Be in shell

Variables, scripts, compound commands

Pavel Fibich

pavel.fibich@prf.jcu.cz
dep. Botany - Na Zlaté stoce 1

03-2016

Přírodovědecká
fakulta
Faculty
of Science

variables

variables

What to do with the output? - variables

Output of the command can be stored as variable too. They are faster than files, because they are stored in memory.

```
$ Y=5  # set variable Y to 5
$ echo $Y  # print variable Y, must use $
5
$ X="ls -l | wc -l; $Y"; echo $X  # double quotes
ls -l | wc -l; 5
$ X='ls -l | wc -l; $Y'; echo $X  # quotes (apostrophes)
ls -l | wc -l; $Y
$ X=`ls -l | wc -l; $Y`; echo $X  # backquotes (key left from 1)
bash: 5: command not found
4
$ unset Y; echo $Y  # unset variable Y

$ set | less  # list in set of variables
```

Double and single quotes define text (variable substitution is possible only for the first). Back quotes or $() are used for running command inside them.

`declare` variables

In Bash, we can also use `declare`

```
$ declare COST
$ COST=5
$ declare -x COST=5  # equiv for export COST=5, see slide about visibility
```

Variables are stored as strings, we can specifie the type of variable

```
$ declare -i COST=5  # integer type
$ declare -rx COST=5  # read only exported variable
```

Substring of variable

```
$ MYV=tmp/filesID345561for.csv
$ echo ${MYV:9:8}  # name, offset, length
ID345561
```

Task

Small difference can do a lot of troubles. Check difference `X=`wc -l file`` and `X=`cat file | wc -l``.

Variables in practice

Exercise

- create variable MN and use `whoami` to set it
- create variable MR and set as the number of files in /
- create var MRI like MR, but only from files having i in the name
- create var AVAR and set it to ”Hello dear”
- create var BVAR and set it to value of AVAR + MR, and print it
- check differences

```
$ printf "\%s\n" $BVAR; printf "\%s\n" "$BVAR"
$ echo $BVAR; echo "$BVAR"; echo '$BVAR'
```

- if you store command in variable, how to run it? e.g. `cat /proc/cpuinfo`
- for `A=pwd; B='$A'` try to print B, and using B force substitution of its value by `eval` (run command stored in A)

Array variables

Bash supports one dimensional array variable. Subscripts are integers from zero.

```
$ NAMES=(max helen sam zach)
$ echo ${NAMES[0]} # print first element
max
$ echo ${NAMES[*]} # print all elements as one value (for array use @)
max helen sam zach
$ declare -a  A='([0]="xy" [1]=yv [2]=vz)'
$ farm=(dog cat sheep cow)
$ echo ${farm[@]}
dog cat sheep cow
$ X=(*)  # create array from files in the folder
```

You can check the length

```
$ echo ${#A[*]} # length of array
3
$ echo ${#NAMES[1]} # length of element
5
```

Variables defaults and messages

We can check for default values. `a-c` are not set.

Offering default

```
$ echo ${a:-myval}  # if not set use default
myval
$ echo $a; a=new; echo ${a:-myval}

new
```

Set default

```
$ echo ${b:=$(whoami)}  # if not set use default
pvl
$ echo $b; b=new; echo ${b:=$(whoami)}
pvl
new
```

Testing if variable is set

```
$ cd ${c:?Variable is not set at $(date +%Y)}
bash: c: Variable is not set at 2014
```

Variables visibility

Visibility of variables is limited

```
$ XYVAR=7; echo $XYVAR # set and print variable
7
$ env | grep XYVAR # XYVAR is not in env. vars
$ bash # run new shell
$ echo $XYVAR # print varibale

$ exit # exit shell
```

Using environmental variables (available through `env`)

```
$ export XYVAR=6; echo $XYVAR # add varible to ... and print
6
$ env | grep XYVAR # XYVAR is in evn. vars.
XYVAR=6
$ bash
$ echo $XYVAR
6
$
```

Wildcards

If we do not want to the specify exact name, we can use wildcards

- * - no, one or many characters
- ? - one character
- [] - range of characters, e.g. $[a-z]$, $[1-9]$, $[a-d, x-z]$, $[145]$

```
$ ls *file  # list files ending with "file"
linkfile  myfile
$ ls myfil?  # list files having one character after "myfil"
myfile
$ ls fi*  # list files starting fi
fi1  fi1234  fi2  fi3
$ ls fi[1-2]
fi1  fi2
$ ls fi[1-2]*
fi1  fi1234  fi2
$ ls fi[^1-3]  # caret for excluding
fia  fiA  fib
```

Wildcards in practice

Run following commands to generate input for exercise

```
$ mkdir wexer; cd wexer
$ touch max{a..d}; touch max{1..16}
```

Brace expansions {a..d},{one,two,three}, {1..100} generate lists.

Exercise

- create variable mymax, set it to max* and print the exact value
- create array variable and store there all files having at least one digit in the name, print the array and its lenght
- save list of file names having exactly one digit in the file out.txt and variable wout, then print file out.txt and variable wout
- delete files having two digits in the name and count them
- generate 213 files, in form `my1file, my2file, ... my213file` and store their names in file `maxa`
- print content of all files having `a` in the name

String matching

Bash provides string pattern matching operators that can manipulate pathnames and other strings.

- minimal matching prefix #

```
$ MV="34 wild dog, calm sheep, bad cat"
$ echo ${MV#*,}
calm sheep, bad cat
```

- maximal matching prefix ##

```
$ echo ${MV##*,}
bad cat
```

- % and %% is used for minimal and maximal matching suffixes, respectively

```
$ echo ${MV%%,*}
34 wild dog
```

Arithmetic expression

`let` command performs math calculations and expects string

```
$ let "SUM=5+5"; printf "%d" $SUM
10
$ let "SUM=SUM+3"; echo $SUM
13
$ let "SUM++"; echo $SUM
14
$ let "RES=SUM!=14"; echo $RES
0
$ let "RES=SUM!=1"; echo $RES
1
$ let "RES=SUM<20"; echo $RES
1
```

We can also combine `let` with other commands

```
$ let X=`cat ideff.csv | wc -l`; echo $X
15
$ let X=`cat ideff.csv | wc -l`*2; echo $X
30
```

More expansions

Arithmetic expansion by $((*exp*)), without $ you get only status code

```
$ cat my60
#!/bin/bash
echo -n "How old are you? "
read age  # wait for input from user
echo "Wow, you have $((60-age)) years to sixty!"
$ ./my60
How old are you? 10
Wow, you have 50 years to sixty!
```

(()) evaluates arithmetic expr., by $ you get output the value

```
$ x=5 y=8; echo $((2*$x + 10*y))  # $ in brackets is not necessary
90
$ (( w=x*y )); echo $w  # we did not used $ because of setting w
40
$ myvar=$(( $(wc -l < /proc/cpuinfo) - 100 ))
$ echo $myvar
4
$ (( x*y ))  # how you get output from the expr?
```

Exercise on expansions and arithmetics

Exercise

- get data by

```
$ wget http://botanika.prf.jcu.cz/fibich/ideff.csv
```

- create variable containing the last line of the file
- create variable corresponding day of week (1..7); 1 is Monday
- create a variable and set it to 3 * (times) number of lines of the file
- print first half of the file (lines 1 to N/2)
- from the last line of the file, get value of the last column (columns are separated by commas, try string matching)
- create variable for the count of all variables in the current shell (check `set`)

Exercise on expansions and arithmetics

Exercise

- check differences in values of `Z`, `S`, `T` and `R`

```
$ wget http://botanika.prf.jcu.cz/fibich/ideff.csv
$ Z=1+1
$ Z=$(1+1)
$ Z=$((1+1))
$ Z=$[1+1]
$ let S=Z+cat ideff.csv| wc -l
$ let T=Z+`cat ideff.csv | wc -l`
$ let "R=Z+`cat ideff.csv| wc -l`"
```

- how to do more complicated math? try `| bc -l` and set variable to the result of `2/3`
- write command that sets variable to the number of lines of some file minus 1 and divided by 2
- **go through all exercises**
- read next slides before the lesson

script basics

script basics

Short recapitulation

Recapitulation

- quotes and double quotes are used for texts (second one allow variable substitution)

```
$ MVAR="whoami"; echo $MVAR
whoami
```

- back quotes and `$(..)` run commands

```
$ MVAR=$(whoami); echo $MVAR
pvl
```

- `$[..]` and `$((..))` do arithmetic expansions

```
$ MVAR=$((2+2)); echo $MVAR
4
```

- `{1..100}`, `{a..g}`, ... generate lists
- `file[1-9]`, `*.pdf`, ... wildcards can be used for general patterns
- script should have `x` rights to allow running

Script basics

Our first script is the sequence of commands

```
$ cat whoson
#!/bin/bash
date
echo "Currtely logged in users"
who
$
```

#! defines interpreter for script (can be bash, sh, python, R, ...)

```
$ ls -l whoson
-rw-r--r-- 1 pvl pvl 53 Feb  4 10:06 whoson
$ ./whoson
bash: ./whoson: Permission denied
$ chmod u+x whoson # add permissions to run script
$ ./whoson
Tue Feb  4 10:06:56 CET 2014
Currtely logged in users
...
```

Script basics

How to run script (or binary) without path is defined in variable `PATH`

```
$ echo $PATH
/usr/local/bin:/usr/bin:/bin
$ whoson
bash: whoson: command not found
$ export PATH=`pwd`:$PATH # prepend actual directory into PATH
$ whoson
Tue Feb  4 10:06:56 CET 2014
Currtely logged in users
...
$
```

Few commands for searching where command come from

```
$ which ls # locate command
/bin/ls
$ whereis ls # locate binary, source and manuals for command
ls: /bin/ls /usr/share/man/man1/ls.1.gz
$ locate ldd # find files by name
...
```

Script running

Several ways how to run script

```
$ ./script.sh
$ bash script.sh
$ . script.sh # copy script in the current env.
$ source script.sh # . is abbrevation for source
```

Debugging by argument to `bash`

- `-n` *no execution*: checks syntax errors without execution
- `-x` *debuging*: turn debugging mode (remember `set -o xtrace`)
- (advanced) watching single variable by trap

```
#!/bin/bash
declare -i CNT=0
trap ': CNT is now $CNT' DEBUG
while [ $CNT -lt 3 ] ; do
  CNT=CNT+1
done
$ bash -x ./trap.sh
```

Exercise on scripts

Exercise

Run

```
$ mkdir escripts; cd escripts; touch {a..k}files;
```

- write command or script that prints how many times it was executed
- store the names of all files in the file one per line and all in one line
- write script that count all files in the current folder and store value in the variable `MFI` (use export to make variable visible)
- improve the script that it stores actual date and the count at the end of file `mfi.log` (both at one line), check it be re-running
- generate some new files (e.g. by `1..20news`) and improve the script to print message about the difference from the previous value of `MFI` and chek if it works
- try to debug your script (e.g. by -x option to bash)

control structures

if then else fi

To control flow of commands, we can use if *test-command* then..else.. structure

```
$ cat mytest
echo -n "Write a: "; read a
echo -n "Write b: "; read b
if test $a == $b; then  # check man pages of test
  echo "Match!"
  else
  echo "Do not match!"
fi
$ ./mytest
Write a: a
Write b: a
Match!
$
```

`else` part is not necessary, and we can also add `elif` part with *test-command*

if then else

Check the number of script arguments ($#)

```
$ cat cat chkargs
if test $# -eq 0; then
  echo "Supply arguments!"
  exit 1;
else
  echo "First argument of $0 is $1"
fi
$ ./chkargs
Supply arguments!
$ ./chkargs hello
First argument of ./chkargs is hello
```

Warnings

Always check arguments!

`test` command

Many options to check files, their permissions, arithmetic, ...

```
$ ls -l mytest*
-rwxr--r-- 1 pvl pvl 131 Feb  4 15:05 mytest
$ test -e mytest; echo $? # test of file existence
0
$ test -e mytestNONE; echo $? # non existing file
1
$ test ! -d mytest; echo $? # test for NON directory, ! for negation
0
# combined conditions -a for AND and -o for OR
$ test -e mytest -a -d mytest; echo $? # exists and is directory
1
$ test -e mytest -o -d mytest; echo $? # exists or is directory
0
# square brackets do the test command, check man test
$ if [ -e mytest ]; then echo "exists"; fi
exists
$ if [ 5 -gt 4 ]; then echo "definitely"; fi
definitely
```

`test` command with wildcards

For pattern matching and strings we can use [[ ]]

```
$ COMP="Faculty of Science"
$ if [[ $COMP = F* ]]; then echo "Start by F"; fi
Start by F
$ if [[ $COMP = [ABC]* ]]; then echo "Start by A, B, or C
    "; fi
$ if [[ $COMP = +(F)*Science ]]; then echo "More special"
    ; fi
More special
$
```

+ is used for one or more characters (recall * is for zero or more). You can use also [:alpha:], [:digit:], [:lower:], ..

```
$ if [[ $COMP = [[:alpha:]]* ]]; then echo "Contains only
    alphabetics"; fi
Contains only alphabetic
```

Conditions

To combine conditions you can you also notation

```
$ if [[ 30 -gt $age && $age < 60 ]]; then ..
$ if ((30 < age && age < 60)); then ..
```

Exercise

- write `if` command that checks that actual month is March
- write `if` command that checks if the last command was successful (remember `$?`)
- write `if` command that checks if value $VAR contains value $IN
- write script that cheks if in the current folder is more files than number you give as the first argument of the script
- try to debug your script (e.g. by -x option to bash)
- read further lesson's slides