Basic Shell Programming

W. H. Bell
W.Bell@cern.ch

Abstract

A basic overview of the BASH scripting language is given. The language is introduced using examples covering a few aspects at a time. Reference tables are provided for commonly used BASH functionality.

W. H. Bell 2010

Introduction to BASH

"Bash is an sh-compatible command language interpreter that executes commands read from the standard input or from a file. Bash also incorporates useful features from the Korn and C shells (ksh and csh)."

- Compound Commands
  - Loops: for, while, until,
  - Conditional Statements: select, case, if
- File Operations
  - Conditional Expressions
  - Reading and Writing to Files
- Expansion and Its Uses
- Command line arguments
- String Operations
  - Conditional Expressions
  - Uses of Parameter Expansion
  - Other Commands
- Functions
- External Commands

Typing `man bash` gives a lot of extra information.

Starting a Shell

- Each user has a default shell: defined in passwd file.

```
]$ echo $SHELL
/bin/bash
```

- Can start another shell from the default shell either by typing the name of the shell:

```
]$ bash
```

or by creating a file with the PATH to the shell at the top of the file, eg:

```
]$ vi example\_01.bash
#!/bin/bash
~
```

A Basic Shell Program

- First create a file containing the script.

```
#!/bin/bash
echo "In the beginning..."
```

- The file does not need to have a specific file extension. (For the examples given in this course the BASH scripts have a suffix of .sh)
- Then make the file executable

```
]$ chmod u+x ex1.sh
```

- And finally run the script.

```
]$ ./ex1.sh
In the beginning...
```

White Spaces and New Lines

- Spaces and new lines are very important in shell programming.
- In many languages white spaces are ignored i.e. the compiler or interpreter skips over them.
- This is not so in basic shell programming. Try taking some of the white spaces out of the following examples and see the result.

Compound Commands: Loops: `for`

```
#!/bin/bash

word="a b c"
i=0

# Read each character from $word and and assign
# it to $name
for name in $word; do

  # Use 'let' to increment i.
  let i++

  # Print the value of $name
  echo $name

done

echo "Looped $i times."
```

example_02.sh: A program to demonstrate a type 1 `for` loop.

Compound Commands: Loops: `for`

```
#!/bin/bash

# Loop from 0 to 9.
for((i=0; i<10; i++)) ; do

  # Append the string form of i to the
  # end of the string j
  j=$j$i

done

echo $j
```

example_03.sh: A program to demonstrate a type 2 `for` loop.

Compound Commands: Loops: `while`, `until`

```
#!/bin/bash

nloops=3
i=0

echo "while loop"
while [[ $i<$nloops ]]; do
    echo $i
    let i++
done

echo
echo "until loop"
i=0
until [[ $i>$nloops ]]; do
    echo $i
    let i++
done
```

example_04.sh: A program to demonstrate `while` and `until` loops.

Compound Commands: Conditional Statements: `if`

```
#!/bin/bash

for ((i=0;i<3;i++)) do
    if [[ $i == 1 ]]; then
        echo "Turnip"
    elif [[ $i == 2 ]]; then
        echo "Potato"
    else
        echo "Carrot"
    fi
done
```

example_05.sh: A program to demonstrate `if`, `elif`, `else` conditional statements.

File Operations: Conditional Expressions

```
#!/bin/bash

files="test test_dir test_link"

for file in $files; do
  if [[ -a $file ]]; then
    echo "File $file Exists"
  fi

  if [[ -f $file ]]; then
    echo "File $file is a regular file"
  fi

  if [[ -d $file ]]; then
    echo "File $file is a directory"
  fi

  if [[ -h $file ]]; then
    echo "File $file is a symbolic link"
  fi
done
```

example_06.sh: A program to that uses conditional expressions to test for the presence of a file.

File Operations: Conditional Expressions

- Before running example 6.

```
]$ touch test; mkdir test_dir
]$ ln -s test test_link
```

Usage	Result
-a *file*	True if *file* exists
-b *file*	True if *file* exists and is a block special file
-c *file*	True if *file* exists and is a character special file
-d *file*	True if *file* exists and is a directory.
-e *file*	True if *file* exists
-f *file*	True if *file* exists and is a regular *file*
-g *file*	True if *file* exists and is set-group-id
-h *file*	True if *file* exists and is a symbolic link
-k *file*	True if *file* exists and its "sticky" bit is set
-p *file*	True if *file* exists and is a named pipe (FIFO)
-r *file*	True if *file* exists and is readable
-s *file*	True if *file* exists and has a size greater than zero
-u *file*	True if *file* exists and its set-user-id bit is set
-w *file*	True if *file* exists and is writable
-x *file*	True if *file* exists and is executable
-O *file*	True if *file* exists and is owned by the effective user id
-G *file*	True if *file* exists and is owned by the effective group id
-L *file*	True if *file* exists and is a symbolic link
-S *file*	True if *file* exists and is a socket
-N *file*	True if *file* exists and has been modified since it was last read
file1 -nt *file2*	True if *file1* is newer than *file2*
file1 -ot *file2*	True if *file1* is older than *file2*

File Operations: Reading and Writing Files

```
#!/bin/bash

words="electron muon tau"
outputfile="test.out"

rm -f $outputfile

for name in $words; do
  echo $name >> $outputfile
done

echo "cat $outputfile:"
cat $outputfile

echo
echo "Reading the words back in."
for name in $(<$outputfile); do
  str="$str$name, "
done
echo $str
```

example_07.sh: A program demonstrate file i/o using output redirection.

Expansion and Its Uses

```
#!/bin/bash

echo "Brace expansion - 1{2,3,4}5:"
echo 1{2,3,4}5

echo "Tilde Expansion HOME - ~:"
echo ~

echo "Tilde Expansion PWD - ~+:"
echo ~+

echo "Tilde Expansion wbell's HOME:"
echo ~wbell/
```

example_08.sh: A program to demonstrate brace and tilde expansion.

Expansion and Its Uses

```
#!/bin/bash

i=1
j=4

i=$((++i*j))

echo $i
```

example_09.sh: A program to demonstrate arithmetic expansion

- Provides functionality of `let` i.e. `+`, `-`, `*`, `/`, `**`
- Anything more complicated i.e. sine, sqrt etc: use `bc` or `perl`

Command Line Arguments

```
#!/bin/bash

echo "The number of args following the command = $#";

for arg in $* ; do
    str="$str $arg"
done

echo "$0$str"
```

example_10.sh: A simple program to illustrating how command line arguments can be read inside a shell program.

- Commands can be accessed directly by using $\$n$ where n is the number of the command. E.g.

  ```
  echo "The first argument is $1"
  ```

- Be careful to check the value is defined before using it.

String Operations: Conditional Expressions

```
#!/bin/bash

if [[ -n $CHECK_ME ]]; then
  echo "CHECK_ME = $CHECK_ME"
else
  echo "CHECK_ME is unset."
fi
```

example_11.sh: A program to demonstrate the use of conditional string operators.

To test the example program try setting and unsetting the CHECK_ME environmental variable:

```
]$ export CHECK_ME=1
]$ unset CHECK_ME
```

Run the script before and after the environmental variable is set.

String Operations: Conditional Expressions

Usage	Result
-z *string*	True if the length of string is zero
-n *string*	True if the length of string is non-zero
string1 == *string2*	True if the strings are equal
string1 != *string2*	True if the strings are not equal

A summary of the most useful conditional string operators.

String Operations: Parameter Expansion

```
#!/bin/bash

somestring=abcdef

echo "length = ${#somestring}"

i=2
echo "After $i characters ${somestring:$i}"
echo "Before $i characters ${somestring: -$i}"

j=2
echo "From char $i to of length $j ${somestring:$i:$j}"
```

example_12.sh: A script demonstrating substring selection via Parameter Expansion.

String Operations: Parameter Expansion

```
#!/bin/bash

parameter="filename.dat"
word=".dat"

remainder=${parameter%$word}

echo "parameter=$parameter word=$word"
echo "remainder=$remainder"
```

example_13.sh: A script to remove part of a string using Parameter Expansion.

- There are two types of this sort of parameter expansion.
  - `${parameter#word}` - Matching the beginning.
  - `${parameter%word}` - Matching the end.
- One # or % character for the shortest and two for the longest matching case.

String Operations: Parameter Expansion

```
#!/bin/bash

parameter="filename.dat"
pattern=".dat"
string=".root"

new_filename=${parameter/$pattern/$string}

echo "parameter=$parameter pattern=$pattern"
echo "string=$string"
echo "new_filename=$new_filename"
```

example_14.sh: A script to demonstrate string substitution using Parameter Expansion.

- The `pattern` is a pattern and not a `word`.

String Operations: Pattern Matching

```
#!/bin/bash

filename1="string"
filename2=" string "
match1=$filename1
match2=" string"

if [[ "$filename1" == "$match1" ]]; then
  echo "\"$match1\" matches \"$filename1\""
fi

if [[ "$filename2" == *"$match1"* ]]; then
  echo "\"$match1\" matches \"$filename2\""
fi

if [[ "$filename2" == "$match2"* ]]; then
  echo "\"$match2\" matches \"$filename2\""
fi
```

example_15.sh: A script demonstrating string pattern matching.

String Operations: Other Commands

A range of commands outside of the bash language can be used to operate on strings.

- `expr` - Provides many string operations together with logic and numeric functions. (Type `info expr` for more information.)
- `sed` - A stream editor used to perform operations on text.
- `awk` - Is the interpreter for The AWK Programming Language.

Functions

```
...
usage() {
  echo ""
  echo " Usage: $0 <directory>"
  echo ""
  exit 1
}

baddir() {
  echo ""
  echo " $1 can not be listed"
  echo ""
}
...
```

An extract from example_16.sh: Two functions: one using a global parameter, the other a local one.

Functions

```
...
# Check at least one argument is given
if [[ -z $1 ]]; then
  usage
fi
...
else
  baddir $dir
fi
...
```

An extract from example_16.sh: Calling the two functions previously defined.

External Commands

```
...
files=$(ls $dir)
if [[ $? == 0 ]]; then
...
```

An extract from example_16.sh: Demonstrating how to execute external commands.

- `$?` Contains the return value from the command.
- The return statement is the last return value. Therefore do not put an `if` statement between the command and the test on `$?`