BASH as a Modern Programming Language

Jason Brittain
eBay

Why BASH instead of another programming language?

- Very apt when moving files around, controlling processes, shell uses
- Is the default OS shell for all OSs except Windows
- WORA except for Windows, but then we have Cygwin
- BASH is effectively the same on all OSs
- Does not need to be upgraded first before a new program can run
- Scripts can be very small, while implementing a lot
- Works well with every other programming language
- Easily invokes a myriad of native binaries, both platform specific and multiplatform

This is shell code (just an example):

```
numberToWord() {
    local arg1="$1"
    local value
    if ( test "$arg1" == "1" ); then
        value="one"
    else if ( test "$arg1" == "2" ); then
            value="two"
        fi
    fi
    echo "$value"
}

printNumberOfItems() {
    /bin/echo -n "I have this many items: "
    numberToWord 2
}
```

Conditionals: Some supported syntaxes

```
if ( test "$variable" == "one" ); then
    doSomething
fi

if (( "$variable" == "one" )); then
    doSomething
fi

if [ "$variable" == "one" ]; then
    doSomething
fi

if test "$variable" == "two"; then
    doSomething
fi

[ "$variable" == "one" ] && doSomething
```

Conditionals: Regular expressions

```
if [[ $myString =~ /etc/rc\d*..d ]]; then
    doSomething
fi
```

As long as you don't run on BASH 3.1 (and earlier), you may perform case insensitive regex conditionals like this:

```
shopt -s nocasematch
if [[ "$myString" =~ (friday) ]]; then
  echo "Match!"
fi
shopt -u nocasematch
```

.. just make sure you use the double square bracket, and the "=~".

Conditionals: Checking for zero-length string & unset string

If $ENV_VAR is either unset or set to "", these will print "empty or unset":

```
if [ -z "$ENV_VAR" ]; then echo "empty or unset"; fi
```

```
if [ "$ENV_VAR" == "" ]; then echo "empty or unset"; fi
```

If $ENV_VAR is unset, this will detect just that case, and print "unset":

```
if [ -z "${ENV_VAR+xxx}" ]; then echo "unset"; fi
```

Loops: for loops

Iterating over a set of strings:

```
for x in one two three; do echo "doing $x"; done
```

Looping from 1 to 100:

```
for i in $(seq 1 100); do echo "$i."; done
```

Another 1 to 100 for loop syntax:

```
max=100
for (( i=1; i<=$max; i++ )); do echo $i; done
```

Loops: while loops

Iterating over a set of strings:

```
while [ "$myVar" != "xxxx" ]; do myVar="${myVar}x"; echo $myVar; done
x
xx
xxx
xxxx
```

Forever loop:

```
while [ true ] ; do date ; sleep 1; done
Fri Jul 26 00:52:21 PDT 2013
Fri Jul 26 00:52:22 PDT 2013
Fri Jul 26 00:52:23 PDT 2013
...
```

Variable assignment: Setting a default if the var isn’t set

Set PORT to 8080 if it isn’t set:

```
PORT="${PORT-8080}"

-- OR --

PORT="${PORT:-8080}"
```

Another example of setting the value if the variable is not already set:

```
echo ${JPDA_TRANSPORT:="dt_socket"}
```

Variable assignment: Append to value only if the var is set

Prepend another path and colon, but only if LD_LIBRARY_PATH is set:

```
export
LD_LIBRARY_PATH=$HOME/myapi/lib${LD_LIBRARY_PATH+:$LD_LIBRARY_PATH}
```

The '+' means that only if LD_LIBRARY_PATH is set, append what follows.

Variable assignment: Search and replace in a value

Replacing the first occurrence of a string:

```
VAR="MyMyString”
VAR=${VAR/My/Your}
echo $VAR
YourMyString
```

Replacing all occurrences of a string:

```
VAR="MyMyString”
VAR=${VAR//My/Your}
echo $VAR
YourYourString
```

Variable assignment: All occurrences of a char in a string

Remove all regular space characters from the string:

```
VAR="123 456”
VAR=${VAR// /}
echo $VAR
123456
```

Another way that trims out the tabs as well:

```
VAR=${VAR//[[:space:]]/}
```

Backslash escape all forward slashes in a string:

```
CB=${CATALINA_BASE//\//\\\/}
```

Variable assignment: Function to trim white space

Trim function that will trim all leading and trailing white space.
Invoke it like this: `trim "   abc   "`
Changes value to: `”abc"`

```
trim() {
  local str=$1

  # Remove leading whitespace characters.
  str="${str#"${str%%[![:space:]]*}"}"

  # Remove trailing whitespace characters.
  str="${str%"${str##*[![:space:]]}"}"

  /bin/echo -n "$str"
}
```

Variable assignment: Substring and string length

Substring from the beginning of a value:

```
VAR="bashful"
echo ${VAR:0:4}
bash
```

Substring from a specified starting character, specified length:

```
echo ${VAR:4:3}
ful
```

Get the length of a string:

```
echo ${#VAR}
7
```

Arrays: Simple instantiation and referencing

To declare an integer-indexed array and instantiate an empty one:

```
$ myArray=()
$ myArray[0]="zero"
$ myArray[1]="one"
$ myArray[2]="two"
$ echo ${myArray[1]}
one
```

Arrays: Simple initialization

Here’s how to build the same array in one line of script code:

```
myArray=(zero one two)

-- OR --

myArray=([0]="zero" [1]="one" [2]="two")
```

Arrays: Declaring array variables, and scope

To declare the same kind of array variable, but not initialize one:

```
typeset -a myArray

-- OR --

declare -a myArray
```

NOTE: Using declare inside a function causes the declared variable to be locally scoped (not visible globally), and using it outside a function causes the declared variable to be globally visible.

Arrays: Accessing, Appending, Converting

To get the number of elements nn the array:

```
$ echo "Number of elements allocated in myArray is: ${#myArray[@]}"
Number of elements allocated in myArray is: 3
```

To add a new element at the end of an existing array:

```
myArray[${#myArray[@]}]="newValue"
```

To print the array as a string:

```
myArrayAsString=${myArray[@]}
echo $myArrayAsString
firstValue secondValue newValue
```

Arrays: Iterating through

Iterating through an array of plugin names:

```
printf "Plugins: "
for pn in ${plugins[@]}; do
  printf "$pn "
done
echo
```

Output:

```
Plugins: myplugin1 myplugin2 myplugin3
```

Arrays: Arrays of Arrays

Iterating through elements of secondary arrays whose keys are elements of a primary array:

$plugins is an array containing a list of all of the plugin names.

$<pluginName>Commands is an array containing a list of commands for the plugin.

```
for pn in ${plugins[@]}; do
  # Loop through all commands that are part of this plugin.
  cmds=$(echo \$\{${pn}Commands\[\@\]\})
  cmds=$(eval for cm in $cmds\; do echo \$cm\; done)
  for co in $cmds; do
    echo "Command: $co"
  done
done
```

Arrays: Removing an element, like Perl’s splice()

To remove the third element from an array of four elements:

```
a=(one two three four)
let i=2 # Remove element 2 (0-based)
if [ "${#a[@]}" == "$i" ]; then
  a=(${a[@]:0:$(($i))})
else
  a=(${a[@]:0:$i} ${a[@]:$(($i + 1))})
fi
```

Arrays: Sorting

To sort an array:

```
$ a=(one two three four five)
$ a=($(echo ${a[*]} | tr ' ' "\n" | sort)) # 1-liner sort
$ echo ${a[*]}
five four one three two
```

Associative Arrays / Hashtables

This works only in BASH v4 and higher:

```
declare -A ASSOC
ASSOC[First]="first element"
ASSOC[Hello]="second element"
ASSOC[Smeagol]="Gollum"

echo "Smeagol's real name is ${ASSOC['Smeagol']}"
Smeagol's real name is Gollum
```

Files: Parsing files

```
set -u
flist=files.txt
find . -name \*.java > $flist

((totalLines=0))
((totalFiles=0))
while read aFile; do
   ((thisLineCount=0))
   while read aLine; do
      ((thisLineCount=thisLineCount+1))
   done < $aFile
   ((totalLines=totalLines+thisLineCount))
   ((totalFiles=totalFiles+1))
   echo "$thisLineCount $aFile"
done < $flist
echo "total files: " $totalFiles
echo "total lines: " $totalLines
```

Files: Canonicalizing file paths

```
getCanonicalFile() {
  local origFile="$1"
  local readlink="`which readlink`"
  PATH=`$readlink -f $0`
  echo $PATH
}
```

.. though on MacOS –f means something else.

OOP: Functions / Methods / Subroutines

In BASH, these are called "functions". Declare one like this:

```
myFunction() {
  echo "It ran."
}
```

Call them like this:

```
myFunction
```

OOP: Function return values

You may only return a numeric from a BASH function, and the smallest value you may return is zero:

```
myFunction() {
  return 2
}

# Call it and receive the return value..
myFunction
let returnValue=$?
```

If you want to return -1, the only way to write that is:

```
return -- -1
```

.. and even then, the actual return value is 255.

OOP: Functions: Returning a string value

In order to return a string, you must do this:

```
myFunction() {
  echo "This string is the returned value."
}

RETURN_VALUE=$(myFunction)
echo $RETURN_VALUE
```

OOP: Functions: Passing arguments

```
calledFunction() {
  ARG1="$1"
  ARG2="$2"
  echo "calledFunction() was called with arg1=$ARG1 arg2=$ARG2"
}

callerFunction() {
  # Call the function, passing it two argument values.
  calledFunction "value1" "value2"
}
```

OOP: Functions: Returning an array of values

```
myFuncThatReturnsAnArray() {
  local myValArray=("one" "two" "three")
  echo "${myValArray[@]}"
  return 0
}

callerFunction() {
  # Call the function and receive an array of return values.
  local returnedValArray=( $(mFuncThatReturnsAnArray) )

  # Loop through the values, printing each one.
  for v in $returnedValArray; do
    echo $v
  done
}
```

OOP: Functions: Returning multiple values to local vars

```
calledFunction() {
  variableToBeModified1="magic"
  variableToBeModified2="incantation"
}

callerFunction() {
  local variableToBeModified1 variableToBeModified2
  echo "Before obtaining a value: $variableToBeModified1 $variableToBeModified2"
  calledFunction
  echo "Before obtaining a value: $variableToBeModified1 $variableToBeModified2"
}

callerFunction
echo "After function scope: $variableToBeModified1 $variableToBeModified2”
```

OOP: Classes: Namespace scoping of functions

```
class1() { echo "class 1."; func() { echo "func 1"; }; }
class2() { echo "class 2."; func() { echo "func 2"; }; }

class1; eval func\(\) \{ class1\; func\; \}
class2; eval func\(\) \{ class2\; func\; \}
```

.. then you can switch between classes and use the functions for the current class:

```
$ class1
class 1.
$ func
func 1
$ class2
class 2.
$ func
func 2
```

Introspection

Listing defined variables whose names are prefixed with a known string:

```
$ prefix_foo="one"
$ prefix_bar="two"
$ echo "${!prefix_*}"
prefix_bar prefix_foo
```

Detecting whether a function is defined:

```
if type myFunc >/dev/null 2>&1; then
  echo "The myFunc() function is defined."
fi
```

Debugging

```
$ set -x
++ update_terminal_cwd
++ local 'SEARCH= '
++ local REPLACE=%20
++ local PWD_URL=file://mymachine/Users/jasonb/examples
++ printf '\e]7;%s\a' file://mymachine/Users/jasonb/examples
```

See Also:

BASH Debugger

http://bashdb.sourceforge.net

THANKS!

Jason Brittain jason.brittain@gmail.com

Download this presentation at:

http://goo.gl/pcoj4s

Related Reading

BASH: Bourne Again SHell scripting

http://tiswww.case.edu/php/chet/bash/bashtop.html

BASH For Loop Examples

http://www.cyberciti.biz/faq/bash-for-loop

Advanced Bash-Scripting Guide: Chapter 24. Functions (very nice!)

http://tldp.org/LDP/abs/html/functions.html

more…

Related Reading (continued)

Hash Tables / Associative Arrays in BASH v3

http://stackoverflow.com/questions/688849/associative-arrays-in-shell-scripts

BASH Debugger

http://bashdb.sourceforge.net

JSON.sh: Very simple BASH parsing of JSON data (recommended)

https://github.com/dominictarr/JSON.sh

Using JSON with BASH

http://bokov.net/weblog/programming/using-json-with-bash/

Parsing JSON with sed and awk

http://stackoverflow.com/questions/1955505/parsing-json-with-sed-and-awk

Detect the OS from a BASH Script

http://stackoverflow.com/questions/394230/detect-the-os-from-a-bash-script

more…

Related Reading (continued)

Testing Bash Scripts

http://stackoverflow.com/questions/1339416/testing-bash-scripts

BASH Socket Programming with /dev/tcp

http://thesmithfam.org/blog/2006/05/23/bash-socket-programming-with-devtcp-2/

gtkdialog

http://code.google.com/p/gtkdialog

http://xpt.sourceforge.net/techdocs/language/gtkdialog/gtkde03-GtkdialogUserManual/index.html

How To Make a GUI for BASH Scripts (Zenity, wish, dialog, bashgui, etc)

http://stackoverflow.com/questions/928019/how-to-make-a-gui-for-bash-scripts

Bash Plugin Architecture: BPA

http://code.google.com/p/b-p-a