IPSL BootCamp: Bash scripting*

Institute Pierre Simone Laplace, IPSL BootCamp

The content of the BootCamp can be found in:
https://forge.ipsl.jussieu.fr/igcmg_doc/wiki/Train

March 25, 2016

Contents

1 Introduction **1**

2 Basic Examples **1**
2.1 before starting . . . 1
2.2 if . . . 2
2.3 pipe . . . 3
2.4 loop . . . 4
2.5 case . . . 6
2.6 complex script . . . 7

3 Function **8**

4 Useful links **8**

1 Introduction

Bash scripting is the common way to automatize repetitive tasks in a unix/linux system. It basically consists of the writing of an executable file filled with instructions of the system. As a interpreted language it has standard programming structures such as `if`, `do`.

It is very useful and it is massively used for all the community of linux users.

There are different scripting `SHELL` environments: `sh`, `csh`, `bash`, `ksh`, ... These notes use `bash` (*'bourne again shell'*)

2 Basic Examples

Some generic examples of the most basic commands/structures are provided here.

2.1 before starting

Some very basic zero stuff:

- `#`: comment character
- `$`: starting character for variable. For a given variable `var`, there are only slight differences between `$var`, `${var}`

*Main author of this chapter: L. Fita, Laboratoire de Météorologie Dynamique (LMD).

- \: Character of continuation of line
- No need to indent, but highly recommendable
- Case Sensitive program
- No error message if the variable does not exist or has any value!
- `coreutils`: System provided powerful tiny tools. They are used with arguments (space separeted 'words' after their call) and modified they behavior with flags (`-[something]`)
- Extension of the scripts `.bash`

2.2 if

The boolean expressions are driven by `if`, `then`, `elif`, `else`, `fi`[1]

A numeric based 'if'. Let's create a file called `test.bash`

```
#!/bin/bash
# Numeric if
value=-4
large=true
if test ${value} -eq 1
then
  echo "one"
elif test ${value} -lt 0; then
  echo "Negative"
else
  echo "Large"
  large=true
fi
```

Selection of `SHELL` environment
Comment
Shell only works with integers

'test' coreutil[a] evaluation of if condition

Printing on the terminal
';' for new line

[a] system provided tiny and powerful tools https://wiki.debian.org/coreutils

Steps to use the script `test.bash`:

1. Giving execution permits:

```
chmod u+x test.bash
```

2. Using it:

```
$ ./test.bash
Negative
```

If with a boolean variable:

```
# Boolean variable
if ${large}; then
  "Is large!"
  exit
fi
```

to exit the program

As results, when executing the script:

```
$ ./test.bash
Negative
Is large!
```

[1] equivalences

`-eq`	equal to	`-ls`	less than
`-gt`	great than	`-le`	less equal than
`-ge`	less equal than	`!`	not
`&&`	and	`||`	or

If with string variable:

```
# String if
value="one"
if test !  ${value} = 'one'; then
  echo "Different than 'one' !"
  exit
fi
```

'!' as 'not'

Checking file existence:

```
# File existence
dateval=`date +%Y%m%d%H%M%S`
filen=${dateval}'_file.txt'
if test !  -f ${filen}; then
  echo "File '"${filen}"' does not exist!"
  exit
fi
```

Use of ` `, to capture coreutil 'date' in a variable

'-f' coreutil 'test' option to check file existence

When using (after setting `large=false`):

```
$ ./test.bash
Negative
File '20160215002245_file.txt' does not exist!
```

2.3 pipe

'pipe': Concatenation of execution of linux instructions.

```
# Pipes
Nfiles=`ls -1 *bash| wc -l`
echo "Number of files:  "$Nfiles
```

'|' to connect consecutively instructions
'ls -1': 1 column output
'wc -l': coreutil to count, in this case lines

When used:

```
$ Nfiles=`ls -1 *bash| wc -l`
$ echo ${Nfiles}
1
```

2.4 loop

Standard loop is constructed using three basic words `for`, `do`, `done`:

```
# Incremental loop
i=1
Rangeloop=10
echo "Initial values"
while test $i -le ${Rangeloop}; do
  echo $i
  i=‘expr $i + 1‘
done
```

No spaces on definition of variables
Case-sensitive!

Loop initialization ';' for new line

Loop increment using coreutil ‘expr’

When used:

```
Initial values
1
2
3
4
5
6
7
8
9
10
```

Adding text file generation:

```
# Incremental loop
i=1
Rangeloop=10
files=”
echo "Initial values"
while test $i -le ${Rangeloop}; do
  echo $i
  iS=‘printf %02d $i‘
  filen=${iS}_file.txt
  echo $i > ${filen}
  if test $i -eq 1; then
    files=${filen}
  else
    files=${files}’:’${filen}
  fi
  i=‘expr $i + 1‘
done
```

Initialization of a string variable

‘printf’: coreutil format printing tool

'>' to write the left side result into a file
Starting an if

When used:

```
Initial values
1
2
3
4
5
6
7
8
9
10
```

Reading text files:

```
echo "quadratic values"
# 'variable' loop
for filen in *_file.txt; do
  val=`cat ${filen}`
  valpot=`expr ${val} '*' ${val}`
  echo ${valpot}
done
```

Getting all files with the given expression
'cat' coreutil to output file content
'*' here as multiply

When used:

```
quadratic values
1
4
9
16
25
36
49
64
81
100
```

Bulding loop with a given list of values:

```
echo "Content of files"
# 'assigned variables' loop
fs=`echo ${files} | tr ':'  '  '`
for filen in ${fs}; do
  echo ${filen}"..."
  cat ${filen}
done
```

Use of 'tr' coreutil substitution tool
substitution of ':' by spaces

When used:

```
Content of files
01_file.txt...
1
02_file.txt...
2
03_file.txt...
3
04_file.txt...
4
05_file.txt...
5
06_file.txt...
6
07_file.txt...
7
08_file.txt...
8
09_file.txt...
9
10_file.txt...
10
```

2.5 case

'case': conditional with multiple options. It is built with:`case`, `in`, `;;`, `*)` and `esac`

```
# Case
num=3
case ${num} in
  1)
    echo "one"
  ;;
  2)
    echo "two"
  ;;
  3)
    echo "three"
  ;;
  *)
    echo "other than one, two, three !"
  ;;
esac
```

Opening the inspection of variable
Actions for this value

End for this value

Default value (any of previous)

End of the instruction

When used:

```
three
```

2.6 complex script

Example of complex script that will count the number of days of each month between a period of two years. It will create a table with the results which will be output as a pdf using LaTeX

```
# Complet file name generation
Syear=2012
Eyear=2014
iyr=${Syear}
otex='table'
cat << EOF > ${otex}.tex
\\documentclass{article}

\\begin{document}
\\begin{center}
\\begin{tabular}{cccl}
{\\bfseries{year}} & {\\bfseries{month}} &
{\\bfseries{Ndays}} & {\\bfseries{file}}
\\\\ \\hline
EOF
while test $iyr -le ${Eyear}; do
  im=1
  while test $im -le 12; do
    imS=`printf %02d $im`
    d1=`date +%j -d"${iyr}${imS}01"`
    d2=`date +%j -d"${iyr}${imS}01 1 month"`
    Ndays=`expr $d2 - $d1`
    if test $d2 -eq 1; then Ndays=31; fi
    id=1
    while test $id -le ${Ndays}; do
    idS=`printf %02d $id`
    id=`expr $id + 1`
  done
  mon=`date +%b -d"${iyr}${imS}${idS}"`
  cat << EOF >> ${otex}.tex
$iyr & ${mon} & ${Ndays} &
$iyr$imS$idS.nc \\\\
EOF
    im=`expr $im + 1`
  done
  iyr=`expr $iyr + 1`
done
cat << EOF >> ${otex}.tex
\\end{tabular}
\\end{center}

\\end{document}
EOF
pdflatex ${otex}
pdflatex ${otex}
evince ${otex}.pdf &
```

Keeping the name of the file as a variable
Everything until 'EOF' will be kept inside the file
LaTeX code section
\\ to write in file '\'

End of the writing into the file

This part only works with `${iyr}` and not with `$iyr`

Summarized 'if' in a single line

Writing until 'EOF' after the last writing ('>>')
Shell values will be written into the file

Calling pdf-latex generation

showing pdf

When it is used only the LaTeX output is seen. The pdf `table.pdf` is also shown.

3 Function

Definition of a function:

```
# Function
function foldInf() {
# Function information of a folder
  fold=$1
  fend=$2

  NTfiles=‘ls -1 ${fold} | wc -l‘
  NEf=‘ls -1 ${fold}/*${fend} | wc -l‘
  DiskSpace=‘du -hsc ${fold} | grep total‘

  echo "Information of ’"${fold}"’ _______"
  echo "Total Number of files:  "${NTfiles}
  echo "Files ending ’"${fend}"’:  "${NEf}
  echo "Disk space:  "${DiskSpace}
}
```

Opening of the function

First argument (‘word’) of function

Listing in 1 column and counting lines
Total content ‘du’ system tool

End of the function

Using the function:

```
foldInf ./ _file.txt
```

Using function ‘foldInf’ with two arguments:
’./’ means the actual folder as first argument
’_file.txt’ end of files to use

```
$ foldInf ./ _file.txt
Information of ’./’ _______
Total Number of files:  34
Files ending ’_file.txt’:  10
Disk space:  412K total
```

4 Useful links

- Starting one: http://www.faqs.org/docs/air/tsshell.html
- The One: http://www.gnu.org/software/bash/manual/bashref.html
- Fairly complete: http://pubs.opengroup.org/onlinepubs/009695399/utilities/xcu_chap02.html
- Advance: http://tldp.org/LDP/abs/html/
- The core utils: https://wiki.debian.org/coreutils
- AWK: http://www.vectorsite.net/tsawk.html