Bash Scripts

James Kress [1]

[1]Department of Computer Science
Boise State University
Boise, Idaho

February 6, 2013

BOISE STATE UNIVERSITY
COLLEGE OF ENGINEERING
Department of Computer Science

Creating a Script

Hello World

```
#!/bin/bash
echo "Hello World!"
```

Output

```
james@cptnjtk:~/Dropbox/cs253 Grading/scripting/code$ ./helloWorld.bash
Hello World
```

Running a Script

Steps to execution

- Make it executable
- ./scriptName

Without executable permissions

```
james@cptnjtk:~/Dropbox/cs253 Grading/scripting/code$ ls  -l
total 12
-rw-rw-r-- 1 james james 2507 Feb  3 08:59 calcTest.bash
-rw-rw-r-- 1 james james  161 Feb  3 08:58 divline.bash
-rw-rw-r-- 1 james james   32 Feb  3 09:17 helloWorld.bash
```

Running a Script

Steps to execution

- Make it executable
- ./scriptName

With executable permissions

```
james@cptnjtk:~/Dropbox/cs253 Grading/scripting/code$ chmod +x helloWorld.bash
james@cptnjtk:~/Dropbox/cs253 Grading/scripting/code$ ls -l
total 12
-rw-rw-r-- 1 james james 2507 Feb  3 08:59 calcTest.bash
-rw-rw-r-- 1 james james  161 Feb  3 08:58 divline.bash
-rwxrwxr-x 1 james james   32 Feb  3 09:17 helloWorld.bash
```

Positional Parameters

- Hold the command line arguments to scripts
  - Named: 1, 2, 3, etc.
  - Denoted: $1, $2, $3,...
- Special Parameters: 0, #, @
  - 0: Holds the name of the script as it was invoked
  - #: Holds the number of command line arguments
  - @: Expands to the positional parameters, starting from one

Single Command Line Argument (pp1.bash)

```
#!/bin/bash
echo Command Line Argument $#: $1
```

Positional Parameters

- Hold the command line arguments to scripts
  - Named: 1, 2, 3, etc.
  - Denoted: $1, $2, $3,...
- Special Parameters: 0, #, @
  - 0: Holds the name of the script as it was invoked
  - #: Holds the number of command line arguments
  - @: Expands to the positional parameters, starting from one

Multiple Command Line Argument (pp2.bash)

```
#!/bin/bash
echo
echo Number of Arguments: $#
echo
echo $1 :: $2 :: $3 :: $4
echo
```

Positional Parameters

- Hold the command line arguments to scripts
  - Named: 1, 2, 3, etc.
  - Denoted: $1, $2, $3,...
- Special Parameters: 0, #, @
  - 0: Holds the name of the script as it was invoked
  - #: Holds the number of command line arguments
  - @: Expands to the positional parameters, starting from one

Special Parameter: 0 (pps1.bash)

```
#!/bin/bash
echo Command Line Argument $#: $0
```

Positional Parameters

- Hold the command line arguments to scripts
  - Named: 1, 2, 3, etc.
  - Denoted: $1, $2, $3,...
- Special Parameters: 0, #, @
  - 0: Holds the name of the script as it was invoked
  - #: Holds the number of command line arguments
  - @: Expands to the positional parameters, starting from one

Special Parameter: @ (divline.bash)

```
#!/bin/bash
echo
echo ========================================
echo $@
echo ========================================
echo
```

Positional Parameters

- Hold the command line arguments to scripts
  - Named: 1, 2, 3, etc.
  - Denoted: $1, $2, $3,...
- Special Parameters: 0, #, @
  - 0: Holds the name of the script as it was invoked
  - #: Holds the number of command line arguments
  - @: Expands to the positional parameters, starting from one

Special Parameter: @ (pps2.bash)

```
#!/bin/bash
echo
echo Number of Arguments: $#
echo
echo $0 :: $1 :: $2 :: $3 :: $4
echo
echo $@
echo
```

Bash Supported Flow Control Concepts

- if/else
  - execute a list of statements if a certain condition is/is not true
- for
  - execute a list of statements a fixed number of times
- while
  - execute a list of statements repeatedly while a certain condition holds true
- until
  - executes a list of statements repeatedly until a certain condition holds true
- case
  - execute one of several lists of statements depending on the value of a variable
- **In addition, bash provides a new type of flow-control construct:**
  - select
    - Allows the user to select one of a list of possibilities from a menu

Bash Supported Flow Control Concepts

- if/else
  - execute a list of statements if a certain condition is/is not true

Example (fc1.bash)

```
#!/bin/bash

if [ "a" \> "b" ]; then
echo "a > b"
else
echo "a < b"
fi
```

Bash Supported Flow Control Concepts

- for
  - execute a list of statements a fixed number of times

Example (fc2.bash)

```
#!/bin/bash

for f in *.bash; do
echo $f
done
```

Bash Supported Flow Control Concepts

- while
  - execute a list of statements repeatedly while a certain condition holds true

Example (fc3.bash)

```
#!/bin/bash

i=0;
while [ $i -lt 10 ]; do
let i++
echo $i
done
```

Bash Supported Flow Control Concepts

- until
  - executes a list of statements repeatedly until a certain condition holds true

Example (fc4.bash)

```
#!/bin/bash

until [ -z "$1" ]
do
echo $1
shift
done
```

Bash Supported Flow Control Concepts

- case
  - execute one of several lists of statements depending on the value of a variable

Example (fc5.bash)

```
#!/bin/bash

case "$1" in
        start)
            echo "in start..."
            ;;
        stop)
            echo "in stop..."
            ;;
        status)
            echo "in status..."
            ;;
        restart)
            echo "in restart"
            ;;
        *)
            echo $"Usage: $0 {start|stop|restart|status}"
            exit 1

esac
```

Bash Supported Flow Control Concepts

- **In addition, bash provides a new type of flow-control construct:**
  - select
    - Allows the user to select one of a list of possibilities from a menu

Example (fc6.bash)

```
#!/bin/bash

printf "Select your facorite animal:\n"
select term in \
    'Dolphin' \
    'Panda' \
    'Tiger' \
    'Bronco'
do
    case $REPLY in
        1 ) TERM=Dolphin;;
        2 ) TERM=Panda ;;
        3 ) TERM=Tiger ;;
        4 ) TERM=Bronco ;;
        * ) printf "invalid." ;;
    esac
    if [[ -n $term ]]; then
        printf "Your favorite animal is: $TERM\n"
        break
    fi
done
```

File Attributes

Operator	True if...
-a *file*	*file* exists
-d *file*	-d *file* exists and is a directory
-e *file*	*file* exists; same as -a
-f *file*	*file* exists and is a regular file (not a directory, ...)
-r *file*	You have read permission on *file*
-s *file*	*file* exists and is not empty
-w *file*	You have write permission on *file*
-x *file*	You have execute permission on *file*
-N *file*	*file* was modified since last read
-O *file*	You own *file*
-G *file*	file's group ID matches yours
file1 -nt file2	file1 is newer than file2
file1 -ot file2	file1 is older than file2

- fa1.bash

Integer Conditionals

Test	Comparison
-lt	Less than
-le	Less than or equal
-eq	Equal
-ge	Greater than or equal
-gt	Greater than
-ne	Not equal

Arithmetic Operators

Operator	Meaning
++	Increment by one (prefix and postfix)
–	Decrement by one (prefix and postfix)
+	Plus
-	Minus
*	Multiplication
/	Division (with truncation)
%	Remainder
>>	Bit-shift left
<<	Bit-shift right
&	Bitwise and
|	Bitwise or
~	Bitwise not
!	Logical not
∧	Bitwise exclusive or
,	Sequential evaluation

String Comparison Operators

Operator	**True if...**
str1 = str2	str1 matches str2
str1 != str2	str1 does not match str2
str1 <str2	str1 is less than str2
str1 >str2	str1 is greater than str2
-n str1	str1 is not null (has length greater than 0)
-z str1	str1 is null (has length 0)

Typed Variables

Option	Meaning
-a	The variables are treated as arrays
-f	Use function names only
-F	Display function names without definitions
-i	The variables are treated as integers
-r	Marks the variables read-only
-x	Marks the variables for export via the environment

Examples

- tv1.bash

Typed Variables

Option	Meaning
-a	The variables are treated as arrays
-f	Use function names only
-F	Display function names without definitions
-i	The variables are treated as integers
-r	Marks the variables read-only
-x	Marks the variables for export via the environment

Examples

- tv1.bash
- tv2.bash

Typed Variables

Option	Meaning
-a	The variables are treated as arrays
-f	Use function names only
-F	Display function names without definitions
-i	The variables are treated as integers
-r	Marks the variables read-only
-x	Marks the variables for export via the environment

Examples

- tv1.bash
- tv2.bash
- tv3.bash

Typed Variables

Option	Meaning
-a	The variables are treated as arrays
-f	Use function names only
-F	Display function names without definitions
-i	The variables are treated as integers
-r	Marks the variables read-only
-x	Marks the variables for export via the environment

Examples

- tv1.bash
- tv2.bash
- tv3.bash
- tv4.bash

I/O Redirection and Pipes

Examples of I/O and Pipes

- calcTest
- process
- sendGrade

Bash Scripts

James Kress [1]

[1]Department of Computer Science
Boise State University
Boise, Idaho

February 6, 2013

BOISE STATE UNIVERSITY
COLLEGE OF ENGINEERING
Department of Computer Science