Introduction à *$BASH* (Bourne-again shell)

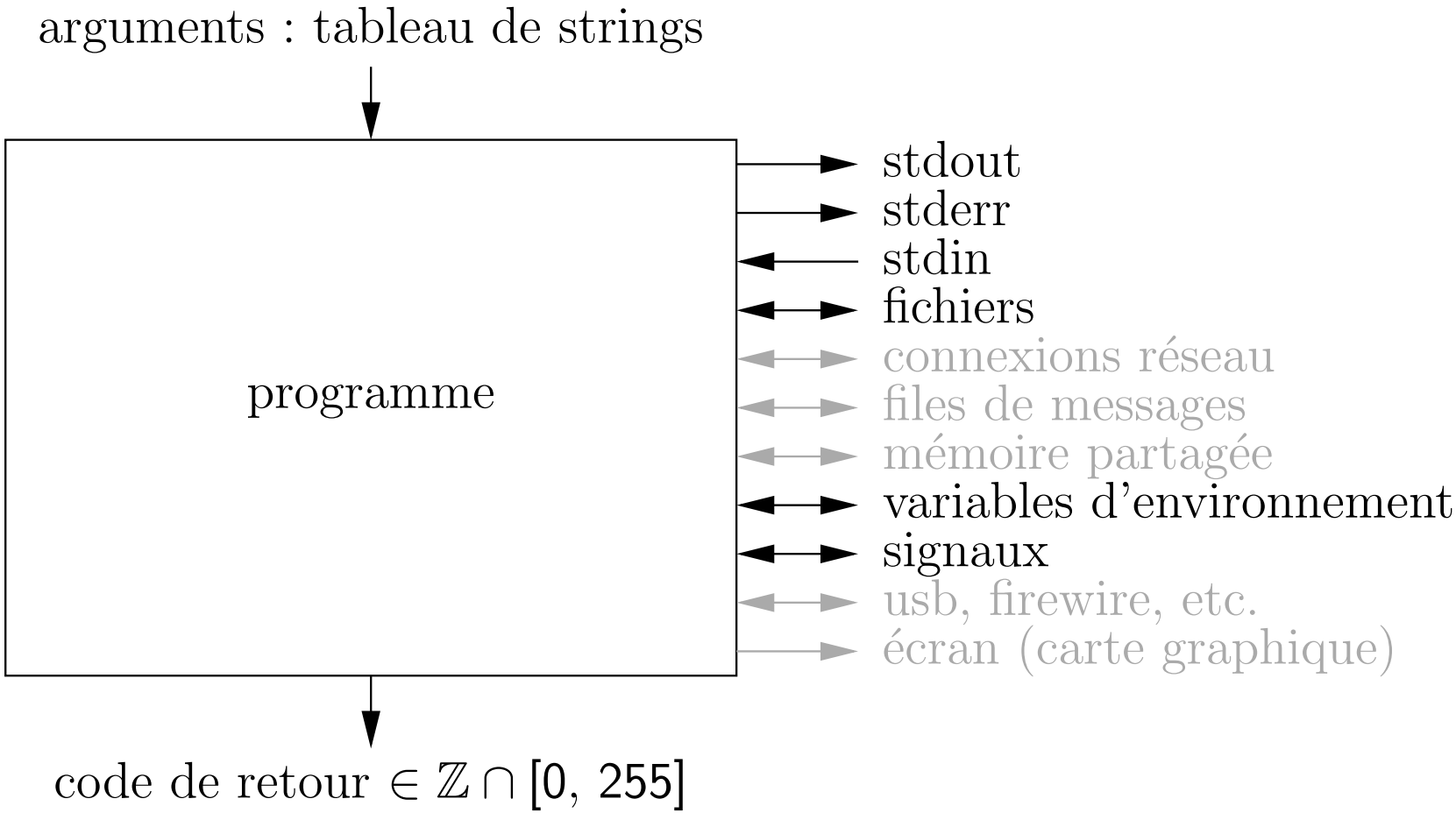
Sébastien PIÉRARD

Institut Montefiore, Université de Liège — 28 mars 2014

Outline

Outline

Un programme qui ne communique pas, ça ne sert à rien !



BASH est un programme qui interprète un langage qui sert principalement à gérer l'ensemble des programmes qui s'exécutent et les communications entre eux.

Le réflexe : lire la documentation !

- sudo apt-get install manpages-fr manpages-fr-dev manpages-fr-extra
- man est le programme à utiliser pour lire les pages de manuel
  - man man
  - man printf (/usr/bin/printf) (h,q,/,...)
  - man 1 printf (/usr/bin/printf)
  - man 3 printf (stdio.h)
  - man ascii
- si vous n'avez aucune idée d'où est la documentation
  - man -K printf
  - apropos printf
- Il y a aussi info
  - info coreutils (p, n, u, ↑, ↓)
  - info coreutils 'printf invocation'
- Et pour les commandes de bash (help, fg, bg, jobs, echo, printf, kill...), on peut faire help

Outline

Les redirections : stdout vers fichier

```
1  # créons un fichier vide
2  rm -f snoopy.txt
3  ls -alh snoopy.txt
4  touch snoopy.txt
5  ls -alh snoopy.txt
6  cat snoopy.txt
7
8  # premier test
9  echo 'The best dog of the world is ...' > snoopy.txt
10 cat snoopy.txt
11 echo 'Snoopy !' > snoopy.txt
12 cat snoopy.txt
13
14 # effacons ce fichier
15 rm snoopy.txt
16
17 # deuxième test
18 echo 'The best dog of the world is ...' > snoopy.txt
19 echo 'Snoopy !' >> snoopy.txt
20 cat snoopy.txt
```

Les redirections : stdout et/ou stderr vers fichier

Commençons par créer un fichier stdin_stdout.c :

```
1  #include <stdlib.h>
2  #include <stdio.h>
3  #define EOL "\n" // for linux / mac os X
4  int main ( int argc , char * argv [] ) {
5    if ( argc != 1 ) {
6      fprintf ( stderr , "Usage : %s" , argv [ 0 ] ) ;
7      exit ( EXIT_FAILURE ) ;
8    }
9    fprintf ( stdout , "Message écrit sur stdout" EOL ) ;
10   fprintf ( stderr , "Message écrit sur stderr" EOL ) ;
11   exit ( EXIT_SUCCESS ) ;
12 }
```

Puis exécutons ceci :

```
1  gcc stdin_stdout.c -o stdin_stdout
2  ./stdin_stdout
3  ./stdin_stdout > snoopy.txt
4  cat snoopy.txt
5  ./stdin_stdout 2> snoopy.txt
6  cat snoopy.txt
7  ./stdin_stdout &> snoopy.txt
8  cat snoopy.txt
```

Les redirections : stdout et/ou stderr vers fichier

Que font ces commandes ?

- ./ stdin_stdout > snoopy_stdout.txt 2> snoopy_stderr.txt
- ./ stdin_stdout > /dev/null 2> snoopy_stderr.txt
- ./ stdin_stdout > snoopy_stdout.txt 2> /dev/null

Préparons un fichier comme ceci :

```
1  n=10 ;
2  rm -f snoopy.txt ;
3  for i in `seq 1 $n` ; do
4    printf 'La ligne %03d\n' $i >> snoopy.txt ;
5  done
```

ou comme ceci :

```
1  n=10 ;
2  for i in `seq 1 $n` ; do
3    printf 'La ligne %03d\n' $i ;
4  done > snoopy.txt
```

ou encore comme ceci :

```
1  n=10 ;
2  for (( i = 1 ; i <= n ; ++ i )) ; do
3    printf 'La ligne %03d\n' $i ;
4  done > snoopy.txt
```

stdout | stdin

Que font ces commandes ?

- cp snoopy.txt tmp.txt ; **cat** tmp.txt >> snoopy.txt ; rm tmp.txt
- **cat** snoopy.txt
- **cat** snoopy.txt | head −n 3
- **cat** snoopy.txt | head −n −3
- **cat** snoopy.txt | tail −n 3
- **cat** snoopy.txt | tail −n +3
- **cat** snoopy.txt | wc
- **cat** snoopy.txt | wc −l
- **cat** snoopy.txt | sort −R
- **cat** snoopy.txt | shuf
- **cat** snoopy.txt | sort
- **cat** snoopy.txt | wc

stdout | stdin

Que font ces commandes ?

- **cat** snoopy.txt | tr [: digit :] '*' | tr L l
- **cat** snoopy.txt | cut −c 2−8
- **cat** snoopy.txt | cut −c 2−
- **cat** snoopy.txt | cut −d ' ' −f 3
- **cat** snoopy.txt | **sed** 's/0$/Z/g'
- **history** | tail

Que fait cette commande ?

- find ~ -maxdepth 2 -**type** f -name "*.jpg"-print0 | xargs --null -n 1 -I _@_ **echo** _@_

Et en supposant qu'il n'y a pas d'espaces dans les noms de dossiers ou de fichiers, que fait celle-ci ?

- find ~ -maxdepth 1 -**type** f -name "*.jpg"| xargs -n 1 -I _@_ **echo** "echo -n . ; convert \"_@_\" \"/tmp/\`basename _@_ .jpg\`.png\"" | $BASH ; **echo** ''

Autre exemple :

- find ~ -**type** f -name "*.sh" -print0 | xargs --null -n 1 -I _@_ grep -n -H -i sebastien '_@_' ;

Conseil : attention aux caractères spéciaux (espace , ' , ", etc.) dans les noms de fichiers ! Utilisez-les le moins possible, ça vous évitera des tracas inutiles . . .

Outline

La hiérarchie des processus et sa gestion par les signaux

- **ps** aux
- top
- htop
- free −m
- pstree
- **jobs**
- ctrl −z suivi de **bg**
- **fg**
- **kill**
- killall
- $!
- $$

Que fait cette commande ?

- **kill** −9 $$

Outline

Les variables

```
1  #!/bin/bash -x
2
3  MAVARIABLE=Valeur_1
4  function mafonction {
5    local MAVARIABLE=Valeur_2
6    echo $MAVARIABLE
7  }
8  echo $MAVARIABLE
9  mafonction
10 echo $MAVARIABLE
```

Variables spéciales :

- $! : PID du dernier processus lancé
- $$: PID du processus courant
- $? : résultat de la commande précédente
- $# : nombre d'arguments

Opérations arithmétiques

- +
- −
- *
- /
- %
- $ ((...))

Autres opérateurs

- ||
- &&
- -lt
- -gt
- -le
- -ge
- -eq
- -ne

Expressions conditionnelles

```
1  #!/bin/bash
2
3  T1="foo"
4  T2="bar"
5
6  if [ "$T1" = "$T2" ];
7  then
8    echo expression evaluated as true
9  else
10   echo expression evaluated as false
11 fi
```

Que se passe-t-il si on retire les guillemets dans le if et qu'un argument est vide ?

Expressions conditionnelles

```
1  function ExitIfNotDirectory {
2    [ -d "$1" ]
3    if [ $? != 0 ]
4    then
5      echo "error : $1 is not a directory"
6      exit
7    fi
8  }
9
10 function ExitIfNotFile {
11   [ -f "$1" ]
12   if [ $? != 0 ]
13   then
14     echo "error : $1 is not a file"
15     exit
16   fi
17 }
18
19 function ExitIfNotPositiveInteger {
20   if [ ! $(echo "$1" | grep -E "[0-9]+") ]
21   then
22     echo "error : $1 is not a positive integer"
23     exit
24   fi
25 }
```

Les boucles while

Is this code correct ?

```
1  #!/bin/bash
2
3  n=0;
4  find / -maxdepth 1 | while read line ; do
5    echo "files [ $n ] : $line" ;
6    n=$((n+1)) ;
7  done
8  echo "$n files have been found." ;
```

This is better :

```
1  #!/bin/bash
2
3  n=0;
4  while read line ; do
5   echo "files [ $n ] : $line" ;
6    n=$((n+1)) ;
7  done < <( find / -maxdepth 1 )
8  echo "$n files have been found." ;
```