Master Calcul Scientifique
Shell

1 Shell parameters

1.1 A few definitions

Definition 1. *A* parameter *of a shell is either a number, a special caracter (see below) or a name (sequence of alphanumeric characters that is not a number or a special character)*
A variable *of a shell is a parameter corresponding to a name*
A position parameter is a parameter that is not special and not a variable

We say that a parameter is allocated is it has a *value* (null is a value). To disacollate a variable, the only way is through the command unset.

1.2 Special and positions parameters

- `0` : current command name;
- `#` : number of position parameters;
- `*`, `@` : all the position parameters;
- `1` to `9` : the 9 first position parameters;
- `x` : the position parameter $x(> 9)$;
- `$` : the pid (process identifier) of the current command;
- `_` : the last used parameter;
- `-` : the flags (options) of the current command;
- `?` : the *exit-status* of the last command runned.
  - everything ok: `?` = 0,
  - something anormal: `?` $\neq 0$;
- `$!` : the pid of the last run process in background

The `shift` command shifts the numbered parameters (`1` is lost and `#` is updated)

1.3 Variables

A variable is defined as soon as it is affected: `$ FOO="Hello world"`
`echo` prints its given argument:

```
$ echo FOO
FOO
```

To evaluate a variable, one adds $ before its name:

```
$ echo $FOO
Hello world
```

In a shell, **everything is a character chain**.
Each command is a chain evalated by the shell. There are three delimiters:

- quotes ' ' disable evaluation;
- quotation mark " " make a character chain after evaluation of what is inside;
- backquotes ` ` make a chain evaluated as a command.

```
$ echo '$FOO'
$FOO
$ echo "echo '$FOO'"
echo 'Hello world'
$ BAR="anything you want" ; echo $BAR
anything you want
$ BAR=`anything you want`
anything: Command not found.
```

Around variable names

- `${parameter%regexp}` removes the shortest suffix defined by regexp in the evaluation of parameter,
- `${parameter%%regexp}` removes the longest suffix defined by regexp in the evaluation of parameter,
- `${parameter##regexp}` removes the shortest prefix defined by regexp in the evaluation of parameter,
- `${parameter##regexp}` removes the longest prefix defined by regexp in the evaluation of parameter

Example 1.

```
$ FOO=babarerre.tar.gz
$ echo ${FOO%ba*}
ba
$ echo ${FOO%%ba*}

$ echo ${FOO%%re*}
baba
$ echo ${FOO%re*}
babarer
$ echo ${FOO%%ba}
babarerre.tar.gz
```

```
$ echo ${FOO#ba}
barerre.tar.gz
$ echo ${FOO##ba}
barerre.tar.gz
$ echo ${FOO##ba*}

$ echo ${FOO%.gz}
babarerre.tar
$ echo ${FOO#*.}
tar.gz
$ echo ${FOO##*.}
gz
```

1.4 Some control structures

- Simple condition:

```
if instruction-test
  then instruction
elif other_instruction-test
  then other_instruction
else last_instruction
fi
```

If `instruction-test` has an exit status equal to 0, then `instruction` is excuted, otherwise, if `other-instruction-test` has an exit status equal to 0, then `other-instruction` is executed. Otherwise, `last-instruction` is executed.

- Multiple conditions: the function `case`

```
case expr in
  regular_expression_1) instructions;;
  regular_expression_n) instructions;;
esac
```

Once a condition has been satisfied, the other conditions are not executed (this is diffenrent in C).

- Numerative iteration:

```
For var in expr do
  instruction
done
```

  - If `expr` is empty, the instruction is not executed,
  - `in expr` can be omitted ; by default, this will be `in "$@"`.

- Conditionnal iteration

```
while instruction-test do
  instruction
done
```

- Inverse conditionnal iteration

```
until instruction-test do
  instruction
done
```

2 The less command

It is used to display some text. In particular, it is what the `man` function uses to display its help. There are usefull commands you can use while viewing a document via `less`:

- `h` will display the available commands,
- `q` will quit the viewer,
- `/` followed by an expression will search the next apparition of this expression in the document, move to the first results, and highlight the others,
- `n` moves to the next apparition of the searched expression,
- `N` moves to the previous apparition of the searched expression,
- `?` do the same as `/` but the search is made forward (therefore `n` and `N` are somehow reversed),

3 Few things about scripts

3.1 First definitions

One can put shell commands in a file `foo`; then, one can eihter

- Interpret the file with the current shell via `. foo`;
- Makes the file executable (`\$ chmod u+x foo`) and use it directly ; that will be interpreted in another shell.

Definition 2. *A script is a file containing shell commands. If one want it to be executable, we put in the first line which interpret we want to use (for instance,* `#!/bin/sh`*)*

Remark : `/bin/sh` is just a link to the shell interpreter you are using. It can be for instance `bash`, `dash` or `csh`. You can see which version you are using via the command `$ ls -l /bin/sh`
Remark : Of course, a script can aslo be used by the function call `sh -[opt] name_of_the_script`
The position parameters in a script are the arguments of the line using it.
Going to a new line in the file terminates the command. For instance, the command

```
if [ $# -ne 2 ] ; then echo "pb" ; fi
```

is the same thing than writing

```
if [ $# -ne 2 ]
then echo "pb"
fi
```

3.2 One example

```
# This is a comment
#!/bin/bash
NBPARAM=2 # number of parameters of the script

usage()
# How to use this script
{
    echo "Usage: `basename $0` firstparam secondparam"
    echo "Print \"'firstparam' and 'secondparam'\"".
    return 0 # exit status of the function
}

# We test the number of parameters and use
# the usage() function if it is different from 2
if [ $# -ne $NBPARAM ]
then
    usage # the function call does not use ()
    exit 1
fi

# We ouput the parameters
echo $1 and $2
# and everything is fine
exit 0
```

A function follows the following syntax :

```
name_of_the_function()
instructions
redirection of an exit status
```