

Unix et Programmation Shell

Philippe Langevin

département d'informatique
UFR sciences et technique
université du sud Toulon Var

Automne 2013

brouillon en révision

- site du cours :
<http://langevin.univ-tln.fr/cours/UPS/upsh.html>
- localisation du fichier :
<http://langevin.univ-tln.fr/cours/UPS/doc/direct.pdf>

dernières modifications

perm.tex	2016-09-30	09:41:54.766553521	+0200
file.tex	2016-09-30	09:19:02.810595120	+0200
bash.tex	2016-09-15	12:09:09.887948313	+0200
term.tex	2016-09-14	18:50:05.124091515	+0200
upsh.tex	2015-10-25	18:09:36.027434338	+0100
proc.tex	2015-10-20	22:09:35.450391618	+0200
man.tex	2015-09-18	20:13:44.822492893	+0200
shell.tex	2015-09-10	19:31:04.581529236	+0200
prologue.tex	2015-09-07	09:06:31.773157847	+0200
tools.tex	2015-07-11	09:04:38.890915266	+0200
pipe.tex	2014-10-02	19:10:22.426127326	+0200
direct.tex	2014-10-02	07:49:17.162784238	+0200
syntaxe.tex	2014-10-01	23:52:29.859357485	+0200
part.tex	2014-10-01	23:52:29.372363438	+0200

7 - redirection

fichier C

La bibliothèque `glibc` propose aux C-programmeurs de puissantes fonctions d'entrées/sorties pour opérer sur les **flux**.

```
1 #include <stdlib.h>
2 #include <stdio.h>
3 int main( int argc, char *argv[] )
4 {
5     char* msg = "hello";
6     FILE *dst;
7     dst = fopen( "/tmp/dst.txt", "w" );
8     if ( ! dst ) perror( "fopen" ), exit (1);
9     fprintf (dst, "%s", msg );
10    fclose(dst);
11    return 0;
12 }
```

flux

Les **flux** sont g er  par une structure relativement complexe :

```
~> gdb --quiet  
-x flux.gdb flux.  
exe  
| grep -A14  
fileno  
> flux.out
```

```
1 set pagination off  
2 break main  
3 run  
4 whatis FILE  
5 ptype FILE  
6 quit
```

flux

Les **flux** sont g er  par une structure relativement complexe :

```

~> gdb --quiet
  -x flux.gdb flux.
  exe
  | grep -A14
  fileno
  > flux.out

```

```

1 set pagination off
2 break main
3 run
4 whatis FILE
5 ptype FILE
6 quit

```

```

1 type = struct _IO_FILE {
2   int _flags ;
3   char *_IO_read_ptr;
4   char *_IO_read_end;
5   char *_IO_read_base;
6   char *_IO_write_base;
7   char *_IO_write_ptr ;
8   char *_IO_write_end;
9   char *_IO_buf_base;
10  char *_IO_buf_end;
11  char *_IO_save_base;
12  char *_IO_backup_base;
13  char *_IO_save_end;
14  struct _IO_marker *_markers;
15  struct _IO_FILE *_chain;
16  int _fileno ;

```

descripteur

Un descripteur de fichier est une entrée dans une table des fichiers.
Les ES sont gérées par des appels de bas niveau : `read`, `write`.

```
1 #include <sys/types.h>
2 #include <sys/stat.h>
3 #include <fcntl.h>
4 #include <stdlib.h>
5 #include <unistd.h>
6 int main( int argc, char *argv[] )
7 {
8     char* msg = "hello";
9     int fd;
10    fd = open( "/tmp/dst.txt", O_WRONLY );
11    if ( fd < 0 ) exit (1);
12    write( fd , msg, 6);
13    close( fd );
```


cat

La commande `cat` copie sur la sortie standard :

```
~> cat  
Hello World      <-  clavier  
Hello World      <-  ecran
```

Les descripteurs standards peuvent être redirigés vers des descripteurs ou des fichiers

```
~> cat >fichier  
Hello World      <-  clavier
```

cat

La commande `cat` copie sur la sortie standard :

```
~> cat  
Hello World      <-  clavier  
Hello World      <-  ecran
```

Les descripteurs standards peuvent être redirigés vers des descripteurs ou des fichiers

```
~> cat >fichier  
Hello World      <-  clavier
```

```
~> cat fichier  
Hello World
```

cat

La commande `cat` copie sur la sortie standard :

```
~> cat  
Hello World      <- clavier  
Hello World      <- ecran
```

Les descripteurs standards peuvent être redirigés vers des descripteurs ou des fichiers

```
~> cat >fichier  
Hello World      <- clavier
```

```
~> cat fichier  
Hello World
```

```
~> cat <fichier  
Hello World
```

cat

La commande `cat` copie sur la sortie standard :

```
~> cat  
Hello World      <- clavier  
Hello World      <- ecran
```

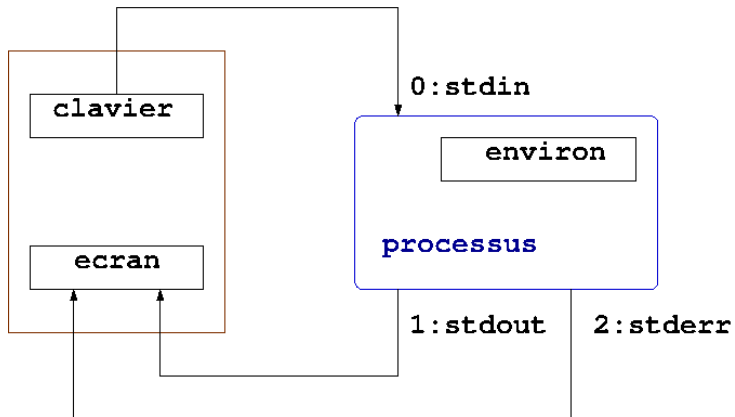
Les descripteurs standards peuvent être redirigés vers des descripteurs ou des fichiers

```
~> cat >fichier  
Hello World      <- clavier
```

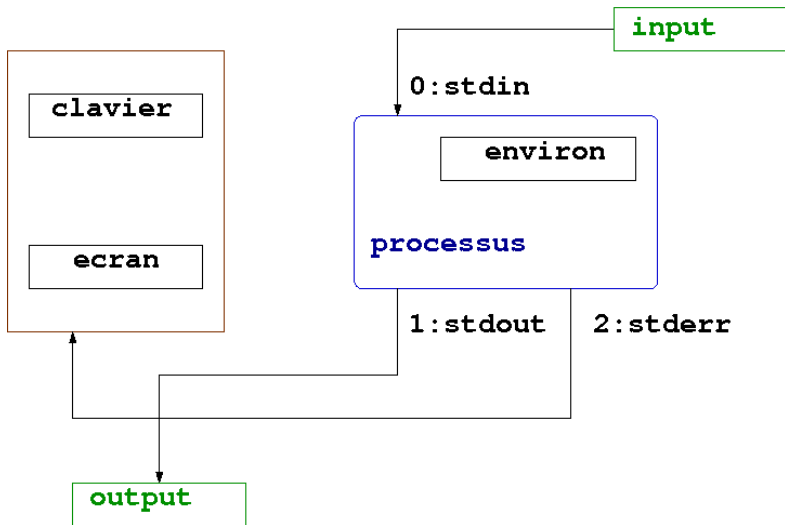
```
~> cat fichier  
Hello World
```

```
~> cat <fichier  
Hello World
```

fichier standard



redirection



descripteurs et fichiers spéciaux

- 0 : entrée standard
- 1 : sortie standard
- 2 : sortie erreur standard
- `extern FILE *stdin;`
- `extern FILE *stdout;`
- `extern FILE *stderr;`
- `/dev/null`
- `/dev/random`
- `/dev/zero`
- `/dev/tcp/hote/port`

évaluation

debug Condidérons les fichiers :

```
1 cat <input >output
```

direct.sh

```
1 cat <input >output 2>&1
```

double.sh

```
1 cat <input 2>&1 >output
```

erreur.sh

d'après le manuel, la troisième est incorrecte.

```
→ strace -o {fic} -ff -e trace={lst} bash {  
script}
```


En effet

```
1 :: cat direct .sh erreur 2>&1 > /tmp/foo
2 cat: erreur : Aucun fichier ou dossier de ce type
3
4 :: cat direct .sh erreur > /tmp/bar 2>&1
```

En effet

```
1 :: cat direct .sh erreur 2>&1 > /tmp/foo
2 cat: erreur : Aucun fichier ou dossier de ce type
3
4 :: cat direct .sh erreur > /tmp/bar 2>&1
```

```
1 :: echo $?
2 1
```

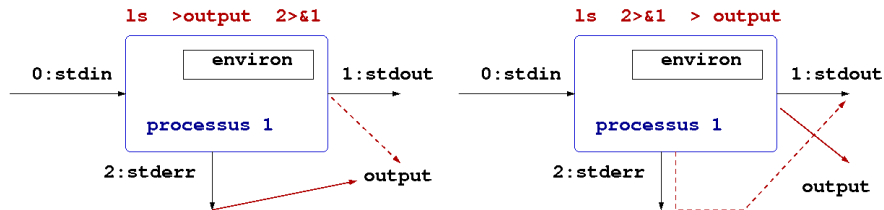
En effet

```
1 :: cat direct .sh erreur 2>&1 > /tmp/foo
2 cat: erreur : Aucun fichier ou dossier de ce type
3
4 :: cat direct .sh erreur > /tmp/bar 2>&1
```

```
1 :: echo $?
2 1
```

```
1 :: cat /tmp/foo
2 cat <input >output
3
4 :: cat /tmp/bar
5 cat <input >output
6 cat: erreur : Aucun fichier ou dossier de ce type
```

direction : gauche à droite !



```
1 ls >output 2>&1
2 ls 2>&1 >output
```

BASH(1)

Il y a deux formes pour effectuer une double redirection :

`&>mot` et `>&mot`

On préfère généralement la première, équivalente à :

`>mot 2>&1`

aspect syntaxique

Avec `bash`, les entrées/sorties sont redirigeables à l'aide des `<`, `>` :

```
1 cat <input >output
2 >output cat <input
```

sont des abbréviations de

```
1 cat 0<input 1>output
2 1>output cat 0<input
```

```
1 3.6.1 Input      : [n] < word
2 3.6.2 Output    : [n] > word :simil : [n] >> word
3 3.6.4 Standards : &> word    :equiv: >word 2>&1
4 3.6.5 Appending : &>>word    :equiv: >> word 2>&1
5 3.6.6 Here Documents
6     <<[-]word
7     here-document
```

redir.sho

```
1 #!/bin/bash
2 set -v
3 nc -l 31415 > /tmp/nc.log &
4 sleep 1
5 netstat -plent --ip | sed -E 's/[ \t]+/ /g'
6 set +v
7 cat > /dev/tcp/localhost/31415 <<- this
8     bash redirige pas mal de chose !
9     this
10 set -v
11 cat /tmp/nc.log
```

redir.out

```
1 nc -l 31415 > /tmp/nc.log &
2 sleep 1
3 netstat -plent --ip | sed -E 's/[ \t]+/ /g'
4 (Tous les processus ne peuvent être identifiés, les infos sur
   les processus
5 non possédés ne seront pas affichées, vous devez être root
   pour les voir toutes.)
6 Connexions Internet actives (seulement serveurs)
7 Proto Recv-Q Send-Q Local Address Foreign Address State
   Utilisatr Inode PID/Program name
8 tcp 0 0 0.0.0.0:57250 0.0.0.0:* LISTEN 29 8802 -
9 tcp 0 0 0.0.0.0:111 0.0.0.0:* LISTEN 0 8689 -
10 tcp 0 0 0.0.0.0:22 0.0.0.0:* LISTEN 0 9505 -
11 tcp 0 0 0.0.0.0:31415 0.0.0.0:* LISTEN 501 59251 2845/nc
12 tcp 0 0 127.0.0.1:631 0.0.0.0:* LISTEN 0 9059 -
```

Tracer un processus

Les outils de traçages :

- `ltrace` : trace les appels aux bibliothèques.
- `strace` : trace les signaux et les appels systèmes.

permettent d'analyser finement les processus.

```
hello.out : hello.exe
strace -o hello.sys \
    -e read , write , open , execve , close , stat ./
    hello.exe
ltrace -o hello.lib ./hello.exe
ltrace -c -o hello.lc ./hello.exe
strace -c -o hello.tmp ./hello.exe
sed 's/[ ]\+/\t/g' hello.tmp > hello.sc
awk '/hello.out/,/@@@/' makefile > hello.out
```

makefile

src/hello.c

```
1 #include <stdlib.h>
2 #include <stdio.h>
3 int main( int argc, char*argv[] )
4 {
5     int nb;
6     FILE *dst;
7     dst = fopen( "hello.txt", "w" );
8     if ( ! dst ) perror( "fopen" ), exit (1);
9     nb = fprintf ( dst, "%s", argv[0] );
10    fprintf ( stderr , "%d" , nb );
11    return 0;
12 }
```

remarque

Le code \LaTeX du transparent suivant contient la ligne :

```
\lstinputlisting[language=make]{src/hello.out}
```

qui explique la dernière ligne des actions de la cible `hello.out`.

hello.exe : trace des appels bibliothèques

% time	seconds	usecs/call	calls	function
53.89	0.000603	603	1	fopen
46.11	0.000516	258	2	fprintf
100.00	0.001119		3	total

hello.exe : trace des appels systèmes

% time	seconds	usecs/call	calls	errors	syscall
91.34	0.000738	738	1		execve
8.66	0.000070	8	9		mmap
0.00	0.000000	0	1		read
0.00	0.000000	0	2		write
0.00	0.000000	0	3		open
0.00	0.000000	0	2		close
0.00	0.000000	0	3		fstat
0.00	0.000000	0	3		mprotect
0.00	0.000000	0	1		munmap
0.00	0.000000	0	3		brk
0.00	0.000000	0	1		l_access
0.00	0.000000	0	1		arch_prctl
100.00	0.000808	30	1		total

hello.exe : trace des appels bibliothèques

```
__libc_start_main(0x4005e4, 1, 0x7fffae1cd278, 0
    x400680, 0x400670 <unfinished ... >
fopen("hello.txt", "w")
    = 0xae7010
fprintf(0xae7010, "%s", "./hello.exe")
    = 11
fprintf(0x395df76860, "%d", 11)
    = 2
+++ exited (status 0) +++
```

hello.exe : trace des appels systèmes

```
execve("./hello.exe", ["./hello.exe"], [/* 54 vars
    */]) = 0
open("/etc/ld.so.cache", O_RDONLY)          = 3
close(3)                                     = 0
open("/lib64/libc.so.6", O_RDONLY)         = 3
read(3, "\177ELF
    \2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0000\354\30
    832) = 832
close(3)                                     = 0
open("hello.txt", O_WRONLY|O_CREAT|O_TRUNC, 0666) =
    3
write(2, "11", 2)                            = 2
write(3, "./hello.exe", 11)                 = 11
```

open etc...

OPEN(2) Linux Programmer's Manual

SYNOPSIS

```
int open(const char *path, int flags);
int open(const char *path, int flags, mode_t md)
;
int creat(const char *path, mode_t mode);
```

DESCRIPTION

Given a pathname for a **file**, `open()` returns a **file descriptor**, a small, nonnegative integer for use in subsequent system calls.

The **file descriptor** returned by a successful call will be the lowest-numbered free **file descriptor**.

SEE ALSO

`chmod(2)`, `chown`, `close`, `dup`, `fcntl`, `link`, `lseek`,
`mknod`, `mmap`, `mount`, `openat`, `read`, `socket`, `stat`

table des descripteurs

Un processus possède une table de 255 descripteurs de fichiers, les trois premiers sont initialisés :

- 0 : entrée standard
- 1 : sortie standard
- 2 : erreur standard

L'appel système `open(fichier, mode)` configure un descripteur de fichier pour des opérations de lecture `read`, d'écriture `write` avec le fichier physique.

- -1 : appel système a échoué!

L'appel système `close` ferme un descripteur. Les appels système `dup` permettent de manipuler les descripteurs.

dupliquer

```
1 #include <sys/types.h>
2 #include <sys/stat.h>
3 #include <fcntl.h>
4 #include <stdlib.h>
5 #include <unistd.h>
6 #include <stdio.h>
7 int main( int argc, char*argv[] )
8 { int fd;
9 fd = open( argv[1],O_RDWR |O_CREAT|O_TRUNC, 0644 );
10 if ( fd < 0 ) perror( "open" ), exit (1);
11 close( 1 );
12 if ( dup( fd ) < 0 ) perror( "dup" ), exit (1);
13 close(fd);
14 if ( execlp( "ls", "ls", "-alt", NULL ) < 0 )
15     perror( "execlp" ) exit (1);
```

dupliquer

```
↪ gcc -Wall dup.c -o dup.exe
```

```
↪ ./dup.exe dup.txt
```

```
↪ head -5 dup.txt
```

```
total 300
-rw-r--r--. 1 pl    pl          0 14 sept. 14:07
  dup.txt
-rw-rw-r--. 1 pl    pl    1313 14 sept. 14:07
  sdup.out
-rw-rw-r--. 1 pl    pl    2160 14 sept. 14:07
  sdup.tmp
-rw-rw-r--. 1 pl    pl    2861 14 sept. 14:07
  ldup.out
```

dup

DUP(2) Linux Programmer's Manual

NAME

dup, dup2, dup3 – duplicate a file descriptor

SYNOPSIS

```
int dup(int oldfd);  
int dup2(int oldfd, int newfd);
```

DESCRIPTION

These system calls create a copy of oldfd.

dup() uses the lowest-numbered unused descriptor for the new descriptor.

dup2() makes newfd be the copy of oldfd, closing newfd first if necessary.

dup.exe : trace des appels bibliothèques

% time	seconds	usecs/call	calls	function
29.91	0.178987	121	1474	strlen
11.49	0.068741	121	564	__ctype_get_mb_cur_max
7.76	0.046463	15487	3	dcgettext
6.22	0.037220	120	310	memcpy
5.61	0.033547	120	278	fputs_unlocked
4.18	0.025027	120	208	localeconv
3.51	0.021030	404	52	iswprint
2.75	0.016455	5485	3	getpwuid
2.26	0.013501	192	70	strstr
2.03	0.012120	130	93	malloc
1.93	0.011560	167	69	acl_extended_file
1.83	0.010972	120	91	__errno_location
1.82	0.010910	158	69	lgetfilecon
1.70	0.010199	147	69	localtime
1.70	0.010196	147	69	__lxstat64
1.52	0.009074	120	75	strcpy
1.43	0.008538	121	70	readdir64
1.41	0.008452	122	69	__sprintf_chk
1.39	0.008315	120	69	fwrite_unlocked
1.38	0.008280	120	69	stpcpy
1.37	0.008205	118	69	__memcpy_chk
1.31	0.007868	655	12	wcswidth
1.12	0.006725	120	56	memset
0.85	0.005068	181	28	free
0.77	0.004597	191	24	mbstowcs
0.41	0.002436	203	12	nl_langinfo
0.38	0.002297	120	19	strcoll
0.38	0.002285	190	12	memcpy
0.33	0.001968	984	2	fflush

dup.exe : trace des appels systèmes

% time	seconds	usecs/call	calls	errors	syscall
--------	---------	------------	-------	--------	---------

56.65	0.000277	55	5		socket
31.08	0.000152	4	37		close
7.98	0.000039	1	47		mmap2
4.29	0.000021	0	69		lstat64
0.00	0.000000	0	23		read
0.00	0.000000	0	1		write
0.00	0.000000	0	38		10 open
0.00	0.000000	0	3		1 execve
0.00	0.000000	0	2		2 access
0.00	0.000000	0	1		dup
0.00	0.000000	0	4		brk
0.00	0.000000	0	2		2 ioctl
0.00	0.000000	0	13		munmap
0.00	0.000000	0	1		uname
0.00	0.000000	0	12		mprotect
0.00	0.000000	0	2		_llseek
0.00	0.000000	0	2		rt_sigaction
0.00	0.000000	0	1		rt_sigprocmask
0.00	0.000000	0	1		getrlimit
0.00	0.000000	0	69		stat64
0.00	0.000000	0	28		fstat64
0.00	0.000000	0	2		getdents64
0.00	0.000000	0	138		138 getxattr
0.00	0.000000	0	69		lgetxattr
0.00	0.000000	0	2		futex
0.00	0.000000	0	2		set_thread_area
0.00	0.000000	0	1		set_tid_address
0.00	0.000000	0	1		clock_gettime
0.00	0.000000	0	2		statfs64

Diagnostic

Nous pouvons faire une analyse des appels système du `bash` de deux manières. Dans les deux cas, il faut penser à tracer les sous-processus :

- 1 On place les commandes à analyser dans un script :

```
~> strace -o {fic} -ff -e trace={lst} bash {  
script}
```

- 2 On détermine le **pid** du shell à tracer, et on trace le les commandes à analyser par une capture :

```
~> strace -o {fic} -ff -e trace={lst} -p {pid  
}
```

direct.sh

```

close(255) = 0
open("input", O_RDONLY|O_LARGEFILE) = 3
dup2(3, 0) = 0
close(3) = 0
open("output", O_WRONLY|O_CREAT|O_TRUNC|O_LARGEFILE
, 0666) = 3
dup2(3, 1) = 1
close(3) = 0
execve("/bin/cat", ["cat"], [/* 46 vars */]) = 0
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
close(3) = 0
open("/lib/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF
\1\1\1\3\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0\220\347\31
B4\0\0\0"..., 512) = 512
close(3) = 0

```

double.sh

```

close(255) = 0
open("input", O_RDONLY|O_LARGEFILE) = 3
dup2(3, 0) = 0
close(3) = 0
open("output", O_WRONLY|O_CREAT|O_TRUNC|O_LARGEFILE
, 0666) = 3
dup2(3, 1) = 1
close(3) = 0
dup2(1, 2) = 2
execve("/bin/cat", ["cat"], [/* 46 vars */) = 0
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
close(3) = 0
open("/lib/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF
\1\1\1\3\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0\220\347\31
B4\0\0\0"
512) = 512

```


erreur.sh

```

close(255) = 0
open("input", O_RDONLY|O_LARGEFILE) = 3
dup2(3, 0) = 0
close(3) = 0
dup2(1, 2) = 2
open("output", O_WRONLY|O_CREAT|O_TRUNC|O_LARGEFILE
, 0666) = 3
dup2(3, 1) = 1
close(3) = 0
execve("/bin/cat", ["cat"], [/* 46 vars */) = 0
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
close(3) = 0
open("/lib/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF
\1\1\1\3\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0\220\347\31
B4\0\0\0" 512) = 512

```

substitution de processus

La substitution de processus s'appuie sur le mécanisme des tubes nommés (FIFOs) ou la méthode `/dev/fd` de noms de fichiers.

```
1  
2 comm -3 <(sort x | uniq) <(sort y | uniq)
```

Il ne s'agit pas d'une redirection !

man comm

COMM(1) User Commands

NAME comm – compare two sorted files line by line

SYNOPSIS comm [-123] FILE1 FILE2

Compare sorted files FILE1 and FILE2 line by line

With no options, produce three-column output. Column 1 contains lines unique to FILE1, column 2 contains lines unique to FILE2, and column three contains lines common to both files. -X suppress column X.

EXAMPLES

```
comm -12 file1 file2
```

Print only lines present in both file1 and file2.

```
comm -3 file1 file2
```

Print lines in file1 not in file2, and vice versa

subproc.sho

```
1 #!/bin/bash -v
2 #from Linux Journal
3 echo -e 'a\nf\nc\ng\ne\nb' > x
4 echo -e 'c\ne\na\nd\nb\nf' > y
5 sort x | uniq > /tmp/X
6 sort y | uniq > /tmp/Y
7 comm -3 /tmp/X /tmp/Y
8   d
9 g
10
11 comm -3 <(sort x | uniq) <(sort y | uniq)
12 sort y | uniq
13 sort x | uniq
14   d
15 g
```