UTSA

The University of Texas at San Antonio

University Technology Solutions

Express Bash Scripting Tutorial Part 3

Quickly Learn Bash Scripting in Linux

Brent League

Overview

- Environment Variables
  - Reading Environment Variables
  - Standard Environment Variables
- Functions
  - The Basics of Functions
  - Using Parameters
  - Piping

Reading Environment Variables

Create a new script called `vars.sh` and make it executable by typing `chmod +x vars.sh`

Your new script should include the following lines:

```
#!/usr/bin/env bash

echo "The PATH is: $PATH"
echo "The terminal is: $TERM"
echo "The editor is: $EDITOR"
```

Reading Environment Variables

We can detect if a value is not set by using an IF statement

Add the lines in red to your `vars.sh` script

```
#!/usr/bin/env bash

echo "The PATH is: $PATH"
echo "The terminal is: $TERM"
echo "The editor is: $EDITOR"

if [[ -z $EDITOR ]]
then
 echo "The EDITOR variable is not set"
fi
```

Changing Environment Variables

We can change the value of any environment variables in our script. Add the lines in red to your `vars.sh` script

```
#!/usr/bin/env bash

echo "The PATH is: $PATH"
echo "The terminal is: $TERM"
echo "The editor is: $EDITOR"

if [[ -z $EDITOR ]]
then
 echo "The EDITOR variable is not set"
fi

PATH="/UTSA"
echo "The PATH is $PATH"
```

Changing Environment Variables

Let's execute our script and view the output. Type `./vars.sh` and press the enter key

```
The PATH is: /home-
new/ytf623/bin:/cm/shared/apps/sge/2011.11p1/bin/linux-
x64:/cm/local/apps/gcc/8.2.0/bin:/cm/shared/apps/slurm/18.08.9/sb
in:/cm/shared/apps/slurm/18.08.9/bin:/cm/local/apps/environment-
modules/4.2.1//bin:/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/s
bin:/sbin:/usr/sbin:/cm/local/apps/environment-
modules/4.2.1/bin:/opt/dell/srvadmin/bin:/home-
new/ytf623/.local/bin:/home-new/ytf623/bin
The terminal is: xterm
The editor is:
The EDITOR variable is not set
The PATH is /UTSA
```

Note: Your path output is going to differ slightly from what you see here

Standard Environment Variables

- HOME – user's home directory
- PATH – directories which are searched for commands
- HOSTNAME – hostname of the machine or system
- SHELL – shell that is being used
- USER – user of this session
- TERM - type of command-line terminal being used

The Basics of Functions

- Functions let us avoid duplicating code in our scripts
- We can create a function to hold a single copy of code that can be called from multiple places
- There are two ways to create functions – the first is to begin the statement with the word “function” and then enter the name of the function followed by a set of parenthesis. Or you can skip the word “function” altogether and enter the NAME of the function, followed by a set of parenthesis

The Basics of Functions

Create a new script called `func.sh` and make it executable by typing `chmod +x func.sh`

Add the following lines of code:

```
#!/usr/bin/env bash

function GoUTSA() {
  echo "Go UTSA"
}

GoRoadrunners() {
  echo "Go Roadrunners"
}

GoUTSA

GoRoadrunners
```

Using Parameters with Functions

We can pass parameters to our functions just like we can to our scripts by using the parameter symbols. Now add the code in red to the `func.sh` script

```
#!/usr/bin/env bash

function GoUTSA() {
  local TNAME= $1
  echo "Go UTSA $1"
}

GoRoadrunners() {
  echo "Go Roadrunners $1"
}

GoUTSA Football
GoRoadrunners Baseball
GoRoadrunners Basketball
GoUTSA Volleyball
```

Using Parameters with Functions

Execute the script by typing `./func.sh`

Your output should resemble the following:

```
Go UTSA Football
Go Roadrunners Baseball
Go Roadrunners Basketball
Go UTSA Volleyball
```

Piping

- Pipes let us take the output of one program and feed it to the input of another.
- Let's create a script and call it `pipe.sh`. What we want to do is show the first three files in our current directory in descending alphabetical order – each file should also have a count.

Piping

Let's create a new script called `pipe.sh` and include the following commands:

```
#!/usr/bin/env bash

FILES=`ls -1 | sort -r | head -3`
COUNT=1

for FILE in $FILES
do
  echo "File #$COUNT = $FILE"
  ((COUNT++))
done
```

Piping

The code that follows the `FILES=` statement tells our script to do the following:

`ls -1` displays our directory contents in a single column

`sort -r` changes the sort order from alphabetical to reverse alphabetical order

`head -3` instructs our script to display the first 3 results

Execute the script by typing `./pipe.sh` and you should see the following output:

```
File #1 = while.sh
File #2 = vars.sh
File #3 = user.sh
```

Note: the file names in your output may be different, depending on what you have in your directory

UTSA

The University of Texas at San Antonio

University Technology Solutions

Express Bash Scripting Tutorial Part 3

Quickly Learn Bash Scripting in Linux

Brent League