Bash tutorial

Anthony Scemama <scemama@irsamc.ups-tlse.fr>

Labratoire de Chimie et Physique Quantiques
IRSAMC (Toulouse)

Introduction

Slides:

http://irpf90.ups-tlse.fr/files/bash.pdf

- The shell lets you interact with the OS
- Many available shells: ksh, pdksh, bash, zsh, csh, tcsh, ...
- It is a command interpreter : each line is interpreted and executed immediately
- A few commands are specific to the shell (cd, kill,...)
- Most Linux commands are programs (ls, grep, echo,...) located in `/bin` or `/usr/bin`
- Such commands are independent of the shell

Unix philosophy

Mike Gancarz: Unix Philosophy

1. Small is beautiful.
2. Make each program do one thing well.
3. Build a prototype as soon as possible.
4. Choose portability over efficiency.
5. Store data in flat text files.
6. Use software leverage to your advantage.
7. Use shell scripts to increase leverage and portability.
8. Avoid captive user interfaces.
9. Make every program a filter.

(see Wikipedia: Unix Philosophy)

Outline

1. Interactive Bash
2. Useful Linux commands
3. Writing scripts
4. Examples

Interactive Bash

Bash execution

- When bash starts, if it is a login shell it executes the `/etc/profile` file followed by the `.bash_profile` file located in the user's home directory
- The the `.bashrc` file is executed (even for non-login shells)
- When a login shell exits, the `.bash_logout` file is executed

Executing commands

```
$ program_name program_arguments
```

- Bash first creates a fork : the current bash process creates a copy of itself and both processes execute concurrently (in the same process group)
- The original bash process waits for an interruption of the new process (background)
- The new bash process transforms itself (exec) into the called program (foreground)
- `Ctrl-C` sends a `TERM` signal to the process in the foreground, asking the process to terminate
- `Ctrl-Z` stops the running process, and the background process comes to the foreground

Executing commands

```
$ program_name program_arguments &
```

starts the new program in the background

```
$ jobs
[1]+  Running     program_name program_arguments &
```

Job number 1 is running in the background

```
$ fg %1
```

Bring job number 1 in the foreground

```
[Ctrl-Z]
$ bg
```

Stop job number 1, return to the shell and resume it in the background

Exit codes

Every process terminates with an *exit code*. The exit code is 0 if the process ends normally. Otherwise, the exit code can be used to understand the abnormal termination.

```
$ ls toto && echo OK
toto
OK
$ ls titi && echo OK
ls: cannot access titi: No such file or directory
```

Runs the command `ls toto`. If the exit code is zero, run `echo OK`. `&&` is interpreted as *and*.

Exit codes

```
$ ls toto || echo Failed
toto
$ ls titi || echo Failed
ls: cannot access titi: No such file or directory
Failed
```

If the exit code is not zero, run `echo Failed`. `||` is interpreted as *or*.

```
$ ls toto && echo OK || echo Failed
toto
OK
$ ls titi && echo OK || echo Failed
ls: cannot access titi: No such file or directory
Failed
```

Pattern Matching

- `*` : Matches any string
- `?` : Matches any character@
- `[a-f]` : Matches any of a,b,c,d,e,f
- `[4-8]` : Matches any of 4,5,6,7,8
- `[G-I]` : Matches any of G,H,I
- `[azerty]` : Matches any of a,z,e,r,t,y
- `[^azerty]` : Matches everything except a,z,e,r,t,y

```
$ ls *.tex
bash.tex
$ ls bash.???
bash.tex  bash.aux  bash.dvi  bash.pdf
$ ls bash.*[^x]
bash.dvi  bash.pdf  bash.text
```

```
$ ls bash.[a-p]*
bash.aux  bash.pdf
```

Variables

Variables are set by assigning them:

```
$ TMPDIR=/tmp
$ EMPTY=
```

Variables can be unset:

```
$ unset TMPDIR
```

The value of a variable is obtained by the $ operator:

```
$ echo TMPDIR $TMPDIR
TMPDIR /tmp
```

New values can be appended to a variable:

```
$ TMPDIR+=/my_tmp ; echo $TMPDIR
/tmp/my_tmp
```

Environment variables

Wikipedia:

Environment variables are a set of dynamic named values that can affect the way running processes will behave on a computer.

- `HOME` : Home directory
- `SHELL` : Name of the current shell
- `USER` : Current user name
- `PATH` : List of directories where to search for executables
- `LD_LIBRARY_PATH` : List of directories where to search for shared libraries

Variables can be declared using the `declare` keyword

```
declare [-aAfFgilrtux] [-p] [name[=value] ...]
```

- `-a` : Indexed array variable
- `-A` : Associative array variable

- `-i` : The variable is treated as an integer
- `-l` : All upper-case characters are converted to lower-case
- `-u` : All lower-case characters are converted to upper-case
- `-r` : Read-only
- `-x` : Export to the environment

```
$ declare -i int_var=1
$ text_var=1
$ text_var+=2  ;  int_var+=2
$ echo $text_var $int_var
12 3
$ declare -u upper_var="This is an Upper Case example"
$ echo $upper_var
THIS IS AN UPPER CASE EXAMPLE
$ declare -r read_only_var="Unchangeable"
$ read_only_var="Modified"
bash: read_only_var: readonly variable
```

Special variables

- `*`, `@`, `#`, `-`, `0` : (see scripts section)
- `?` : Exit status of the most recently executed process,
- `$` : Process ID of the shell
- `!` : Process ID of the most recently executed background command.
- `_` : Last argument to the previous command
- `RANDOM` : Returns a integer uniform random number between 0 and 32767

```
$ ./a.out &
[1] 20941
$ echo $!
20941
$ echo $$
20717
$ ls titi
ls: cannot access titi: No such file or directory
```

```
$ echo $_
titi
$ echo $?
0
$ ls titi
ls: cannot access titi: No such file or directory
$ echo $?
2
```

Arrays

- One-dimensional indexed arrays are referenced using integers (zero-based)
  - declared with `declare -a`
  - affected like: `A[3]=three`
  - used like: `${A[3]}` The braces around `A[3]` shows on what the `$` operator acts.
- If the subscript is less than zero, it is used as an offset from the end
- Associative arrays are referenced using integers
  - declared with `declare -A`
  - affected as: `A[name]=Anthony`
  - used as: `${A[name]}`

Indexed arrays can be affected in a compound statement:

```
$ A=(elem1 elem2 elem3)
$ echo ${A[1]}
elem2
```

For associative arrays,

```
$ A=([first]=elem1 [second]=elem2 [third]=elem3)
$ echo ${A[second]}
elem2
```

All array values can be obtained using `{A[@]}`

```
$ A=(elem1 elem2 elem3)
$ echo ${A[@]}
elem1 elem2 elem3
```

All array keys can be obtained using `{!A[@]}`

```
$ echo ${!A[@]}
0 1 2
```

Input/Output redirection

```
$ command < input_file
```

The standard input of `command` (file descriptor 0) is redirected to file `input_file`

```
$ command > output_file
```

The standard output of `command` (file descriptor 1) is redirected to file `output_file`

```
$ command >> output_file
```

The standard output of `command` (file descriptor 1) is appended to file `output_file`

```
$ command 2> error_file
```

The standard error of `command` (file descriptor 2) is redirected to file `error_file`

```
$ command < input > output_file 2> error_file
```

Multiple outputs can be merged :

```
$ command  > output_file 2>&1
```

The output of `command` is redirected to `output_file`, and then the standard error is redirected to the standard output. This can be re-written as

```
$ command  &> output_file
```

Example using file descriptors

```
$ echo 1234567890 > File  # Write string to "File"
$ exec 3<> File           # Open "File" with fd 3
$ read -n 4 <&3           # Read only 4 chars
$ echo -n . >&3           # Write a decimal point
$ exec 3>&-               # Close fd 3
```

```
$ cat File
1234.67890
```

Pipes

```
$ command_1 | command_2
```

Redirects standard output of `command_1` to standard input of `command_2`.

Pipes are essential:

Make every program a filter

Named pipes

Pipes can be created and used as files.

`mkfifo` creates a *named pipe* on the file system.

```
$ mkfifo my_pipe
$ ls -l
prw-rw-r-- 1 scemama scemama 0 Apr  2 23:20 mypipe
$ ls > my_pipe &
$ # Do some stuff...
$ cat my_pipe
prw-rw-r-- 1 scemama scemama 0 Apr  2 23:20 mypipe
[1]+  Done                    ls > mypipe
$ rm my_pipe
```

The named pipe has to be removed when finished.

Here documents

```
$ cat << EOF
> This is my
> input file
> in an interactive shell
> EOF
This is my
input file
in an interactive shell
```

Read input until a line containing only `EOF` is seen. All of the lines read up to that point are then used as the standard input for a command.

Brace expansion

Braces are used to unambiguously identify variables:

```
$ VARIABLE=abcdef
$ echo Variable: $VARIABLE
Variable: abcdef
$ echo Variable: $VARIABLE123456
Variable:
$ echo Variable: ${VARIABLE}123456
Variable: abcdef123456
```

But also for more exciting things

```
$ echo a{d,c,b}e
ade ace abe
$ ls {test1,test2}.{f90,o}
test1.f90  test1.o  test2.f90  test2.o
```

Using integers separated by `..`

```
$ echo test{6..8}
test6  test7  test8
$ echo test{2..10..3}
test2 test5 test8
$ echo test{8..6}
test8  test7  test6
```

Tilde expansion

~ : home directory of the current user ($HOME)

```
$ echo ~
/home/scemama
$ echo ~user
/home/user
```

~+ : Absolute path of current directory ($PWD) ~- : Absolute path of previous directory ($OLDPWD)

Command expansion

`$(command)` or `` `command` `` : Substitute by the output of the command

```
$ CURRENT_DATE=$(date)
$ echo $CURRENT_DATE
Mon Apr  1 23:29:34 CEST 2013
```

Parameter expansion

- `${parameter:-word}` : If parameter is unset word is substituted. Otherwise, the value of parameter is substituted.
- `${parameter:=word}` : Assign Default Values.
- `${parameter:?word}` : Display Error if Null or Unset.
- `${parameter:+word}` : Use Alternate Value.
- `${parameter:offset}` : Substring starting at the `offset` character
- `${parameter:offset:length}` : Substring starting at the `offset` character up to `length` characters

```
$ B=${A:?Error message}
bash: A: Error message
$ B=${A:-First}  ; echo $B
First
$ B=${A:=Second}  ; echo $B
Second
```

```
$ B=${A:=Third}  ; echo $B
Second
$ B=${C:+Fourth} ; echo $B

$ B=${A:+Fourth} ; echo $B
Fourth
$ echo ${B:3}
rth
```

- `${#parameter}` : Parameter length
- `${parameter#word}` : Remove matching prefix pattern
- `${parameter%word}` : Remove matching suffix pattern
- `${parameter/pattern/string}` : Pattern substitution
- `${parameter^^}` : Convert to upper case
- `${parameter,,}` : Convert to lower case

```
$ A="This is my test string"
$ echo ${#A}
22
$ echo ${A#This is}
my test string
$ echo ${A%test string}
This is my
$ echo ${A/my/your}
This is your test string
$ echo ${A^^}
THIS IS MY TEST STRING
```

Arithmetic expansion/evaluation

Syntax : `$((expression))` The result is substituted by the result of the expression.

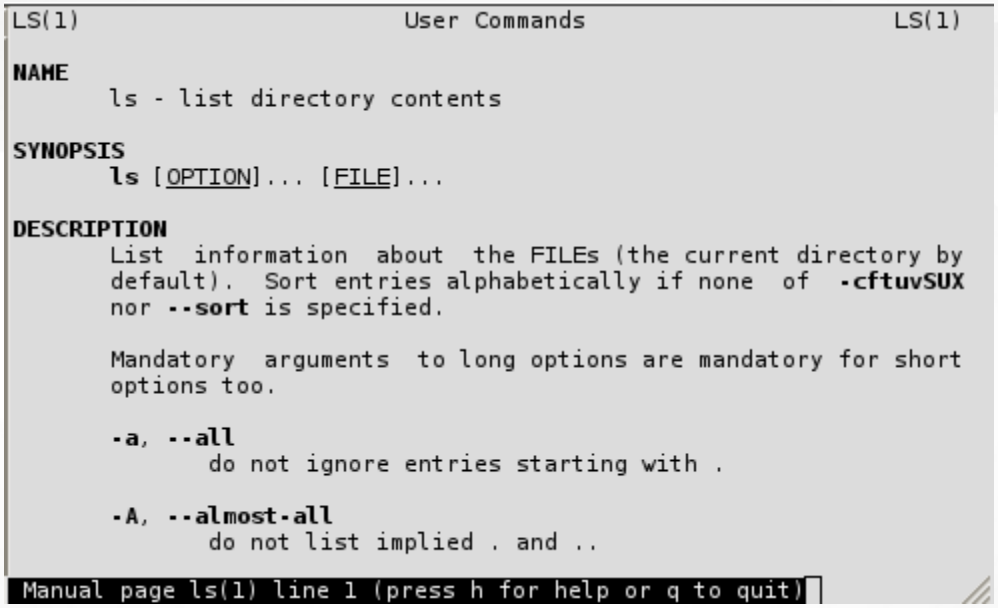
The `let` keyword evaluates arithmetic expressions:

```
$ A=$((3+5))  ; echo $A
8
$ let A++  ; echo $A
9
$ A=$((1<<6))  ; echo $A  # Bit shift
64
```

Useful Linux commands

man

The most important linux command is `man`



`man command` : displays the manual page of `command`.

For all commands in this section, you are encouraged to look at the man pages (including `man bash`, or `man uranus`).

seq

`seq a b c` : Prints a sequence of numbers between a and c using step b

```
$ seq 5 3 12
5
8
11
$ seq 4
1
2
3
4
```

cat

Concatenate files and print on the standard output

```
$ cat File1
content of first file
$ cat File2
the second file
$ cat File1 File2
content of first file
the second file
```

`zcat` is the same as cat for gzipped files.

tac

Same as `cat`, but with lines reversed (last line first)

```
$ tac << EOF
> First line
> Second line
> Third line
> EOF
Third line
Second line
First line
```

rev

Prints lines in reversed order

```
$ rev << EOF
> First line
> Second line
> Third line
> EOF
enil tsriF
enil dnoceS
enil drihT
```

date

Print or set the system date and time

```
$ date
Wed Apr  3 00:32:29 CEST 2013
$ date --date="Next Monday"
Mon Apr  8 00:00:00 CEST 2013
$ date -r mypipe
Tue Apr  2 23:23:35 CEST 2013
```

`date -r` displays the last modification time of a file.

touch

Changes file timestamps, and creates a file if it doesn't exist.

```
$ date ; touch hello
Wed Apr  3 00:36:08 CEST 2013
$ date -r hello
Wed Apr  3 00:36:08 CEST 2013
$ touch hello
$ date -r hello
Wed Apr  3 00:36:27 CEST 2013
$ touch -d "25 December 2000" hello
$ date -r hello
Mon Dec 25 00:00:00 CET 2000
$ ls -l
total 0
-rw-rw-r-- 1 scemama scemama 0 Dec 25  2000 hello
```

pwd

Print current working directory

```
$ pwd
/home/scemama/CurDir
```

mkdir

Creates diretories. The `-p` option creates the parents as needed

```
$ mkdir /tmp/gdr
$ mkdir /tmp/gdr/test/newdirectory
mkdir: cannot create directory `/tmp/gdr/test/newdirectory':
No such file or directory
$ mkdir -p /tmp/gdr/test/newdirectory
$ ls /tmp/gdr/test/
newdirectory
```

yes

Repeatedly output a line with "y"

```
$ touch xx{1..10}
$ ls
xx1  xx10  xx2  xx3  xx4  xx5  xx6  xx7  xx8  xx9
$ rm -i *
rm: remove regular empty file `xx1'? ^C
$ yes | rm -i *
rm: remove regular empty file `xx1'? rm: remove regular
empty file `xx10'? rm: remove regular empty fi [...]
regular empty file `xx8'? rm: remove regular empty file `xx9'?
$ ls
$
```

df

Print the usage of mounted file systems

```
$ df
Filesystem     1K-blocks     Used Available Use% Mounted on
/dev/sda6       12265896  4287552   7348600  37% /
udev              492300        4    492296   1% /dev
tmpfs             201560      424    201136   1% /run
none                5120        0      5120   0% /run/lock
/dev/sda8      148904124 75729884  73174240  51% /home
$ df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda6        12G  4.1G  7.1G  37% /
udev            481M  4.0K  481M   1% /dev
tmpfs           197M  424K  197M   1% /run
none            5.0M     0  5.0M   0% /run/lock
/dev/sda8       143G   73G   70G  51% /home
```

du

Estimate file space usage

```
$ du GDRCorrel/
40      GDRCorrel/Makefiles/Test
168     GDRCorrel/Makefiles
4       GDRCorrel/Bash/test
368     GDRCorrel/Bash
540     GDRCorrel
$ du -h GDRCorrel/
40K     GDRCorrel/Makefiles/Test
168K    GDRCorrel/Makefiles
4,0K    GDRCorrel/Bash/test
368K    GDRCorrel/Bash
540K    GDRCorrel/
```

uptime

Tell how long the system has been running.

```
$ uptime
23:58:45 up 43 days, 11:17, 45 users,
 load average: 2.61, 2.28, 2.03
```

who

Show who is logged on

```
$ who
root     tty3         2013-03-07 18:16
root     tty2         2013-03-07 17:45
boggio   pts/3        2013-04-03 09:52 (lpqpc6.ups-tlse.fr)
morin    pts/6        2013-04-02 15:53 (lpqlx146.ups-tlse.fr)
boggio   pts/7        2013-04-03 14:52 (lpqpc6.ups-tlse.fr)
scemama  pts/16       2013-04-03 15:40 (lpqlx139.ups-tlse.fr)
$ who -b
     system boot  2013-03-01 16:13
$ who -q
root root morin iftner morin boggio vmaire
morin boggio audesimon morin trinquier audesimon
iftner audesimon beangoben marsden scemama
# users=18
```

w

Show who is logged on and what they are doing

```
$ w -s
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU WHAT
root     tty2     -                 07Mar13 26days  0.05s  0.05s -bash
morin    pts/0    lpqlx146.ups-tls 21Mar13  2.00s  2.43s  1.09s vim ../source/md/mdpt.f
iftner   pts/1    lcpqpc153.ups-tl 20Mar13 11:44   3.48s  3.48s -bash
morin    pts/2    lpqlx146.ups-tls 09:48    3:18   0.51s  0.51s -bash
boggio   pts/3    lpqpc6.ups-tlse. 09:52   52:19   0.22s  0.07s vim p44_cas10_631gd_S0TS2_opt.com
vmaire   pts/5    lpqlx126.ups-tls 22Mar13  9days  0.07s  0.07s -bash
morin    pts/6    lpqlx146.ups-tls Tue15    1:14m  3.16s  0.10s vim premiercode.f90
boggio   pts/7    lpqpc6.ups-tlse. 14:52   24:56   0.05s  0.05s -bash
audesimo pts/8    ir-kd153.ups-tls 10:00    1:09   1.07s  0.84s molden deMon.mol
morin    pts/9    lpqlx146.ups-tls 26Mar13  3:03m  1.69s  1.69s -bash
```

bc

Arbitrary precision calculator language. The `-l` option defines the standard math library.

```
$ bc -l
bc 1.06.95
Copyright 1991-1994, 1997, 1998, 2000, 2004, 2006 Free Software Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type `warranty'.
2.+3.
5.
$ echo 5./3. | bc -l
1.66666666666666666666
```

A trick to be able to use floating point operations in bash

basename

Strip directory and suffix from filenames

```
$ basename /usr/bin/sort
sort
$ basename my_picture.jpg .jpg
my_picture
$ FILE=$(basename my_picture.jpg .jpg)
$ mv $FILE.jpg $FILE.png
```

nohup

Run a command immune to hangups. If you kill the bash session, the program will continue to run.

```
$ nohup ./a.out &
nohup: ignoring input and appending output to `nohup.out'
$ exit
```

wc

Print newline, word, and byte counts for each file

```
$ wc bash.tex
  94  156 1832 bash.tex
$ wc -l bash.tex
94 bash.tex
$ wc -w bash.tex
156 bash.tex
$ wc -l < bash.tex
94
$ cat bash.tex | wc -w
156
```

head / tail

Output the first part of files (head) or the end of files (tail)

```
$ seq 10 | head -3
1
2
3
$ seq 10 | tail -3
8
9
10
$ seq 10 | head -7 | tail -3
5
6
7
```

grep

Print lines matching a pattern

```
$ grep "ENERGY =" 3.8.CAS.out
 ----- FROZEN CORE ENERGY =      -182.7238608120
 STATE #    1  ENERGY =    -198.806582658
 STATE #    1  ENERGY =    -198.806584871
     ONE ELECTRON ENERGY =    -301.0460998455
     TWO ELECTRON ENERGY =      90.9596841354
$ grep -m 1 "ENERGY =" 3.8.CAS.out
 ----- FROZEN CORE ENERGY =      -182.7238608120
$ grep "energy =" 3.8.CAS.out
$ grep -m 1 -i "energy =" 3.8.CAS.out
 ----- FROZEN CORE ENERGY =      -182.7238608120
```

cut

Remove sections from each line of files

```
$ grep -m 1 "ENERGY =" 3.8.CAS.out
 ----- FROZEN CORE ENERGY =      -182.7238608120
$
$ grep -m 1 "ENERGY =" 3.8.CAS.out | cut -d "=" -f 2
      -182.7238608120
```

- `-d` : delimiter
- `-f` : field

paste

Merge lines of files

```
$ seq 4 > f1 ; seq 10 14 > f2
$ paste f1 f2
1       10
2       11
3       12
4       13
        14
$ paste -s f1 f2
1       2       3       4
10      11      12      13      14
$ paste -s -d 'x' f1 f2
1x2x3x4
10x11x12x13x14
```

If the delimiter is set to \n, zip the lines of the 2 input files.

column

Format into multiple columns:

```
$ cat << EOF | column -t -s, -n
> name,number,city
> Mr. A,1,Toulouse
> Miss Bee,23,Perpignan
> Sarah Connor,8,Mexico
> Harry Cover,,London
> EOF
name          number  city
Mr. A         1       Toulouse
Miss Bee      23      Perpignan
Sarah Connor  8       Mexico
Harry Cover           London
```

- `-t` : create a table

- -s : delimiter

at

Execute a job at a given time

```
$ at 23:59
warning: commands will be executed using (in order)
a) $SHELL b) login shell c) /bin/sh
at> /usr/bin/do_my_backup
job 103 at Wed Apr  3 23:59:00 2013
$ atq
103     Wed Apr  3 23:59:00 2013 a scemama
```

mail

Send an email

```
$ mail scemama@irsamc.ups-tlse.fr -s "Hello" < email_file
$ cat email_file | mail scemama@irsamc.ups-tlse.fr -s "Hello"
```

tee

Read from standard input and write to standard output and files

```
$ ./hello_world.sh | tee output
Hello World !
$ cat output
Hello World !
```

Sort

Sort lines of text files

```
$ echo $RANDOM > f1 ; echo $RANDOM >> f1
$ echo $RANDOM >> f1
$ cat f1
204
26828
11760
$ sort f1
11760
204
26828
$ sort -n f1
204
11760
26828
```

uniq

Report or omit repeated lines

```
$ seq 2 > f1 ; tac f1 > f2
$ cat f1 f2  | tee f3
1
2
2
1
$ uniq f3
1
2
1
$ sort f3 | uniq
1
2
```

split

Split a file into pieces

```
$ ls -sh
total 100M
100M BigFile
$ split -b 30M BigFile SmallFile.
$ ls -sh
total 200M
100M BigFile       30M SmallFile.aa
 30M SmallFile.ab  30M SmallFile.ac
 10M SmallFile.ad
```

diff

Compare files line by line

```
$ seq 10 > f1 ; seq 3 11 > f2
$ diff f1 f2
1,2d0
< 1
< 2
10a9
> 11
```

sleep

delay for a specified number of seconds

```
$ sleep 10
```

true / false

Return exit status 0 for `true` and 1 for `false`

```
$ true  && echo TRUE || echo FALSE
TRUE
$ false && echo TRUE || echo FALSE
FALSE
```

tr

Translate or delete characters

```
$ echo 'linux' | tr "[:lower:]" "[:upper:]"
LINUX
$ echo 'LINUX' | tr -d "IU"
LNX
$ echo 'LINUX' | tr -d "LINU" "UNI."
UNI.X
```

wait

Wait until the process finishes

```
$ sleep 10 &
[1] 15297
$ wait 15297
[1]+  Done                    sleep 10
$ sleep 5 & sleep 10 & wait
[1] 15381
[2] 15382
[1]-  Done                    sleep 5
[2]+  Done                    sleep 10
```

taskset

Set a process's CPU affinity

```
$ taskset -c 1-3 ./a.out
```

`a.out` will run only on CPU cores 1, 2 and 3.

Use to avoid process migration and improve performance of HPC applications

join

Joins the data fields of two files.

```
$ cat f1
Adams A.    555-6235
Erwin G.    555-1234
Lewis B.    555-3237
Norwood M.  555-5341
Wright M.   555-1234
Xandy G.    555-5015
$ cat f2
Erwin       Dept. 389
Nicholson   Dept. 311
Norwood     Dept. 454
Wright      Dept. 520
Xandy       Dept. 999
$ join f1 f2
```

```
Erwin G. 555-1234 Dept. 389
Norwood M. 555-5341 Dept. 454
Wright M. 555-1234 Dept. 520
Xandy G. 555-5015 Dept. 999
```

time

Run programs and summarize system resource usage

```
$ time gzip BigFile
real    0m4.176s
user    0m3.988s
sys     0m0.156s
$ /usr/bin/time gzip BigFile
4.26user 0.21system 0:04.51elapsed
99%CPU (0avgtext+0avgdata 4000maxresident)k
0inputs+200outputs (0major+302minor)pagefaults
0swaps
```

wdiff

Display word differences between text files

```
$ wdiff f1 f2
Dickerson B.    555-1842
[-Erwin-]
G. {+Erwin+}        555-1234
Jackson J.      555-0256
[-Lewis B.        555-3237-]
Norwood M.      555-5341
Smartt D.       555-1540
{+Scemama A.      555-3237+}
Wright M.       555-1234
```

fold

Wrap each input line to fit in specified width

```
$ echo wrap each input line to fit in \
  specified width | fold -w 12
wrap each in
put line to
fit in speci
fied width
$ echo wrap each input line to fit \
  in specified width | fold -s -w 12
wrap each
input line
to fit in
specified
width
```

xargs

Build and execute command lines from standard input

```
$ ls
$ cut -d: -f1 < /etc/passwd | sort | xargs touch
$ ls
backup   lp           scemama
bin      mail         sync
daemon   man          sys
games    messagebus   syslog
gnats    news         usbmux
irc      nobody       uucp
libuuid  proxy        www-data
list     root
```

wget

Download files from the network

```
$ wget "http://www.netlib.org/lapack/lapack.tgz"
--2013-04-03 23:46:32--  http://www.netlib.org/lapack/lapack.tgz
Resolving www.netlib.org (www.netlib.org)... 160.36.131.121
Connecting to www.netlib.org (www.netlib.org)|160.36.131.121|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 6168281 (5.9M) [application/x-gzip]
Saving to: `lapack.tgz'

100%[==============================>] 6,168,281    544K/s   in 12s

2013-04-03 23:46:44 (513 KB/s) - `lapack.tgz' saved [6168281/6168281]
$ ls
lapack.tgz
```

convert

Convert between image formats.

```
$ convert image.jpg image.gif
$ convert image.jpg image.pdf
```

Writing scripts

Hello world

File: `hello_world.sh`

```
#!/bin/bash
echo Hello world
```

Make the file executable:

```
$ chmod +x hello_world.sh
$ ./hello_world.sh
Hello world
```

First line: Path to the interpreter of the script

Running a script

Run a script in a new process:

```
$ ./hello_world.sh
Hello world
```

Or execute in the current shell (include)

```
$ . ./hello_world.sh
Hello world
$ source ./hello_world.sh
Hello world
```

Tests

```
if [[ expression ]]
then
    commands
elif [[ expression2 ]]
then
    commands
else
    commands
fi
```

Test expressions

- `[[ expression ]]` : Test operator
- `! expression` : Not operator
- `-n STRING` : Non-zero string length
- `-z STRING` : Zero string length
- `STRING1 = STRING2` : Two strings are equal
- `STRING1 != STRING2` : Two strings are not equal
- `INT1 -eq INT2` : Two integers are equal
- `INT1 -ne INT2` : Two integers are not equal
- `INT1 -ge INT2` : Greater or equal
- `INT1 -gt INT2` : Greater than
- `INT1 -le INT2` : Less or equal
- `INT1 -lt INT2` : Less than
- `-e FILE` : FILE exists

- `-f FILE` : FILE is a regular file
- `-d FILE` : FILE is a directory
- `-p FILE` : FILE is a named pipe
- `-r FILE` : FILE has read permissions
- `-w FILE` : FILE has write permissions
- `-x FILE` : FILE has execute permissions
- `-s FILE` : FILE has a size >0
- `FILE1 -nt FILE2` : FILE1 is newer than FILE2
- `FILE1 -ot FILE2` : FILE1 is older than FILE2

```
if [[ -z $TMPDIR ]] ; then
  export TMPDIR=/tmp/$USER
  if [[ ! -e $TMPDIR ]] ; then
    mkdir -p $TMPDIR
  elif [[ ! -d $TMPDIR ]] ; then
    echo "Unable to create TMPDIR"
  elif [[ ! -r $TMPDIR ]] || \
       [[ ! -x $TMPDIR ]] || \
       [[ ! -w $TMPDIR ]] ; then
      echo "TMPDIR: incorrect permissions"
  fi
fi
```

Case

```
case STRING in
  str1)
     commands
     ;;
  str2)
     commands
     ;;
  *)
     commands
     ;;
esac
```

```
case $COLORTERM in
  gnome-terminal)
     echo Gnome Terminal
     ;;
  xterm)
     echo Xterm
     ;;
  rxvt)
     echo rxvt
     ;;
  *)
     echo Unknown terminal
     ;;
esac
```

```
case $F90 in
  gfortran*)
     F90FLAGS=-O2 -mavx
     ;;
  ifort)
     F90FLAGS=-O2 -xAVX
     ;;
  pgf90)
     F90FLAGS=-O2 -fastsse
     ;;
  *)
     echo Unknown F90 compiler
     exit 1
     ;;
esac
```

For

```
for VARIABLE in LIST
do
    commands
done
```

```
for i in *.F90
do
   mv $i $(basename $i .F90).f90
done

for i in figure_{3..5}.pdf
do
   convert $i $(basename $i .pdf).eps
done
```

Loops can also be written using C-style:

```
for ((i=0; i<10; i++))
do
  echo $i
done
```

While

```
while [[ expression ]]
do
    commands
done
```

```
declare -i i=1
while [[ $i -lt 100 ]]
do
  A+=" $i"
  i+=$i
done
echo $A

1 2 4 8 16 32 64
```

Until

Same as while, but with negated condition

```
declare -i i=1
until [[ $i -gt 100 ]]
do
  A+=" $i"
  i+=$i
done
echo $A

1 2 4 8 16 32 64
```

Command-line arguments

- `*` Expands to the arguments, starting from one.
- `@` Same as `*` but different when within double quotes
- `#` Number of arguments
- `-` Current option flags given to bash
- `0` 0 Expands to the name of the script
- `_` Absolute pathname used to invoke the script
- `1,2,...,N` Expands to the argument

```
#!/bin/bash

echo Script: $0
echo $# arguments
echo 2nd argument : $2

echo '$*'
for i in "$*"
do
   echo $i
done

echo '$@'
for i in "$@"
do
  echo $i
done
```

```
$ ./test.sh hello GDR Correl
Script: ./test.sh
3 arguments
2nd argument : GDR
$*
hello GDR Correl
$@
hello
GDR
Correl
```

Shift

Pops the 1st arguments of the command line and shift the next ones one the left.

```
#!/bin/bash
echo $@
shift
echo $@
shift 2
echo $@
```

```
$ test.sh one two three four five
one two three four five
two three four five
four five
```

```
#!/bin/bash

until [[ -z $@ ]]
do
  echo $1 $2
  shift 2
done
```

```
$ ./shift.sh one two three four five six
one two
three four
five six
```

Set

Takes any arguments and assigns them to the positional parameters (`$0..$n`).

```
#!/bin/bash

set one two three four five six

until [[ -z $@ ]]
do
  echo $1 $2
  shift 2
done
```

```
$ ./shift.sh
one two
three four
five six
```

getopt

Parse command line parameters

- `-o` : Short options list
- `-l` : Long options list
- `-n` : Name reported when getopt returns errors
- If an option is followed by `:`, it needs an argument

Move commands on the left:

```
$ getopt -o "1:23" -l "one:,two,three" -- test \
  -1 three --one=two arg1 arg2 -2 --three
-1 'three' --one 'two' -2 --three -- 'test' 'arg1'
'arg2'
```

```
#!/bin/bash

ARGS=$(getopt -o "1:23" -l "one:,two,three" -n $0 -- "$@")

[[ $? -eq 0 ]] || exit 1

eval set -- "$ARGS"

while true
do
  case "$1" in
    -1|--one)
      echo "one : " $2
      shift 2;;

    -2|--two)
      echo "Two"
```

```
    -3|--three)                 shift;;
      echo "Three"
      shift;;

    --)
      shift
      break;;
  esac
done
```

Functions

Functions can be defined in the shell:

```
my_func()
{
  COMMANDS
  return INTEGER
}
my_func
```

The arguments of the function are positional arguments inside the functions. Return code is optional

```
#!/bin/bash
get_cpu_load()
{
  local A
  A=$(uptime | cut -d: -f4)
```

```
  echo $A | cut -f$1 -d,
}
get_cpu_load 1
echo Current CPU load: $(get_cpu_load 2)
```

```
$ ./test.sh
0.61
Current CPU load: 0.29
```

User Interaction : Select

```
select F90 in gfortran ifort pfg90 other
do
  echo "Choose $F90?"
  read result
  if [[ $result = "y" ]] || [[ $result = "Y" ]] ; then
    break
  fi
done
```

read

Reads one line of input

```
$ read
toto
$ echo $REPLY
toto
$ read VAR
toto
$ echo $VAR
toto
$ read VAR1 VAR2
toto titi tata
$ echo $VAR1
toto
$ echo $VAR2
titi tata
```

- `-r` : Read raw input: does not interpret expansions and `\`
- `-d` : Set delimiter instead of newline
- `-n` : Read n characters
- `-p` : Prompt string
- `-s` : Secure input (passwords)

```
function pause()
{
  local X
  read -s -r -n 1 -p \
  "Press any key to continue..." X
}
```

```
function asksure() {
echo -n "Are you sure (Y/N)? "
while read -r -n 1 -s answer; do
  if [[ $answer = [YyNn] ]]; then
     [[ $answer = [Yy] ]] && retval=0
     [[ $answer = [Nn] ]] && retval=1
    break
  fi
done
return $retval
}

if asksure; then
  echo "Okay, performing rm -rf / then, master...."
else
  echo "Pfff..."
fi
```

Examples


Université
Paul Sabatier
TOULOUSE III


LCPQ


cnrs
dépasser les frontières

Example 1: xargs

You are working on a cluster and you have submitted hundreds of jobs by mistake. You want to kill all your jobs in the queue. On your cluster, the `qstat` command returns this output:

```
$ qstat
job-ID  prior   name       user         state submit/start at     queue
---------------------------------------------------------------------------------
  82851 2.50000 job_dummy  scemama      r     04/10/2013 13:45:51 all.q@compute-1-3.local
  82860 2.50000 job_dummy  scemama      r     04/10/2013 13:45:51 all.q@compute-1-3.local
  82868 2.50000 job_dummy  scemama      r     04/10/2013 13:45:51 all.q@compute-1-3.local
  82875 2.50000 job_dummy  scemama      r     04/10/2013 13:45:52 all.q@compute-1-3.local
  [...]
  82942 1.47958 job_dummy  scemama      qw    04/10/2013 13:45:55
  82943 1.46969 job_dummy  scemama      qw    04/10/2013 13:45:55
  82944 1.45999 job_dummy  scemama      qw    04/10/2013 13:45:55
  82902 1.45048 job_dummy  scemama      qw    04/10/2013 13:45:53
```

- Read the output of qstat without the 2 first lines using

```
qstat | tail --lines=+3
```

- Extract the 8 first characters. This corresponds to the job ID

```
qstat | tail --lines=+3 | cut -b-8
```

- Now, use this output as command-line arguments of the `qdel` command

```
$ qstat | tail --lines=+3 | cut -b-8 | xargs qdel
scemama has registered the job 82851 for deletion
scemama has registered the job 82860 for deletion
scemama has registered the job 82868 for deletion
  [...]
scemama has deleted job 82943
scemama has deleted job 82944
scemama has deleted job 82902
$ qstat
$
```

Example 2 : Using compressed files

You use a program that generates very large files. You want this files to be gzipped and gunzipped on the fly, and you don't have access to the source of the program.

For the example, we use the following program:

- If the `-c` option is present, it creates a 2000x2000 matrix filled with random numbers and the matrix is written in the `matrix` file
- If the `-c` option is not present, it reads the matrix from the file
- The program returns the max and min elements of the matrix

```
$ /usr/bin/time ./minmax -c
 Creating Matrix
 Writing Matrix
 Min:  6.94080884877656956E-007
 Max:  0.9999989401156275
5.74user 0.16system 0:06.09elapsed 96%CPU (0avgtext+0avgdata
```

```
128432maxresident)k
0inputs+398440outputs (0major+8100minor)pagefaults 0swaps

$ /usr/bin/time ./minmax
 Min:  6.94080884877656956E-007
 Max:  0.9999998940115627 5
4.39user 0.03system 0:04.42elapsed 99%CPU (0avgtext+0avgdata
128416maxresident)k
0inputs+0outputs (0major+8111minor)pagefaults 0swaps

$ ls -sh matrix*
195M matrix
```

Here is a script that will start `gzip` or `gunzip` in the background to gzip or gunzip your large file on the fly through a pipe.

```
#!/bin/bash
mkfifo matrix
if [[ $1 == -c ]]
then
  gzip < matrix > matrix.gz &
else
  gunzip < matrix.gz > matrix &
fi
./minmax $@
rm matrix
```

```
$ /usr/bin/time ./minmax.sh -c
 Creating Matrix
 Writing Matrix
 Min:  6.94080884877656956E-007
 Max:  0.9999989401156275
```

```
16.32user 3.57system 0:10.20elapsed 194%CPU (0avgtext+0avgdata
128432maxresident)k
0inputs+103816outputs (0major+9473minor)pagefaults 0swaps

$ ls -sh matrix*
51M matrix.gz

$ /usr/bin/time ./minmax.sh
 Min:  6.94080884877656956E-007
 Max:  0.9999998940115627 5
6.31user 0.59system 0:05.48elapsed 125%CPU (0avgtext+0avgdata
128416maxresident)k
0inputs+0outputs (0major+9796minor)pagefaults 0swaps
```

Example 3 : Monitoring CPU load

You want to monitor graphically your CPU load in real time.

Step 1

Use the uptime command to get the CPU load, and save it to a data file every second:

```
#!/bin/bash
DATA_FILE=/tmp/data_file
rm -f $DATA_FILE
while true
do
  uptime >> $DATA_FILE
# 01:34:38 up  4:20,  2 users, load average: 0.31, 0.20, 0.16
  sleep 1
done
```

Step 2

Change the script command to filter out useless data:

```
#!/bin/bash
DATA_FILE=/tmp/data_file
rm -f $DATA_FILE
while true
do
  uptime | cut -b40- | cut -d: -f2 | tr -d "," >> $DATA_FILE
  # 0.31 0.20 0.16
  sleep 1
done
```

Step 3

Plot the data file using gnuplot:

```
#!/bin/bash
DATA_FILE=/tmp/data_file
```

```
gnuplot --persist << EOF
  unset key
  plot '$DATA_FILE' using :1 with lines
replot '$DATA_FILE' using :2 with lines
replot '$DATA_FILE' using :3 with lines
EOF
```

Step 4

Create a pipe to control gnuplot

```
#!/bin/bash
DATA_FILE=/tmp/data_file
GNUPLOT_PIPE=/tmp/gnuplot_pipe

# If the pipe doesn't exist, create it
[[ -e $GNUPLOT_PIPE ]] || mkfifo $GNUPLOT_PIPE

# Push commands to the pipe in the background
```

```
cat > $GNUPLOT_PIPE << EOF &
  unset key
  plot '$DATA_FILE' using :1 with lines
replot '$DATA_FILE' using :2 with lines
replot '$DATA_FILE' using :3 with lines
EOF

# Start gnuplot and pull stdin from the pipe
gnuplot --persist < $GNUPLOT_PIPE

# Clean up pipe on exit
rm $GNUPLOT_PIPE
```

Alternative way:

```
#!/bin/bash
DATA_FILE=/tmp/data_file
GNUPLOT_PIPE=/tmp/gnuplot_pipe
```

```
# If the pipe doesn't exist, create it
[[ -e $GNUPLOT_PIPE ]] || mkfifo $GNUPLOT_PIPE

# Start gnuplot and pull stdin from the pipe (background)
gnuplot --persist < $GNUPLOT_PIPE &

# Push commands to the pipe
cat > $GNUPLOT_PIPE << EOF
  unset key
  plot '$DATA_FILE' using :1 with lines
replot '$DATA_FILE' using :2 with lines
replot '$DATA_FILE' using :3 with lines
EOF

# Clean up pipe on exit
rm $GNUPLOT_PIPE
```

Step 5

Use `tail` to keep the stdin of gnuplot open

```
#!/bin/bash
DATA_FILE=/tmp/data_file
GNUPLOT_PIPE=/tmp/gnuplot_pipe

# If the pipe doesn't exist, create it
[[ -e $GNUPLOT_PIPE ]] || mkfifo $GNUPLOT_PIPE

# Start gnuplot and pull stdin from the pipe (background)
tail -f $GNUPLOT_PIPE | gnuplot &

# Push commands to the pipe
cat > $GNUPLOT_PIPE << EOF
  unset key
  plot '$DATA_FILE' using :1 with lines
replot '$DATA_FILE' using :2 with lines
```

```
replot '$DATA_FILE' using :3 with lines
EOF

# Gnuplot is still alive
sleep 3
echo exit > $GNUPLOT_PIPE

# Clean up pipe on exit
rm $GNUPLOT_PIPE
```

Step 6

Combine everything

```
#!/bin/bash
DATA_FILE=/tmp/data_file
GNUPLOT_PIPE=/tmp/gnuplot_pipe

# If the pipe doesn't exist, create it
```

```
[[ -e $GNUPLOT_PIPE ]] || mkfifo $GNUPLOT_PIPE

# Start from an empty data file
rm -f $DATA_FILE
touch $DATA_FILE

# Start gnuplot
tail -f $GNUPLOT_PIPE | gnuplot &
cat > $GNUPLOT_PIPE << EOF
    unset key
    plot '$DATA_FILE' using :1 with lines
  replot '$DATA_FILE' using :2 with lines
  replot '$DATA_FILE' using :3 with lines
EOF

# On Ctrl-C, remove $DATA_FILE
trap "rm $DATA_FILE" SIGINT
```

```
# Write CPU load to file as long as the
# $DATA_FILE exists
while [[ -f $DATA_FILE ]]
do
  uptime | cut -b40- | cut -d: -f2 | tr -d "," >> $DATA_FILE
  echo replot > $GNUPLOT_PIPE
  sleep 1
done

# Exit cleanly gnuplot
echo exit > $GNUPLOT_PIPE

# Cleanl up the pipe
rm $GNUPLOT_PIPE

echo Clean exit
```

Index