Week 2: Shell Scripting

bash,linux,shell

The Shell and OS

- the shell is a UI 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 vs. Interpreted

Compile languages

- programs translated from original source code into machine code then executed by hardware
- efficient and fast
- require recompiling
- work at low level, dealing with bytes, integers, floating points, etc.
- Ex: C/C++

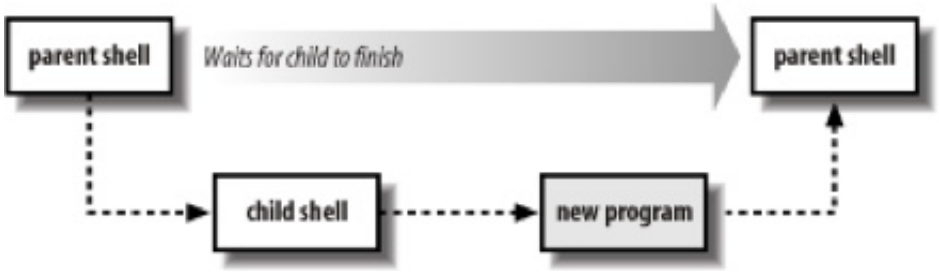
Interpreted languages

- 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

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
```



Example

- a lab directory for each lab
- before each lab:
  - remove old directory called lab
  - create new directory called “lab”
  - create 3 files in “lab
    - lab.log
    - lab.txt
    - hw.txt

```
rm -rf lab              # remove old lab folder (recursively, force)

mkdir lab               # create new lab folder

touch lab/lab.log       # create a lab.log in the lab folder
touch lab/lab.txt       # create a lab.txt in the lab folder
touch lab/hw.txt        # create a hw.txt in the lab folder
```

Executing a Shell Script

```
$ touch script.sh                       # script's name is script.sh
$ ./script.sh                           # (attempt to) execute script
-bash: ./script.sh: Permission denied   # no executable permission set by touch
$ ls -al                                # list files to check permissions
-rw-r--r--  1 bibek csundergrad   0 Apr 4 11:19 script.sh
$ chmod +x script.sh                    # modify permission
$ ./script.sh                           # execute script
```

Simple Execution Tracing

- shell prints out each ommand 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" 46553132.323
4.655e+07
```

Variables

- declared using =
  - var="hello"
  - NO SPACES
- referenced using $
  - echo $var
- not type-safe

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

POSIX Built-in Shell Variables

- important ones for next assignment are highlighted

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.

```
IFS=','
# "Hello,world" will be "tokenized" by the comma separator
```

Exit: Return value

- check exist 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, encolse the number in braces if it's greater than 9

```
1 #! /bih/sh
2 #test script
3 echo first arg is $1
4 #echo tenth arg is $(10)
5
6 ./test hello
7 first arg is hello
```

if Statements

- if statements use the test command or []
- `man test` to see the expressions that can be done

```
1 #!/bin/bash
2 if[ 5 -gt 1 ]                  # if 5 is greater than 1
3 then
4     echo "5 greater than 1"     # indentation REQUIRED
5 else
6     echo "not possible"
7 fi
```

Quotes

- three kinds of quotes

Single Quotes ‘ ‘

- do not expand at all, literal meaning
    - `temp=‘$hello$hello’ ; echo $temp`
        - $hello$hello

Double Quotes “ “

- almost like single quotes but expand backticks and $

Backticks ` ` or $()

- expand as shell commands
    - `temp=`ls` ; echo $temp`

Loops

while loops

```
#!/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 loops

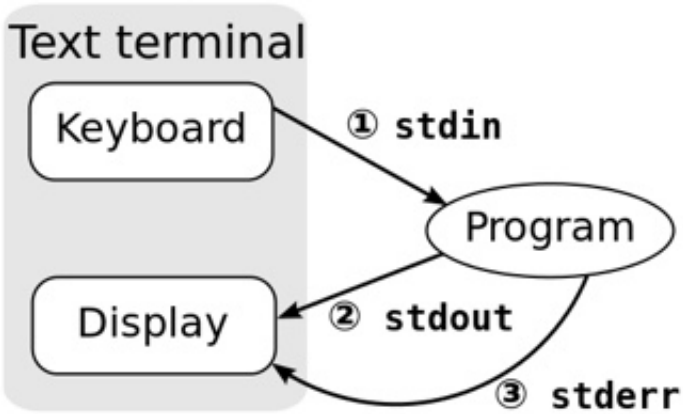
```
#!/bin/sh
temp=`ls`
for f in $temp
do
    echo $f
done

# 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
- `program > file`
  - redirects program's stdout to file
- `program 2> file`
  - redirects program's stderr to file
- `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`