example program

Declaring variables

Declared using =

var=“hello” #NO SPACES!!!

Referenced using $

echo $var

Decision statements

If statements

If..else statements

case..esac statements

Refer sample programs

If statement

```
if [ expression ]
then
   Statement(s) to be executed if expression is true
fi
```

```
if [ expression ]
then
   Statement(s) to be executed if expression is true
else
   Statement(s) to be executed if expression is not true
fi
```

Case statement

```
case word in
  pattern1)
    Statement(s) to be executed if pattern1 matches
    ;;
  pattern2)
    Statement(s) to be executed if pattern2 matches
    ;;
  pattern3)
    Statement(s) to be executed if pattern3 matches
    ;;
esac
```

For loop

```
for var in word1 word2 ... wordN
do
   Statement(s) to be executed for every word.
done
```

While loop

```
while command
do
   Statement(s) to be executed if command is true
done
```

Select loop

```
select var in word1 word2 ... wordN
do
   Statement(s) to be executed for every word.
done
```

Special Variables

$0 - The filename of the current script.

$n - These variables correspond to the arguments with which a script was invoked. Here n is a positive decimal number corresponding to the position of an argument (the first argument is $1, the second argument is $2, and so on).

$# - The number of arguments supplied to a script.

$* - If a script receives two arguments, $* is equivalent to $1 $2.

$? - The exit status of the last command executed.

$$ - The process number of the current shell. For shell scripts, this is the process ID under which they are executing.

Exit values

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