Bash Scripting

Operating System Practice

João Vicente Ferreira Lima

Universidade Federal de Santa Maria
jvlima@inf.ufsm.br
http://www.inf.ufsm.br/~jvlima

2021/2

Outline

Outline

Arithmetic operations

```
#!/bin/bash
# Counting to 11 in 10 different ways.

n=1; echo -n "$n "

let "n = $n + 1"   # let "n = n + 1"  also works.
echo -n "$n "
```

Arithmetic operations

```
#!/bin/bash
# Counting to 11 in 10 different ways.

n=1; echo -n "$n "

let "n = $n + 1"   # let "n = n + 1"  also works.
echo -n "$n "
```

No floating point

Bash does not understand floating point arithmetic. It treats numbers containing a decimal point as strings.

Arithmetic operations

```
: $((n = $n + 1))
#  ":" necessary because otherwise Bash attempts
#+ to interpret "$((n = $n + 1))" as a command.
echo -n "$n "

(( n = n + 1 ))
#  A simpler alternative to the method above.
echo -n "$n "

n=$(($n + 1))
echo -n "$n "
```

Arithmetic operations

```
: $[ n = $n + 1 ]
#  ":" necessary because otherwise Bash attempts
#+ to interpret "$[ n = $n + 1 ]" as a command.
#  Works even if "n" was initialized as a string.
echo -n "$n "

n=$[ $n + 1 ]
#  Works even if "n" was initialized as a string.
#* Avoid this type of construct, since it is obsolete and nonportable.
echo -n "$n "
```

Arithmetic operations

```
# Now for C-style increment operators.
let "n++"           # let "++n"  also works.
echo -n "$n "

(( n++ ))           # (( ++n ))  also works.
echo -n "$n "

: $(( n++ ))        # : $(( ++n )) also works.
echo -n "$n "

: $[ n++ ]          # : $[ ++n ] also works
echo -n "$n "

exit 0
```

Outline

Internal variables

Builtin variables:

- `$BASH` path to the Bash itself
- `$BASH_VERSION` version of Bash
- `$EDITOR` the default editor
- `$HOME` home directory
- `$PATH` path to binaries
- `$PWD` working directory
- `$UID` user ID number

Positional variables:

- `$?` exit status of a command
- `$$` process ID (PID)

Random

`RANDOM` is a internal Bash function that returns a *pseudorandom* integer in 0 - 32767.

```
RANDOM=$$       #  Seeds the random number generator from PID
                #+ of script.

for i in $(seq 1 10)
do
    echo $RANDOM
done
```

Outline

Manipulating strings

String length - `${#string}`

```
stringZ=abcABC123ABCabc

echo ${#stringZ}      # 15
```

Substring extraction - `${string:position:length}`

```
stringZ=abcABC123ABCabc
echo ${stringZ:7}             # 23ABCabc
echo ${stringZ:7:3}           # 23A
                              # Three characters of substring.
```

Manipulating strings

Random password

```
if [ -n "$1" ]  #  If command-line argument present,
then            #+ then set start-string to it.
  str0="$1"
else            #  Else use PID of script as start-string.
  str0="$$"
fi
POS=2  # Starting from position 2 in the string.
LEN=8  # Extract eight characters.

str1=$( echo "$str0" | md5sum | md5sum )
#  Doubly scramble     ^^^^^^   ^^^^^^

randstring="${str1:$POS:$LEN}"

echo "$randstring"
```

Manipulating strings

Substring removal - ${string#substring} deletes shortest match, ${string##substring} deletes longest match.

```
stringZ=abcABC123ABCabc
#       |----|          shortest
#       |----------|    longest

echo ${stringZ#a*C}      # 123ABCabc
# Strip out shortest match between 'a' and 'C'.

echo ${stringZ##a*C}     # abc
# Strip out longest match between 'a' and 'C'.
```

Manipulating strings

Substring replacement - ${string/substring/replacement}, replace first *match* of $substring with $replacement.
Substring replacement - ${string//substring/replacement}, replace all matches of $substring with $replacement.

```
stringZ=abcABC123ABCabc

echo ${stringZ/abc/xyz}  # xyzABC123ABCabc
             # Replaces first match of 'abc' with 'xyz'.

echo ${stringZ//abc/xyz} # xyzABC123ABCxyz
             # Replaces all matches of 'abc' with # 'xyz'.
```

Outline

Parameter substitution

Manipulating and/or expanding variables:

- ${parameter}
- ${parameter-default} if parameter not set, use default.

```
echo ${username-‘whoami‘}
# Echoes the result of ‘whoami‘, if variable $username
# is still unset.

DEFAULT_FILENAME=generic.data
filename=${1-$DEFAULT_FILENAME}
#  if parameter $1 is not specified
```

Parameter substitution

`${parameter=default}` - If parameter not set, set it to *default*.

```
echo ${var=abc}   # abc
echo ${var=xyz}   # abc
# $var had already been set to abc, so it did not change.
```

Outline

for

Parameterized file list

```
#!/bin/bash

filename="*txt"

for file in $filename
do
 echo "Contents of $file"
 echo "---"
 cat "$file"
 echo
done
```

for

File expansion

```
#!/bin/bash
# Globbing = filename expansion.

for file in *
#           ^  Bash performs filename expansion
#+             on expressions that globbing recognizes.
do
    if [ -d "$file" ]; then
        echo "$file is a directory"
    fi
    if [ -f "$file" ]; then
        echo "$file is a regular file."
    fi
done

exit 0
```

for

Function

```
generate_list ()
{
  echo "one two three"
}

for word in $(generate_list)  # Let "word" grab output of function.
do
  echo "$word"
done
```

for

Counting to ten

```
# Using "seq" ...
for a in ‘seq 10‘
do
  echo -n "$a "
done

echo; echo
```

Counting to ten

```
# Now, let's do the same, using C-like syntax.

LIMIT=10
# Double parentheses, and naked "LIMIT"
for ((a=1; a <= LIMIT ; a++))
do
  echo -n "$a "
done

echo; echo
```

While to ten

```
#!/bin/bash

var0=0
LIMIT=10

while [ "$var0" -lt "$LIMIT" ]
do
  echo -n "$var0 "        # -n suppresses newline.

  var0=$(($var0+1))
done

echo
exit 0
```

Test to end

```
#!/bin/bash
                          # Equivalent to:
while [ "$var1" != "end" ] # while test "$var1" != "end"
do
  echo "Input variable #1 (end to exit) "
  read var1                    # Not 'read $var1' (why?).
  echo "variable #1 = $var1"  # Need quotes because of "#" . . .
  # If input is 'end', echoes it here.
  echo
done

exit 0
```

while

C-style while

```
LIMIT=10       # 10 iterations.
((a = 1))       # a=1

while (( a <= LIMIT ))   #  Double parentheses,
do                       #+ and no "$" preceding variables.
  echo -n "$a "
  ((a += 1))             # let "a+=1"
done

echo
exit 0
```

While and pipes

```
#!/bin/bash

ps aux | \
while read user pid cpu mem vsz rss tty stat start time command
do
    echo $pid $mem $command
done | sort -n -r -k2

# sorts by memory usage

exit 0
```

Reading files

```
#!/bin/bash

IFS=':' # internal field separator

while read account password uid gid gecos directory shell
do
    echo $uid $account
done < /etc/passwd

exit 0
```

until

This construct tests for a condition at the top of a loop, and keeps looping as long as that condition is *false* (opposite of *while loop*).

```
#!/bin/bash
LIMIT=10
var=0

until (( var > LIMIT ))
do
  echo -n "$var "
  (( var++ ))
done    # 0 1 2 3 4 5 6 7 8 9 10


exit 0
```

Loop control

The `break` and `continue` loop control commands correspond exactly to their counterparts in other programming languages.

```
LIMIT=19  # Upper limit
echo "Printing Numbers 1 through 20 (but not 3 and 11)."
a=0
while [ $a -le "$LIMIT" ]
do
 a=$(($a+1))

 if [ "$a" -eq 3 ] || [ "$a" -eq 11 ]; then
   continue
 fi

 echo -n "$a "   # This will not execute for 3 and 11.
done
```

Outline

Functions

Simple functions

```
fun () { echo "This is a function"; echo; }

foo() {
    echo "foo"
}

fun
foo
```

Arguments (1/2)

```
#!/bin/bash
DEFAULT=default
func2 () {
   if [ -z "$1" ]; then
     echo "-Parameter #1 is zero length.-"
   else
     echo "-Parameter #1 is \"$1\".-"
   fi
   variable=${1-$DEFAULT}
   echo "variable = $variable"
   if [ "$2" ]; then
     echo "-Parameter #2 is \"$2\".-"
   fi

   return 0
}
```

Arguments (2/2)

```
#!/bin/bash
echo "Two parameters passed."
func2 first second  # Called with two params
echo

echo "\"\" \"second\" passed."
func2 "" second  # Called with zero-length first parameter
echo             # and ASCII string as a second one.

exit 0
```

Functions

Functions return a value, called an *exit status*. This is analogous to the exit status returned by a command.

Exit status

```
E_PARAM_ERR=250 # if no parameter
foo () {
   if [ -z "$1" ]; then
       return $E_PARAM_ERR
   fi
   return 0
}

foo ; res=$?
if [ "$res" -eq $E_PARAM_ERR ]; then
    echo "Missing parameter ..."
fi
exit 0
```

Functions

In contrast to C, a Bash variable declared inside a function is local ONLY if declared as such.

Global or local

```
#!/bin/bash
func () {
  local loc_var=23       # Declared as local variable.
  global_var=999         # Not declared as local.
 }
func

echo "\"loc_var\" outside function = $loc_var"

echo "\"global_var\" outside function = $global_var"

exit 0
```

Outline

Arrays

Sparse arrays

```
#!/bin/bash

area[11]=23
area[51]=UFOs

echo -n "area[11] = "
echo ${area[11]}    #  {curly brackets} needed.

echo "Contents of area[51] are ${area[51]}."

# Contents of uninitialized array variable print blank (null variable).
echo -n "area[43] = "
echo ${area[43]}
echo "(area[43] unassigned)"
```

Arrays

```
#!/bin/bash
# Quoting permits embedding whitespace within individual
#+ array elements.
array2=( [0]="first element" [1]="second element"
         [3]="fourth element" )

echo ${array2[0]}   # first element
echo ${array2[1]}   # second element
echo ${array2[2]}   # Skipped in initialization, and therefore null.
echo ${array2[3]}   # fourth element
echo ${#array2[0]}  # 13    (length of first element)
echo ${#array2[*]}  # 3     (number of elements in array)

exit
```