Shell scripting and system variables

HORT 530

Lecture 5

Instructor: Kranthi Varala

Command-line operations

- All commands so far are run one at a time.
- Redirection and pipes allow combining a few commands together into a single pipeline.
- Lacks logical complexity, such as ability to make decisions based on input / values in file.
- Certain repetitive tasks are tedious to user.
- All commands are being sent to and interpreted by the ‘shell’.

Shell scripts

- Shell scripts are at the simplest level a series of commands.
- Not meant for computational or memory intensive tasks. Commonly used as "glue code".
- Calls on individual programs, such as grep, sed, sort etc. to do the heavy lifting.
- Two reasons to write a script:
  - Ease-of-use e.g., automate repetitive tasks
  - Reproducibility

Terminology

- Terminal: Device or Program used to establish a connection to the UNIX server
- Shell: Program that runs on the server and interprets the commands from the terminal.
- Command line: The text-interface you use to interact with the shell.

Shells

- Shell itself is a program on the server and can be one of many varieties
  1. bash : Most popular shell, default on most Linux systems. Installed on all Linux systems
  2. zsh : A bash-like shell with some extra features. E.g., support for decimals, spelling correction etc.
  3. tcsh : A C-like syntax for scripting, supports arguments for aliases etc.
- We will work with bash shell scripting since it is the most common and supported shell.

Environment variables

- A variable is a container that has a defined value.
- It's called a variable because the value contained inside it can change.
- Variables allow changing a part of the command that is to be executed.
- Every shell has a set of variables, called environment variables, attached to it. You can list them by using the command `env`
- E.g., the variable `SHELL` contains the path to the current shell.

Working with environment variables

- Set the value of a variable as follows:

```
$ FOO=BAR
```

- Retrieve the value of a variable as follows:

```
$ echo $FOO
```

Example Environment variables

- On scholar as of today: using the command `env` in my bash shell shows 142 environment variables.
- Examples:

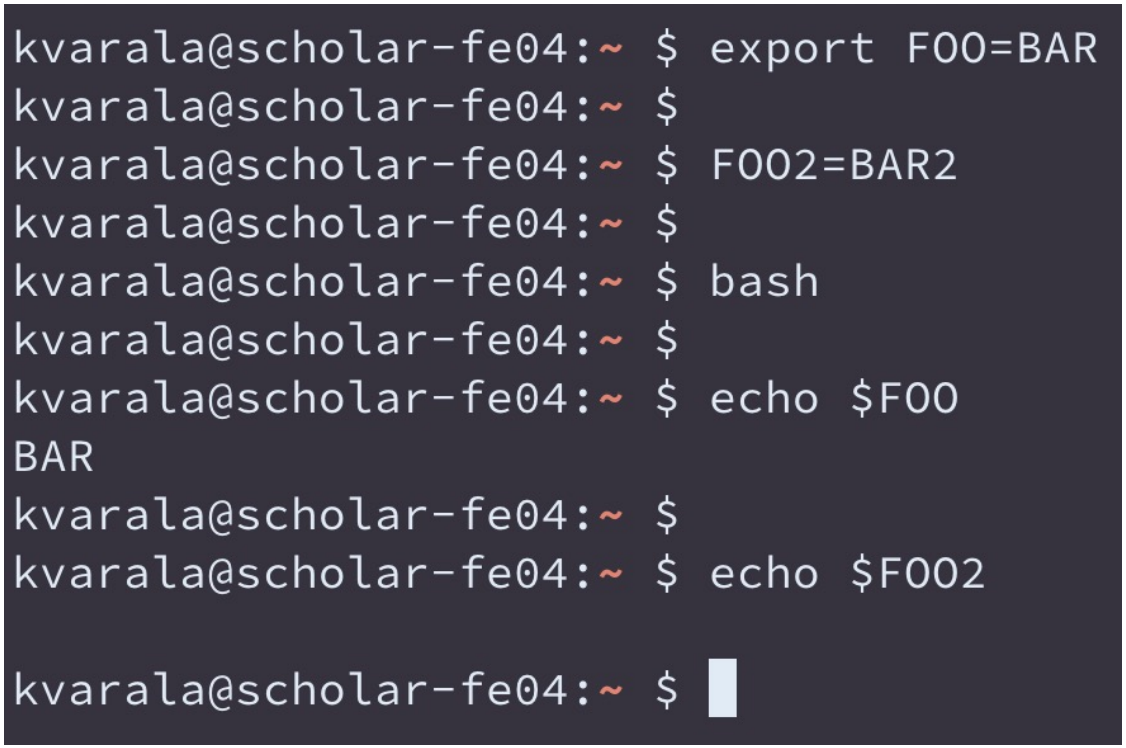
```
HOME=/home/kvarala
SHELL=/bin/bash
HOSTNAME=scholar-fe00.rcac.purdue.edu
HISTSIZE=1000
RCAC_SCRATCH=/scratch/scholar/kvarala
```

Environment vs. Shell variables

- Environment variables are ‘global’ i.e., shared by all shells started AFTER the variable is defined.
- Shell variables are only present in the shell in which they were defined.
- Environment variables are inherited by child shells but shell variables are not.
- Shell variable can be made an environment variable by using `export` command.

```
FOO=BAR
export FOO
```

Environment vs. Shell variables



FOO defined in the environment

FOO2 defined in shell

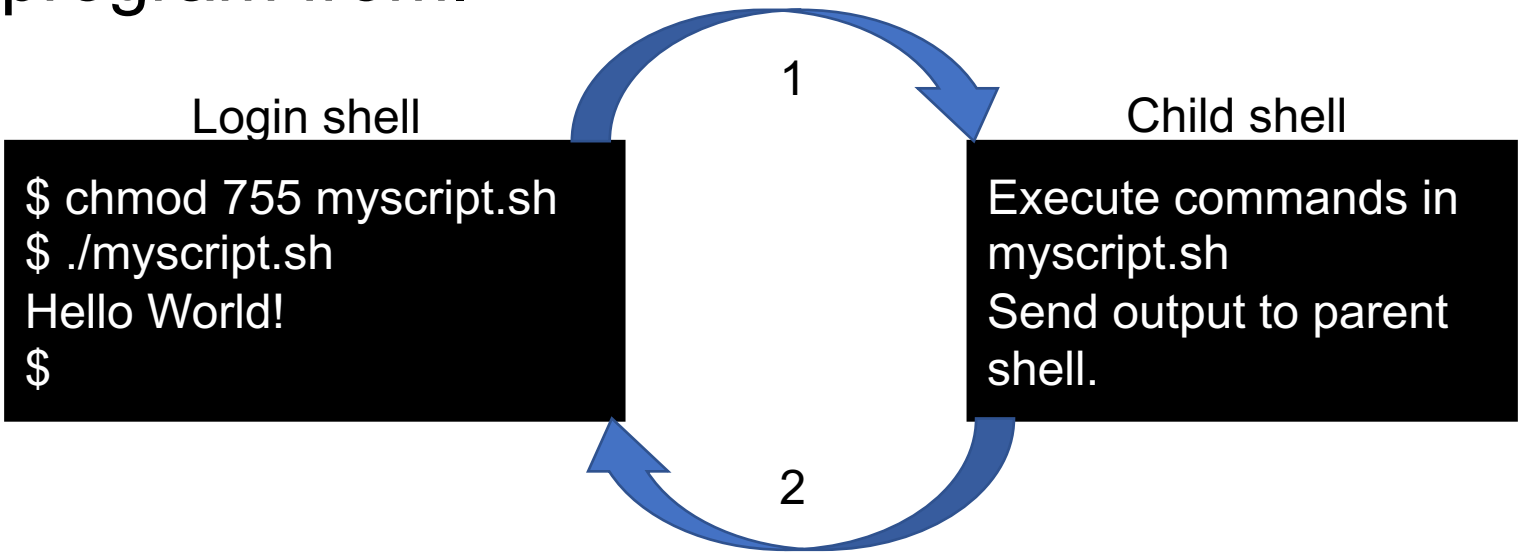
Start a child shell

echoes value of FOO

empty

Script creates its own Shell

- A script is always executed in its own shell, i.e., when you execute a shell script it starts a new child shell within the shell you executed the program from.



Shell Scripting

- Commands within a shell script may or may not be dependent on each other.
- Variables, hence their values, can be transferred from one command to another.
- Shell scripts support:
  - Variables
  - Conditions
  - Loops
  - Functions

Example Shell Script

- First example script: Hello world!

```
#!/bin/bash
# This is our first shell script!!
echo "Hello World!"
```

Variables in Shell Scripting

- Variables are containers that store a value.
- All variables created in a script are shell variables.
- A script can access the environment variables in addition to its own shell variables.
- Variable can store any kind of value i.e., string or integer or floating point number etc.

Variables in Shell Scripting

```
INT=1
FLOAT=1.5
STR=hello
STR2="hello world"
RND=asdf2341.sfe

echo $INT
echo "Value of FLOAT is $FLOAT"
echo "$STR is a string"
echo "$RND is non-sensical"
```

Example Shell Script

- Second example script: `lsScr.sh`

```
#!/bin/bash
# List contents of scratch
cd $RCAC_SCRATCH
ls -l
```

- Make script executable, place it in PATH.

Special shell variables

- Special Variables
  - $# = No. of parameters given to script
  - $@ = List of parameters given to script
  - $0 = Name of current program (script)
  - $1, $2.. = Parameter 1, 2 and so on..
  - $? = Exit value of last command run
- These variables are shell variables and only valid to the current shell.

Even more special characters

- * matches every character, just as in regular expressions.
- So, ls *txt in a script will list all files whose name ends in txt.
- \ is an escape character which tells the shell to not interpret the character after it.
- \ is commonly used to escape the special characters such as *, $ etc.

Example Shell Script

- Third example script: `lsScr.2.sh`

```
#!/bin/bash
# List contents of scratch
echo "Executing script : \"$0\" with
$# parameters"
cd $RCAC_SCRATCH
ls -l
```

- Make script executable, place it in PATH.

Command Blocks

- A block is a set of commands that are grouped together for execution.
- Two fundamental blocks in scripting:
  - Loops
    Repeat the commands in the block until the exit condition is met.
  - Conditions
    Evaluate condition and if true execute commands in the block.

Loops

- Two kinds of loops supported in bash:
  - for loop
    operates on a list and repeats commands in the block for each element on the list
  - while loop
    repeats commands in the block until an exit condition is met.

for loops

- for loop
  operates on a list and repeats commands in the block for each element on the list

```
for x in [ list ];
do
        commands
done
```

for loops

- for loop
  operates on a list and repeats commands in the block for each element on the list

```
for x in 1 2 3 4 5 6 7 8 9 10;
do
      echo "Value of x is : $x"
done
```

for loops

- for loop
  operates on a list and repeats commands in the block for each element on the list

```
for x in $( ls );
do
      echo "Found file $x"
done
```

while loops

- while loop
  repeats commands until exit condition is met

```
while condition
do
      echo "Value of x is : $x"
done
```

while loops

- while loop
  repeats commands until exit condition is met

```
x=10
while [ $x -gt 0 ]
do
        echo "Value of x is : $x"
        let x=x-1
done
```

Initiate variable

Check value of variable

Change value of variable

Condition blocks

- Condition blocks test for a condition and if TRUE execute one block and if FALSE execute another.

```
if [ condition ]
then
     Block 1
else
     Block 2
fi
```

Condition blocks

- Condition blocks test for a condition and if TRUE execute one block and if FALSE execute another.

```
if [ $1 –gt 0 ]
then
     echo "$1 is greater than 0"
else
     echo "$1 is smaller than 1"
fi
```

breaking loops

- Break command asks the shell to exit the loop

```
x=10
while [ 1 ]
do
	echo "Value of x is : $x"
	x=x-1
	if [ $x == 0 ]
	then
		break
	fi
done
```

Run external commands

- backticks are a way to send a command to the shell and capture the result :

```
files=`ls *txt`
echo $files
```

- Another way is to use `$( )` :

```
files=$( ls *txt )
echo $files
```

Functions in shell scripting

- Functions separate logical blocks of code.
- Typically a function contains a piece of code that is used repeatedly in a script.
- Code in a function is only executed when a function is ”called”.
- Functions can “receive” arguments and ”return” values.
- Functions allow code portability.

Functions in shell scripting

```
#!/bin/bash

fileSize(){
	wc -l $1 | awk '{print $1}'
}

for x in $( ls );
do
	fileSize $x #Function output sent to STDOUT
	size=$( fileSize $x ) #Capture function output
	echo -e "Found file $x\t with $size lines."
done
```

Summary

- Shell scripts allow easy automation and reproducibility.
- Shell scripts support basic programming tenets such as Variables, Conditions, Loops and Functions.
- Shell scripting is a useful way to combine the various simple but powerful command-line tools.
- Functions can be copied from one script to another to allow reuse of code.