Unix et Programmation Shell

Philippe Langevin

IMATH, USTV

Automne 2013

sommaire

1. Introduction
2. shell unix
3. manuel
4. login
5. bash
6. fichier
7. permission
8. processus
9. redirection
10. pipeline
11. outils

document

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/upsh.pdf

dernières modifications

```
     upsh.tex 2024-01-28 21:05:32.653614398 +0100
    tools.tex 2024-01-28 21:05:32.653614398 +0100
     term.tex 2024-01-28 21:05:32.651614387 +0100
  syntaxe.tex 2024-01-28 21:05:32.650614381 +0100
    shell.tex 2024-01-28 21:05:32.649614376 +0100
 prologue.tex 2024-01-28 21:05:32.648614371 +0100
     proc.tex 2024-01-28 21:05:32.646614360 +0100
     pipe.tex 2024-01-28 21:05:32.645614355 +0100
     perm.tex 2024-01-28 21:05:32.645614355 +0100
      man.tex 2024-01-28 21:05:32.643614344 +0100
     part.tex 2024-01-28 21:05:32.644614349 +0100
     file.tex 2024-01-28 21:05:32.642614338 +0100
   direct.tex 2024-01-28 21:05:32.641614333 +0100
     bash.tex 2024-01-28 21:05:32.639614322 +0100
```

1 - Introduction

- notions abordées
- partie cachée
- programme
- public et prérequis

Objectifs du cours

Il s'acquérir ou de compléter nos connaissances des systèmes `unix` du point de vue *utilisateur*. Nous aborderons néanmoins quelques notions fondamentales de ces systèmes d'exploitation :

- système de fichier
- processus, tube, redirection
- environnement, permission
- signaux, thread

pour appréhender l'usage :

- ligne de commande,
- programmation shell,
- commandes usuelles,
- applications populaires.

partie cachée

La ligne de commande est la partie visible de l'iceberg. Les aspects internes :

- contexte d'exécution,
- ordonnancement,
- mémoire,
- sémaphore,
- ipc. . .

où les aspects externes

- module,
- matériels,
- protocoles

seront évoqués à la demande mais sans jamais entrer dans les détails.

programme

L'unité d'enseignement *I54* prévoit 30 HE, avec la répartion :

12H	cours	6 séances
3H	travaux dirigés	2 séances
15H	travaux pratiques	5 séances
2H	examen	TP
2H	examen	CT

Intervenants :

- Didier Malarino
- Philippe Langevin

Public

I54 est un cours de mise à niveau et/ou de perfectionnement qui s'adresse idéalement aux étudiants ayant eu une expérience sur un système `unix`, typiquement, un compte utilisateur sous linux.

Prérequis : manipulation de bases des fichiers et répertoires sur la ligne de commande. Base du `langage C` et de la compilation avec `gcc`.

documentation

Le cours est accessible à partir du site

[local] http://langevin.univ-tln.fr

- marquer les pages
- cahier de texte
- liste de discussion

2 - shell unix

- origine
- unices
- GNU/linux
- distribution
- shell unix
- GUI vs CLI
- C-production

naissance de unix

Parmi les nombreux *hackers* du 20e, deux pionniers des BELL LABS sont à l'origine du système **unix** :

naissance de unix

Parmi les nombreux *hackers* du 20e, deux pionniers des BELL LABS sont à l'origine du système **unix** :

naissance de unix

Parmi les nombreux *hackers* du 20e, deux pionniers des BELL LABS sont à l'origine du système **unix** :

1969 Ken Thompson crée le système UNICS

naissance de unix

Parmi les nombreux *hackers* du 20e, deux pionniers des BELL LABS sont à l'origine du système **unix** :

1969 Ken Thompson crée le système UNICS

1971 Dennis Ritchie crée le `langage C`

naissance de unix

Parmi les nombreux *hackers* du 20e, deux pionniers des BELL LABS sont à l'origine du système **unix** :

1969 Ken Thompson crée le système UNICS

1971 Dennis Ritchie crée le `langage C`

[PSLC] prog. système en langage `C` sous Linux, par C. Blaess.

naissance de unix

Parmi les nombreux *hackers* du 20e, deux pionniers des BELL LABS sont à l'origine du système **unix** :

1969 Ken Thompson crée le système UNICS

1971 Dennis Ritchie crée le `langage C`

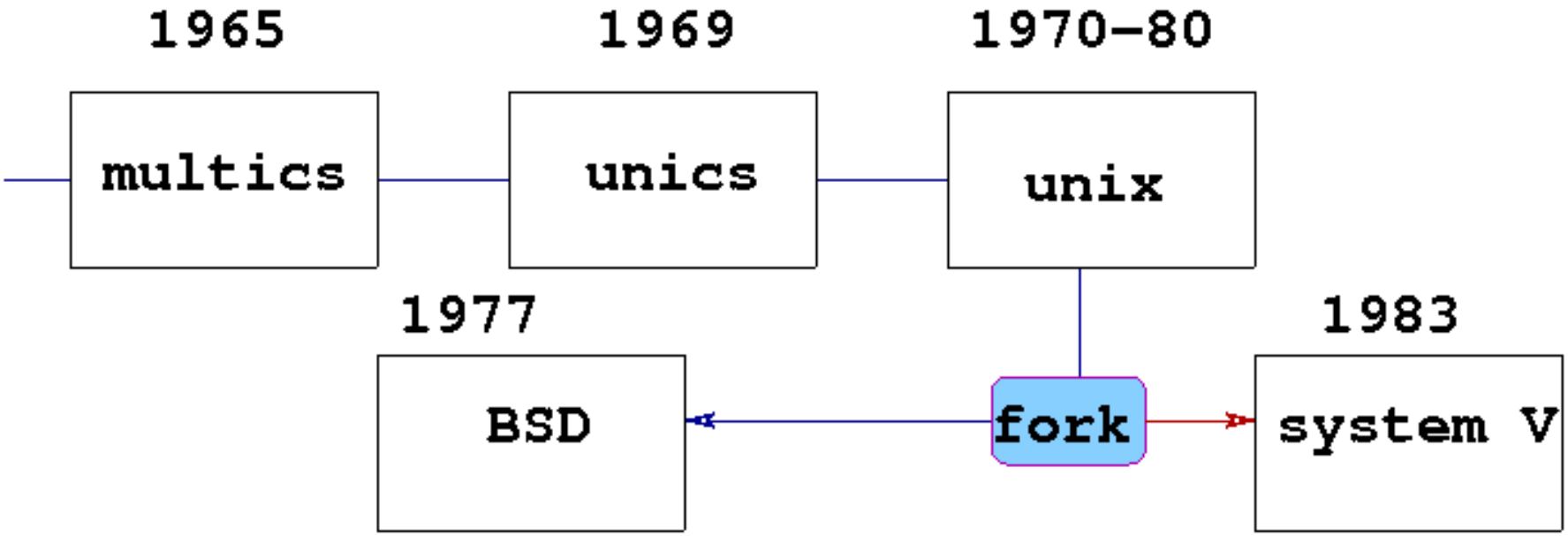
[PSLC] prog. système en langage `C` sous Linux, par C. Blaess.
[EPI] `unix` et l'informatique pédagogique ont le même age !

MULTICS — UNICS — UNIX

1964 MIT, General Electric et les Bell Labs d'AT&T lancent le projet MULTiplexed Information and Computing Service pour développer un nouveau système d'exploitation à temps partagé.

1969 Les Bell Labs se retire du projet.

chronologie



freeBSD	solaris	HP-UX
openBSD	linux	AIX
netBSD	GNU/linux	XENIX
mac OS X	android	ULTRIX

unix populaires

unix a donné naissance à une famille de systèmes, les *unices* dont les plus populaires sont :

1983	System V	Bell labs, AT&T.
1977	BSD	Berkeley Software Distribution
1990	GNU/Linux	Logiciel Libre
1999	OS X	next, apple.
2003	android	androïde, google.

L'ensemble des industriels acteurs du développemement du système **unix** sont regoupés dans l'opengroup propriétaire de la marque `unix` dont le *Single UNIX Specification* certifie les systèmes `unix`.

influences

Trois groupes influent sur la normalisation des systèmes `unix` :

- POSIX : Portable Operating System Interface (IEEE).
- BSD
- GNU : Gnu is Not Unix, logiciel libre.

Je vous recommande

- la description du projet GNU par R. Stallman
- la lecture de la cathédrale et le bazar par E. Raymond.

dialecte

```
PS(1)    Linux User's Manual
NAME
  ps - report a snapshot of the current processes.

DESCRIPTION
  ps displays information about a selection of the
active processes. If you want a repetitive update
of the selection and the displayed information,
use top.

This ps version accepts several kinds of options:
 1 UNIX options, must be preceded by a dash.
 2 BSD options,  must not be used with a dash.
 3 GNU long options, preceded by two dashes.
Options of different types may be freely mixed,
                        but conflicts can appear
```

GNU/linux

Dans les salles de travaux-pratiques, vous utiliserez un système d'exploitation GNU/linux, fusion des composantes du logiciel libre :

- noyau linux (Linus Torvalds, 1991),
- utilitaires GNU (Richard Stallman, 1983).

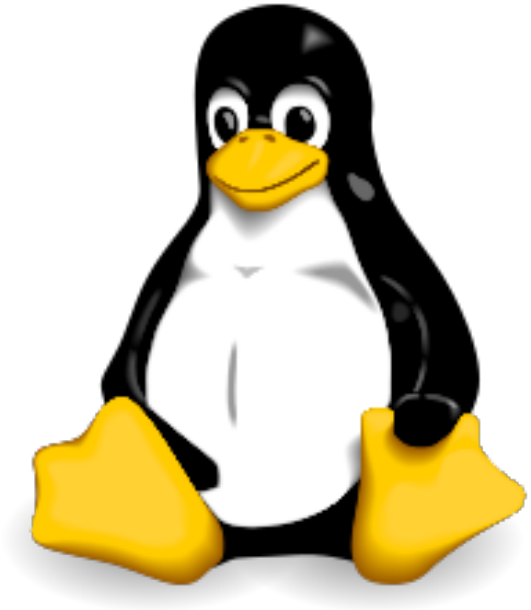
Plus précisément, une distribution ubuntu, basée sur debian. Il s'agit d'un environnement de travail unix de qualité issu du logiciel libre ? !.

- hurd : le noyau GNU n'est pas encore opérationnel.

GNU/linux

libérez unix!

Hello everybody out there using



minix

Le nom GNU/Linux fut utilisé pour la première fois par debian en 1994 pour le nom de leur distribution du système d'exploitation basé sur le noyau Linux et des logiciels GNU.

le 27 septembre 1983 : libérez `unix` !

À partir de Thanksgiving je vais écrire un système logiciel complet compatible avec Unix appelé GNU, et le distribuer librement à quiconque voudra l'utiliser. Il y a grand besoin de contributions sous forme de temps, d'argent, de programmes et d'équipement.
Pour commencer, GNU comprendra un noyau ainsi que tous les utilitaires requis pour écrire et faire tourner des programmes C : éditeur, interpréteur de commandes, compilateur C, éditeur de liens, assembleur et quelques autres encore. Par la suite, nous ajouterons un formateur de texte, un YACC, un jeu Empire, un tableur et des centaines d'autres choses. À terme, nous espérons fournir tous les composants utiles venant normalement avec un système Unix, ainsi que tout autre composant utile, y compris de la documentation en ligne et imprimée...

– Richard Stallman

paquets gnu (2012)

a2ps acct acm adns aetherspace alive anubis archimedes aris aspell auctex autoconf autoconf-archive autogen automake avl ballandpaddle barcode bash bayonne bazaar bc bfd binutils bison bool bpel2owfn c-graph ccaudio ccide ccrtp ccscript cflow cgicc chess cim classpath classpathx clisp cobol combine commoncpp complexity config coreutils cpio cppi cssc dap dc ddd ddrescue dejagnu denemo dia dico diction diffutils dionysus dismal djgpp dmd dominion dotgnu dotgnu-forum dotgnu-pnet dr-geo ed edma electric emacs emacs-muse emms enscript eprints epsilon fdisk ferret findutils fontutils freedink freefont freeipmi freetalk fribidi gama garpd gawk gcal gcc gcide gcl gcompris gdb gdbm gengen gengetopt gettext gforth ggradebook ghostscript gift gimp gleem glib global glpk glue gmediaserver gmorph gmp gnash gnat gnats gnatsweb gnome gnowsys gnu-arch gnu-c-manual gnu-crypto

gnuae gnubatch gnubg gnubiff gnubik gnucap gnucash gnucomm gnue gnufm gnugo gnuit gnujdoc gnujump gnukart gnulib gnumach gnumed gnumeric gnump3d gnun gnunet gnupg gnupod gnuprologjava gnuradio gnurobots gnuschool gnushogi gnuskies gnusound gnuspeech gnuspool gnustandards gnustep gnutls gnutrition gnuzilla goptical gorm gpaint gperf gprolog grabcomics greg grep gretl groff grub gsasl gsegrafix gsl gsrc gss gtick gtk+ gtypist guile guile-dbi guile-gnome guile-gtk guile-ncurses guile-rpc gurgle gv gvpe gxmessage gzip halifax health hello help2man hp2xx httptunnel hurd hyperbole icecat idutils ignuit indent inetutils intlfonts jacal java-getopt jdresolve jel jwhois kawa kopi leg less libc libcdio libextractor libffcall libgcrypt libiconv libidn libmatheval libmicrohttpd libredwg librejs libsigsegv libtasn1 libtool libunistring libxmi lightning lilypond linux-libre liquidwar6 lispintro lrzsz lsh m4 macchanger mailman mailutils

make marst maverik mc mcron mcsim mdk mediagoblin melting metaexchange metahtml mifluz mig miscfiles mit-scheme moe motti mpc mpfr mtools myserver nana nano ncurses nettle network ocrad octave oleo orgadoc osip packaging panorama paperclips parallel parted pascal patch paxutils pcb pdf pem pexec pgccfd phantom-home phpgroupware pies pipo plotutils polyxmass powerguru proxyknife pspp psychosynth pth pythonwebkit qexo quickthreads radius rcs readline recutils reftex rottlog rpge rush sather scm screen sed serveez sharutils shishi shmm shtool sipwitch slib smalltalk smarteiffel snakecharmer social solfege sourceinstall spacechart speex spell sqltutor src-highlite stalkerfs stow stump superopt swbis sysutils talkfilters tar termcap termutils teseq teximpatient texinfo texmacs thales time tramp trans-coord trueprint units unrtf userv uucp vc-changelog vc-dwim vcdimager vera vmgen vmslib w3 wb wdiff websocket4j webstump wget which womb xaos xboard xhippo xlogmaster xmlat xnee xorriso zile

On 25 August 1991, newsgroup comp.os.minix

Hello everybody out there using minix.
I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones. This has been brewing since april, and is starting to get ready. I'd like any feedback on things people like/dislike in minix, as my OS resembles it somewhat (same physical layout of the file-system (due to practical reasons) among other things).
I've currently ported bash(1.08) and gcc(1.40), and things seem to work. This implies that I'll get something practical within a few months, and I'd like to know what features most people would want. Any suggestions are welcome, but I won't promise I'll implement them :-)
Linus (torvalds@kruuna.helsinki.fi)
PS. Yes – it's free of any minix code, and it has a multi-threaded fs. It is NOT portable (uses 386 task switching etc), and it probably never will support anything other than AT-harddisks, as that's all I have :-(. —Linus Torvalds

monolithique vs micro-noyau

Le système minix est système minimaliste créé en 1987 par Andrew Tannenbaum pour illustrer son fameux cours sur les systèmes d'exploitation.

- *To me, writing a monolithic system in 1991 is a truly poor idea.*

monolithique vs micro-noyau

Le système minix est système minimaliste créé en 1987 par Andrew Tannenbaum pour illustrer son fameux cours sur les systèmes d'exploitation.

- *To me, writing a monolithic system in 1991 is a truly poor idea.*
- *Je persiste à penser que concevoir un noyau monolithique en 1991 est une erreur fondamentale. Estime-toi heureux de ne pas être un de mes étudiants. Tu n'obtiendrais pas une bonne note pour une telle conception :-)*

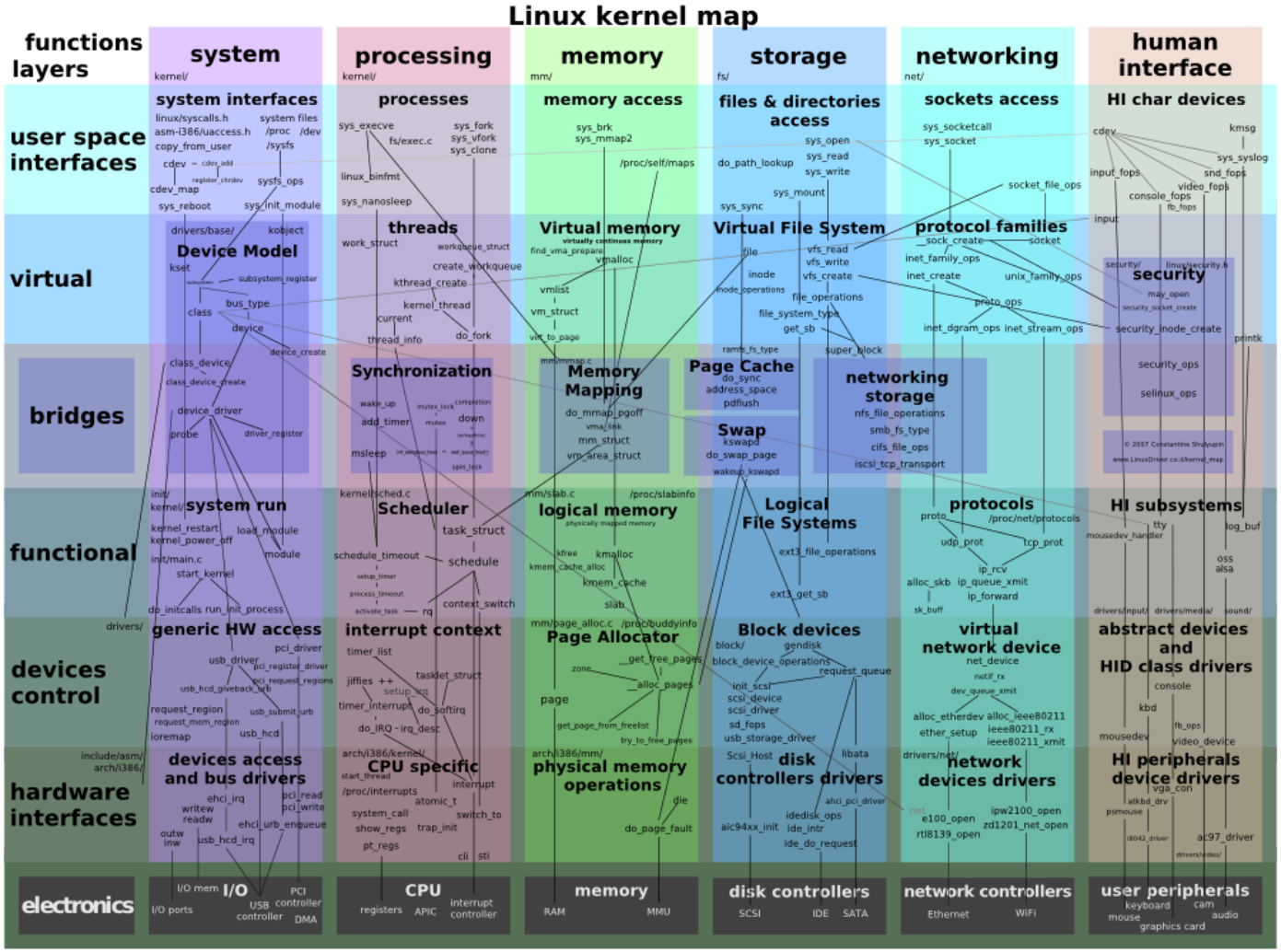
monolithique vs micro-noyau

Le système minix est système minimaliste créé en 1987 par Andrew Tannenbaum pour illustrer son fameux cours sur les systèmes d'exploitation.

- *To me, writing a monolithic system in 1991 is a truly poor idea.*
- *Je persiste à penser que concevoir un noyau monolithique en 1991 est une erreur fondamentale. Estime-toi heureux de ne pas être un de mes étudiants. Tu n'obtiendrais pas une bonne note pour une telle conception :-)*

linux >2 est un noyau monolithique modulaire.

noyau linux


Linux kernel map
functions
layers
system
processing
memory
storage
networking
human interface
user space interfaces
virtual
bridges
functional
devices control
hardware interfaces
electronics
system interfaces
processes
memory access
files & directories access
sockets access
HI char devices
Device Model
threads
Virtual memory
Virtual File System
protocol families
security
Synchronization
Memory Mapping
Page Cache
Swap
networking storage
system run
Scheduler
logical memory
Logical File Systems
protocols
HI subsystems
generic HW access
interrupt context
Page Allocator
Block devices
virtual network device
abstract devices and HID class drivers
devices access and bus drivers
CPU specific
physical memory operations
disk controllers drivers
network devices drivers
HI peripherals device drivers
I/O
CPU
memory
disk controllers
network controllers
user peripherals

distribution

Le système GNU/linux est mis en forme au sein de plusieurs distributions qui intègrent le noyau et les utilitaires avec

- une politique de distribution,
- un système de maintenance,
- une communauté

```
⇝ uname -m -o -r -s
Linux 2.6.34.9-69.fc13.i686 i686 GNU/Linux

⇝ ssh pl@192.168.0.150 uname -mors
Linux 3.8.13-100.fc17.i686.PAE i686 GNU/Linux

⇝ ssh pl@imath01
Linux 2.6.18-348.4.1.el5.centos.plus x86_64 GNU/
   Linux
```

uname –help (man uname)

```
⇝ uname --help
Utilisation: uname [OPTION]
Affiche certaines informations systeme
  -a, --all
  -s, --kernel-name          nom du noyau
  -n, --nodename             hostname
  -r, --kernel-release       version du noyau
  -v, --kernel-version       version du kernel
  -m, --machine              materiel
  -p, --processor            type de processeur
  -i, --hardware-platform    plate-forme
  -o, --operating-system     systeme
```

distribution

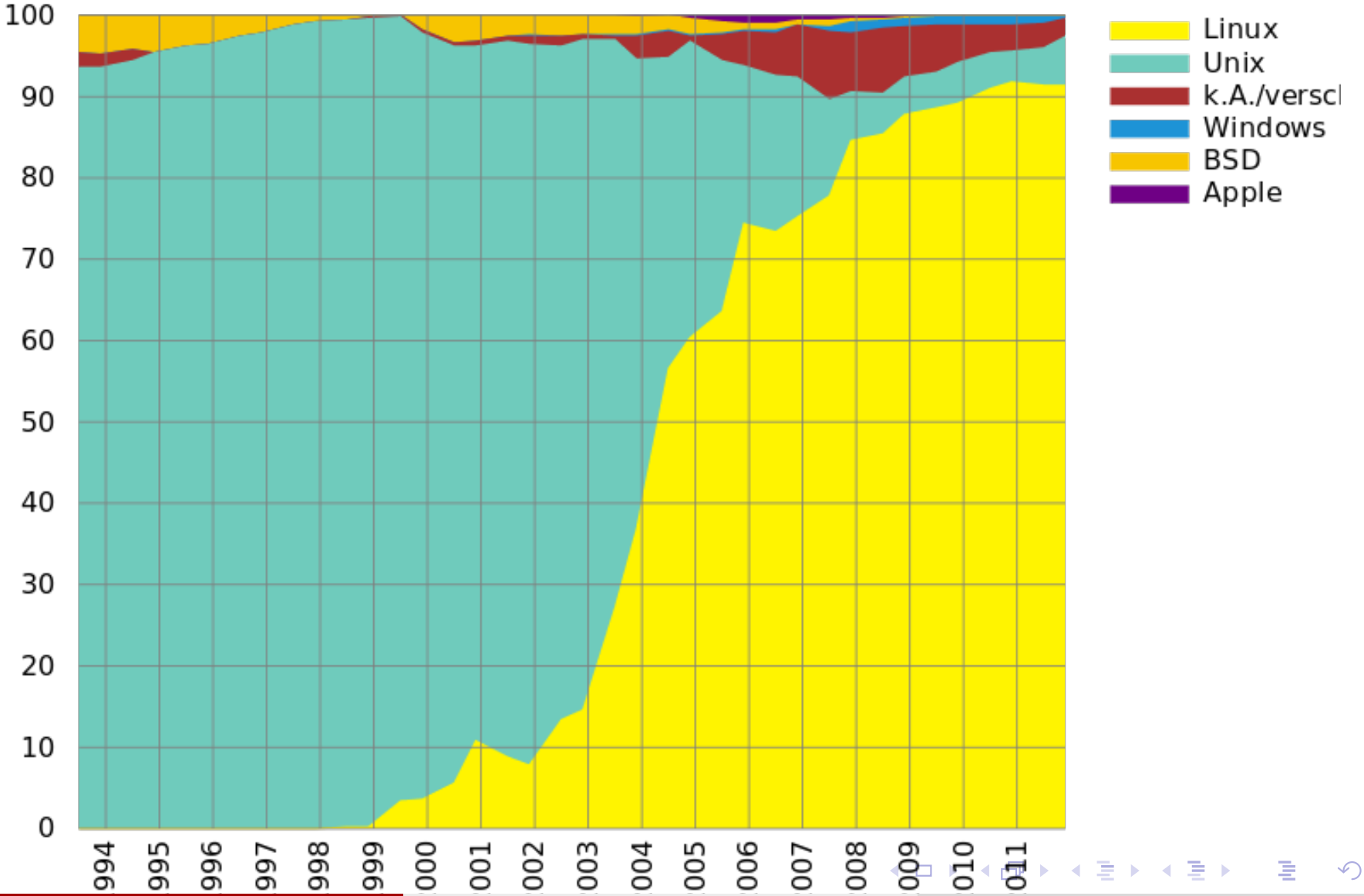
distribution

position

D'aprés la linux foundation, en juin 2012, 98% des supercalculateurs du top500 sont essentiellement équipés d'un système `linux` :

	2012
`linux`	92.4
`unix`	5
`windows`	0.4
`BSD`	0.2

systèmes des supercalculateurs



position

- En 2013, United Space Alliance remplace les `windows` et `scientific linux` des ordinateurs de la station spatiale ISS par une distribution `debian` de `GNU/linux`.

position

- En 2013, United Space Alliance remplace les `windows` et `scientific linux` des ordinateurs de la station spatiale ISS par une distribution `debian` de `GNU/linux`.
- La plupart des serveurs du réseau internet tournent sous un système `unix`.

position

- En 2013, United Space Alliance remplace les `windows` et `scientific linux` des ordinateurs de la station spatiale ISS par une distribution `debian` de `GNU/linux`.
- La plupart des serveurs du réseau internet tournent sous un système `unix`.
- matériel embarqué, matériel réseau

position

- En 2013, United Space Alliance remplace les `windows` et `scientific linux` des ordinateurs de la station spatiale ISS par une distribution `debian` de `GNU/linux`.
- La plupart des serveurs du réseau internet tournent sous un système `unix`.
- matériel embarqué, matériel réseau
- tablette, téléphone

shell ?

La terminologie de la communauté `unix` comprend quelques bizarreries, sigles et acronymes plus ou moins célèbres :

- Portable Operating System Interface X
- foo, bar ? ?
- biff
- shell ? bash ! shabang ♯ !

[ESR's jargon file]
[RFC-3092] International Engineering Task Force.
[FAQ] : `unix` Frequently Asked Questions.

cowsay -W30 'it is easier...

```
 ______________________________
/ it is easier to port a \
| shell than a shell      |
\ script (Larry Wall)     /
 ------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

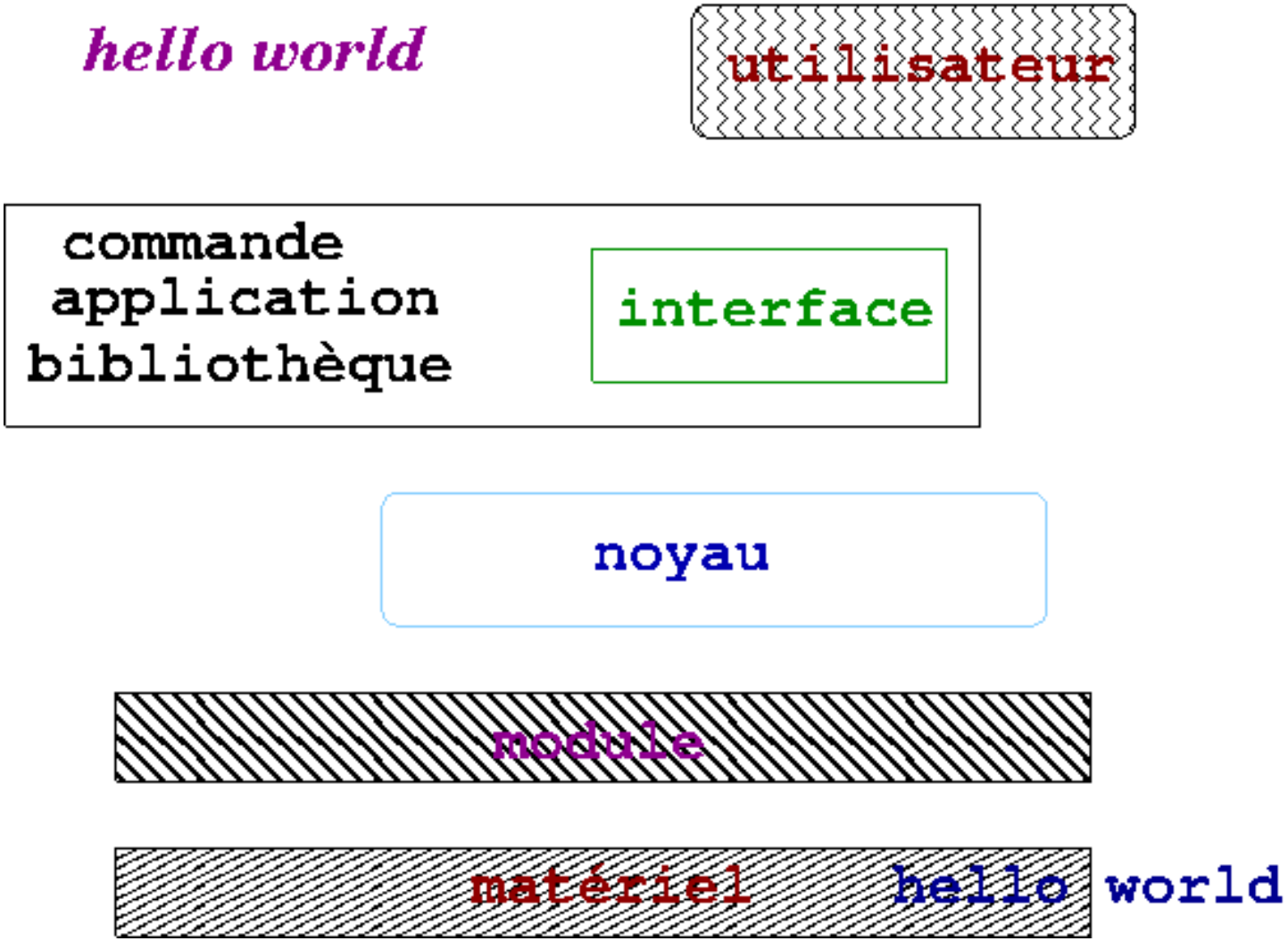
La déclaration de Wall fait référence au partage d'un shell `unix` sur un système `windows`. Le hacker est l'inventeur d'un langage de manipulation de fichiers textes. Lequel ?

cowsay -W30 'it is easier...

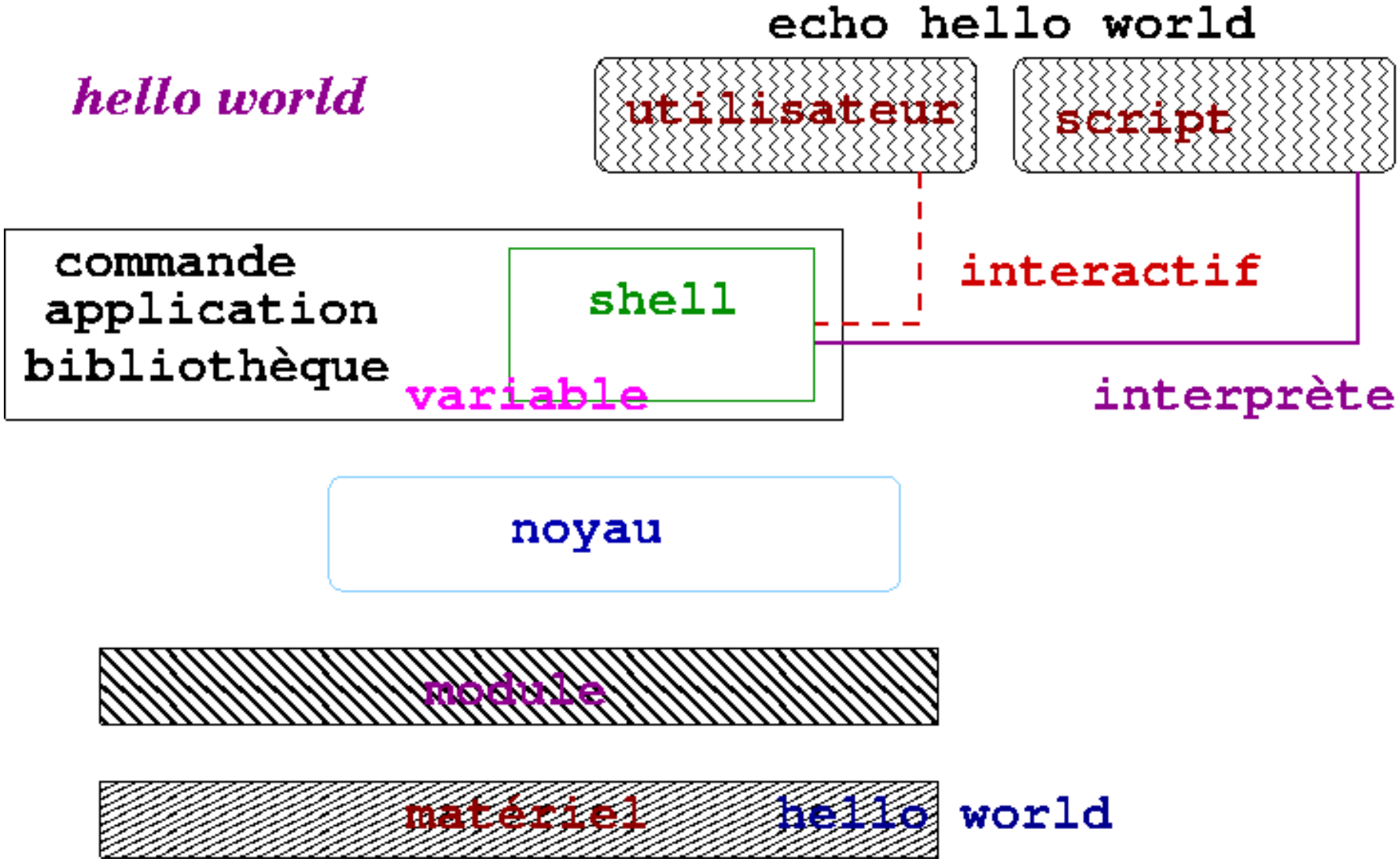
```
 ______________________________
/ it is easier to port a \
| shell than a shell      |
\ script (Larry Wall)     /
 ------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

La déclaration de Wall fait référence au partage d'un shell `unix` sur un système `windows`. Le hacker est l'inventeur d'un langage de manipulation de fichiers textes. Lequel ?

système d'exploitation



système d'exploitation



Thompson shell

La notion de **shell** apparaît dans le système MULTICS, il s'agit d'une application à l'interface entre le noyau et les utilitaires. Le shell développé au début des années 70 par Thompson est un interprète de commandes qui introduit la syntaxe des deux opérateurs fondamentaux des systèmes `unix`, la *redirection* :

$$\texttt{commande} > \textit{destination} < \textit{source}$$

et le *pipeline* de Douglas McIlroy

$$\texttt{commande} \mid \texttt{commande}$$

[man 1 sh]

shell unix

Un *shell unix* est une commande qui permet d'accéder aux fonctionnalités du système d'exploitation :

- utilisation des fichiers et commandes externes
- contrôle des processus

Un *shell* est un langage de programmation qui traitent des variables : globabe, locale et exportée au moyen de commande interne. Un *processus shell* possède un des modes d'exécutions :

- login (initialisation)
- interactif (ligne de commande)
- interpréte (script)

traçage des appels

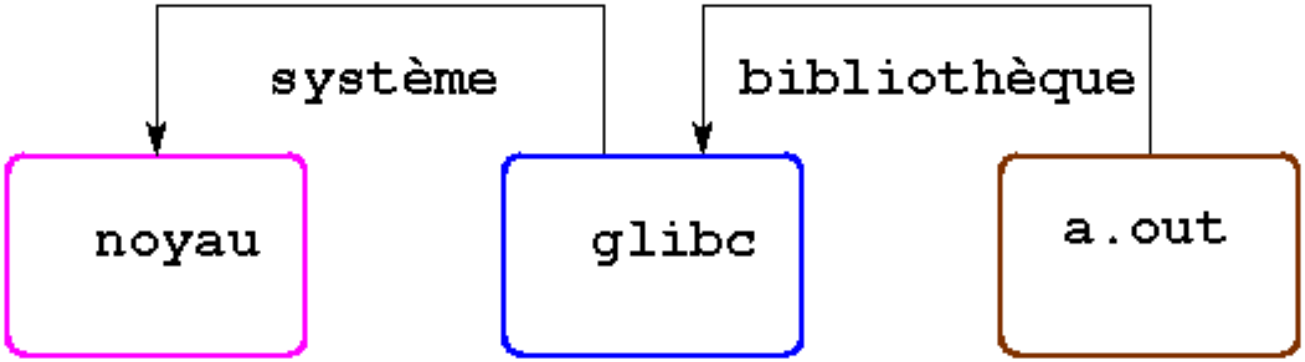
Les débogueurs `ltrace` et `strace` renseignent sur les appels bibliothèques et systèmes effectués par un processus :

```
1 #include <stdio.h>
2 int main( void )
3 {
4 puts("hello world");
5 return 0;
6 }
```

traçage des appels

Les débogueurs `ltrace` et `strace` renseignent sur les appels bibliothèques et systèmes effectués par un processus :

```
1 #include <stdio.h>
2 int main( void )
3 {
4 puts("hello world");
5 return 0;
6 }
```



puts("hello world")

```
1 #!/bin/bash
2 gcc -Wall hw.c
3 ltrace ./a.out |& cut -c1-50 > hw.out
4 strace -e trace=read,write,file ./a.out\
5                 |& cut -c1-50 >> hw.out
```

puts("hello world")

```
__libc_start_main(0x80483b4, 2, 0xbfa92db4, 0x8048
puts("hello world")                                =
hello world
+++ exited (status 0) +++
execve("./a.out", ["./a.out"], [/* 52 vars */]) =
access("/etc/ld.so.preload", R_OK)      = -1 ENOEN
open("/etc/ld.so.cache", O_RDONLY)      = 3
open("/lib/libc.so.6", O_RDONLY)        = 3
read(3, "\177ELF\1\1\1\3\0\0\0\0\0\0\0\0\3\0\3\0\1
write(1, "hello world\n", 12hello world
)           = 12
```

echo hello world

Un exemple plus complexe

```
1 #/bin/bash
2 ltrace    echo hello world |& \
3   cut -c1-50 >hello-lib.out
4
5 strace -e trace=open,read,write,file \
6     echo hello world  |& \
7       cut -c1-50 >hello-sys.out
```

echo hello world

```
__libc_start_main(0x8049070, 3, 0xbfa69174, 0x804b
getenv("POSIXLY_CORRECT")                          =
strrchr("echo", '/')                               =
setlocale(6, "")                                   =
bindtextdomain("coreutils", "/usr/share/locale") =
textdomain("coreutils")                            =
__cxa_atexit(0x804a000, 0, 0, 0x804e5b8, 0xbfa690c
fputs_unlocked(0xbfa695ad, 0xc5e4e0, 0xc5c1e0, 0,
fputs_unlocked(0xbfa695b3, 0xc5e4e0, 0xc5c1e0, 0,
exit(0 <unfinished ...>
__fpending(0xc5e4e0, 0xc5e4e0, 0xb42a24, 0xbfa68fd
fclose(0xc5e4e0hello world
)                                          = 0
__fpending(0xc5e580, 0xc5e4e0, 0xb42a24, 0xbfa68fd
fclose(0xc5e580)                                   =
+++ exited (status 0) +++
```

echo hello world

```
execve("/bin/echo", ["echo", "hello", "world"], [/
access("/etc/ld.so.preload", R_OK)      = -1 ENOEN
open("/etc/ld.so.cache", O_RDONLY)      = 3
open("/lib/libc.so.6", O_RDONLY)        = 3
read(3, "\177ELF\1\1\1\3\0\0\0\0\0\0\0\0\3\0\3\0\1
open("/usr/lib/locale/locale-archive", O_RDONLY|O_
write(1, "hello world\n", 12hello world
)           = 12
```

Bourne [Again] shell

Le Bourne shell est introduit dans le système UNIX V7. Il a été développé par Stephen Bourne avec l'objectif de créer un outil de scriptage des commandes pour faciliter l'administration du système. `sh` est devenu populaire pour sa facilité d'emploi et sa rapidité, standard des systèmes UNIX, c'est toujours le shell par défaut du compte **root**, de certaines commandes `make`.

Bourne [Again] shell

Le Bourne shell est introduit dans le système UNIX V7. Il a été développé par Stephen Bourne avec l'objectif de créer un outil de scriptage des commandes pour faciliter l'administration du système. `sh` est devenu populaire pour sa facilité d'emploi et sa rapidité, standard des systèmes UNIX, c'est toujours le shell par défaut du compte **root**, de certaines commandes `make`.

Le shell de base `sh`

Les `alias`, l'historique des commandes et le contrôle des processus ne sont pas gérés par `sh`.

Le Bourne Again shell est un projet GNU `bash` démarré en 1980 par Brian Fox, actuellement maintenu par Chet Ramey.

shell par défaut

```
⇝ /bin/sh
sh-4.1$ exit
exit

⇝ which sh
/bin/sh

⇝ ls -l /bin/sh
lrwxrwxrwx. 1 root root 4 9 aout 08:47 /bin/sh
   -> bash

⇝ echo $SHELL
/bin/bash
```

shell par défaut

```
⇝ echo $SHELL
/bin/bash

⇝ echo -e 'all:\n\techo $(SHELL)' > /tmp/
    makefile

⇝ make -c /tmp
make: entrant dans le repertoire /tmp
echo /bin/sh
/bin/sh
make: quittant le repertoire /tmp
```

shell populaires

Les shells font légion.

shell	shabang
sh	6 490 000
bash	5 380 000
ash	345 000
ksh	323 000
csh	254 000
tcsh	116 000
zsh	86 000

TABLE – popularité de quelques shells

Ces slides concernent mon shell préféré bash...

Aprés le lambis bien entendu !

[comparatif des shells]

Interfaces

L'utilisateur d'un ordinateur interagit avec le système d'exploitation au moyen d'une interface graphique ou bien d'une interface texte.

- Graphic User Interface
- Command Line Interface

La notion de `shell` est étendue à l'ensemble des interfaces y compris graphique :`gnome`, `kde`, `lxde`, `xfce`, `awesome` ...

- L'évolution des GUIs fait souvent débat !
- Le monde des CLIs parait plus zen...

suggestion : essayez les interfaces !

CLI vs GUI

- Le mode graphique se veut intuitif et facile d'emploi.
- Le mode textuel est puissant mais peu intuitif.

Il n'y a pas lieu d'opposer les deux modes qui sont parfaitement complémentaires, les tâches sur les fichiers textes sont plus faciles à réaliser par la console.
En général, lors d'une session graphique, les utilisateurs ouvrent des pseudos terminaux pour effectuer certaines tâches avec l'interprète de commande.

C-production

Evaluer une C-production

combien de lignes de codes C dans le répertoire **./CC** ?

C-production

Evaluer une C-production

combien de lignes de codes C dans le répertoire **./CC** ?

Pas de solution graphique !

Difficile d'interpréter cette question par des mouvements de souris...

C-production

Evaluer une C-production

combien de lignes de codes C dans le répertoire **./CC** ?

Pas de solution graphique !

Difficile d'interpréter cette question par des mouvements de souris...

Plusieurs solutions textuelles

Au fil des années, les applications `unix` ont été développées, améliorées pour répondre efficacement ce type de questions.

Une ligne de commande

```
$ find ~/CC -name "*.c" | xargs cat | wc -l

400942
```

Une ligne de commande

```
$ find ~/CC -name "*.c" | xargs cat | wc -l

400942
```

Une solution facile à comprendre plus difficile à reproduire sans connaître les usages des commandes et arguments passés par la ligne de commande.

- `find` : outil pour la recherche de fichiers.
- `wc` : compter les lignes, mots, octets.

  Paul Rubin et David MacKenzie
- `xargs` : construire et exécuter des lignes de commandes.

Une ligne de commande

```
$ find ~/CC -name "*.c" | xargs cat | wc -l

400942
```

Une solution facile à comprendre plus difficile à reproduire sans connaître les usages des commandes et arguments passés par la ligne de commande.

- `find` : outil pour la recherche de fichiers.
- `wc` : compter les lignes, mots, octets.

  Paul Rubin et David MacKenzie
- `xargs` : construire et exécuter des lignes de commandes.

```
$ cat $(find ~/CC -name '*.c' ) 2>/dev/null | wc
    -l

400942
```

Une autre solution

Une solution basée sur le filtre `awk` ?

Une autre solution

Une solution basée sur le filtre `awk` ?

```
$ find ~/CC -name "*.c" -exec wc -l {} \;
      | awk 'BEGIN {s=0} {s=s+$0} END {print s}'

400942
```

Une autre solution

Une solution basée sur le filtre `awk` ?

```
$ find ~/CC -name "*.c" -exec wc -l {} \;
      | awk 'BEGIN {s=0} {s=s+$0} END {print s}'

400942
```

- `find` : outil pour la recherche de fichiers.
- `wc` : compter les lignes, mots, octets.
- `awk` : commande d' Alfred Aho, Peter Weinberger et Brian Kernighan pour filtrer les lignes d'un fichier texte.

script

Le rôle d'un informaticien est de réaliser des travaux avec une machine, le plus souvent, dans un temps limité :-).
Il convient déviter des pertes de temps qui ont pour origine

- erreurs
- lacunes

Un *script* est un fichier de commandes pour le shell. La première ligne du script peut préciser le shell d'exécution :

```
1 #!/bin/myshell
2 ...
```

c'est le fameux **shabang**.

exemple

```
1 #!/bin/bash
2 sum=0
3 while read num rem
4    do
5    echo $num $rem
6    let   sum+=$num
7    done < <(find $1 -name $2 -exec wc -l {} \;)
8 echo $sum
```

Aprés avoir sauvé ces lignes dans un fichier `count.sh` :

```
⇝ chmod u+x count.sh
```

pour rendre le script exécutable, on peut alors lancer

```
⇝ ./count.sh ~/CC '*.c'
```

commande complexe

Avec de la pratique, il est possible de d'exécuter une tache complexe directement en ligne de commande :

```
⇝ find ~ -name "*.txt" -exec wc -l {} \;
  | ( sum=0; while read num rem ;
  do let sum+=$num;  done; echo $sum) > sum.txt
```

Il s'agit d'une *commande composée* s'articulant sur des *commandes simples* incluant un *pipeline* et une *redirection*.

3 - manuel

- man help
- navigation
- section
- piège
- a propos
- exercice
- format

documentation

On peut faire beaucoup de choses en mode texte :
commande [option-courte] [option–longue] [argument]

- filtrer : `grep`, `sed`, `find`
- compiler : `make`, `gcc`
- calculer : `bc`, `gmp`
- jouer : `gnuchess`
- naviguer : `lynx`, `links`
- etc. . .

Le manuel **unix** est un moyen efficace pour découvrir et retrouver les *options* des commandes populaires, les *prototypes* des bibliothèques, les *paramètres* de configuration.

pages du manuel

Il existe de nombreuses ressources pour obtenir de l'aide sur les commandes : livre, tutoriel, forum, moteur de recherche, howto, wiki...

`man` renseigne sur les commandes externes.

`help` renseigne sur les commandes internes.

Avant d'aller plus loin, rappelons comment sortir d'un mauvais pas sur la ligne de commande :

- CTRL-c : tuer la commande en cours
- CTRL-d : fermer l'entrée standard
- CTRL-z : stopper la commande
- CTRL-u : nettoyer la ligne de commande
- CTRL-r : retrouver une commande
- `readline`, `bind`, *.inputrc*

man page

Les composants `unix` sont documentés de manière concise sous forme de *man page*. Une information exploitable en ligne de commande par la commande `man`.

Raccourci	Action	Parm.
h	aide	
q	quitter	
flèches	navigation	
SPACE , b	changer de page	suivante, précedente
/ ?	rechercher	avant/arrière
n, N	occurrence	suivante, précédente

TABLE – commandes de `less`

La sortie du manuel est au format roff : un langage à balises léger et, par défaut, elle est interprée par la commande less.

bien commencer : man man ...

```
⇝ man man | wc -l
616

⇝man man | col -b | grep -A2 AUTEUR
AUTEUR
       John W. Eaton est l'auteur historique
de man. Federico Lucifredi <flucifredi@acm.org>
en assure aujourd'hui la maintenance.
```

bien commencer : man man ...

```
⇝ man man | col -b | grep BUGS -C4

A manual page consists of several sections.

Conventional section names include NAME,
SYNOPSIS, CONFIGURATION, DESCRIPTION,
OPTIONS, EXIT STATUS, RETURN VALUE, ERRORS,
ENVIRONMENT, FILES, VERSIONS, CONFORMING TO,
NOTES, BUGS, EXAMPLE, AUTHORS, and SEE ALSO.
```

section du manuel

1. commandes internes et externes.
2. appels système.
3. bibliothèque.
4. fichiers spéciaux.
5. formats des fichiers et conventions.
6. jeux.
7. divers (y compris les macropaquets et les conventions).
8. gestion du système.
9. Interface du noyau Linux.

man 1 intro

```
INTRO(1) Manuel de l'utilisateur Linux

NOM
        intro — Introduction aux commandes
    utilisateur

DESCRIPTION
        La section 1 du manuel decrit les commandes
et outils de l'utilisateur, comme les utilitaires
de manipulation de fichiers, les interpreteurs de
commandes, les compilateurs, les navigateurs web,
les editeurs et outils de visualisation de fichiers
et d'images, etc...
```

Exemple de boucle

```
⇝ for x in {1..8}; do man $x intro | grep intro
  ; done

intro - Introduction aux commandes utilisateur
intro - Introduction a la section des appels
  systeme
intro - Introduction aux fonctions de
  bibliotheque
intro - Introduction aux fichiers speciaux.
intro - Introduction a la section Formats de
  fichiers
intro - Introduction aux jeux
intro - Introduction a la section panoramas,
  conventions,  et
intro - Introduction aux commandes d'
```

puzzle

```
1 char *p="char *p=%c%s%c;
2 main(){printf (p,34, p,34);} ";
3 main(){printf(p,34, p,34);}
```

quine.c

puzzle

```
1 char *p="char *p=%c%s%c;
2 main(){printf (p,34, p,34);} ";
3 main(){printf(p,34, p,34);}
```

quine.c

```
⇝ tr '\n' ' ' < quine.c > q.c
⇝ cat q.c
```

puzzle

```
1 char *p="char *p=%c%s%c;
2 main(){printf (p,34, p,34);} ";
3 main(){printf(p,34, p,34);}
```

quine.c

```
⇝ tr '\n' ' ' < quine.c > q.c
⇝ cat q.c
```

```
char *p="char *p=%c%s%c; main(){printf(p,34,p
    ,34);}"; main(){printf(p,34,p,34);}
```

puzzle

```
1 char *p="char *p=%c%s%c;
2 main(){printf (p,34, p,34);} ";
3 main(){printf(p,34, p,34);}
```

quine.c

```
⇝ tr '\n' ' ' < quine.c > q.c
⇝ cat q.c
```

```
char *p="char *p=%c%s%c; main(){printf(p,34,p
    ,34);}"; main(){printf(p,34,p,34);}
```

```
⇝ gcc q.c
./a.out
char *p="char *p=%c%s%c; main(){printf(p,34,p
    ,34);}"; main(){printf(p,34,p,34);}
```

hint

Trois consultations du manuel permettent de comprendre ce qui se passe :

- `man tr` pour obtenir des détails sur le filtre `tr`.
- `man printf` pour le mode d'emploi de la fonction `printf`.
- `man ascii` pour un mémo sur le codage des caractères.

man ascii

```
⇝ man ascii | grep '30 40' -A12 | cut -c26-

   30 40 50 60 70 80 90 100 110 120
  ------------------------------------
 0:     ( 2 < F P Z d   n   x
 1:     ) 3 = G Q [ e   o   y
 2:     * 4 > H R \ f   p   z
 3:  !  + 5 ? I S ] g   q   {
 4:  "  , 6 @ J T ^ h   r   |
 5:  #  - 7 A K U _ i   s   }
 6:  $  . 8 B L V ` j   t   ~
 7:  %  / 9 C M W a k   u   DEL
 8:  &  0 : D N X b l   v
 9:  '  1 ; E O Y c m   w
```

printf du C??

```
⇝ man printf
```

```
PRINTF(1)                                User Commands
NAME
       printf - format and print data

SYNOPSIS
       printf FORMAT [ARGUMENT]...
       printf OPTION

DESCRIPTION
       Print ARGUMENT(s) according to FORMAT, or
   execute according to OPTION:

       --help display this help and exit
```

[printf dans le web]

printf du C !

```
⇝ man 3 printf
```

```
PRINTF(3)                                Linux Programmer's
    Manual

NAME
        printf, fprintf, sprintf, snprintf, vprintf,
        vfprintf,  vsprintf, vsnprintf — formatted
    output conversion

SYNOPSIS
        #include <stdio.h>

        int printf(const char *format, ...);
```

exercice

Utiliser le manuel pour expliquer :

```
⇝ man man | col -b | grep -E '^[A-Z]+$'
NAME
SYNOPSIS
DESCRIPTION
EXAMPLES
OVERVIEW
DEFAULTS
OPTIONS
ENVIRONMENT
FILES
HISTORY
```

```
⇝ man man | col -b | grep -Ex '[A-Z]+'
```

```
⇝ man man | grep EXAMPLES -A24
EXAMPLES
 man ls
     Display the manual page for the item  ls.
 man -a intro
     Display, in succession, all of the available

 man -k printf
     Search the short descriptions and manual
    page
     names for the keyword printf as  regular
     expression.  Equivalent to apropos -r printf
    .
 man -f smail
     Lookup  the manual pages referenced by smail
     and print out the short descriptions of any
     found.  Equivalent to whatis -r smail.
```

man -k RE

```
⇝ man -k '[a-z]{3}printf' | cut -c1-40
vasprintf (3)          - print to allocate
vfwprintf (3)          - formatted wide-ch
vfwprintf (3p)         - wide-character fo
vsnprintf (3)          - formatted output
vsnprintf (3p)         - format output of
vswprintf (3)          - formatted wide-ch
vswprintf (3p)         - wide-character fo
```

man -k RE

```
⇝ man -k '[a-z]{3}printf' | cut -c1-40
vasprintf (3)          - print to allocate
vfwprintf (3)          - formatted wide-ch
vfwprintf (3p)         - wide-character fo
vsnprintf (3)          - formatted output
vsnprintf (3p)         - format output of
vswprintf (3)          - formatted wide-ch
vswprintf (3p)         - wide-character fo
```

```
⇝ man -k '[a-z]{3}printf' | sed 's/-.*/@/'
vasprintf (3)          @
vfwprintf (3)          @
vfwprintf (3p)         @
vsnprintf (3)          @
vsnprintf (3p)         @
```

printf

```
⇝ man -f printf
printf (3p)     - print formatted output
printf (3)      - formatted output conversion
printf (1)      - format and print data
printf (1p)     - write formatted output
```

- `whatis printf`

Il y a donc

- une fonction `printf`
- une commande `printf`

attention !

```
⇝ /bin/printf
/bin/printf: operande manquant
Saisissez /bin/printf --help pour plus
                        d'informations.

⇝ cd /bin

⇝ printf
printf: utilisation:printf [-v var]
                        format [args]

⇝ ./printf
./printf: operande manquant
Saisissez ./printf --help pour plus
                        d'informations
```

piege classique

Les commandes homonymes font que le manuel ne renseigne pas toujours sur la commande que l'on croit ! ! !

echo

```
⇝ man echo
```

```
ECHO(1)   User Commands
NAME
echo – display a line of text
SYNOPSIS
echo [SHORT–OPTION]... [STRING]...
echo LONG–OPTION
DESCRIPTION
    Echo the STRING(s) to standard output.
   –n do not output the trailing newline
   –e enable interpretation of backslash
   –E disable interpretation of backslash
   ––help display this help and exit
    ...
```

help

```
⇝ echo --help
--help

⇝ which echo
/bin/echo

⇝ /bin/echo --help
Utilisation : /bin/echo [SHORT-OPTION]...
          ou : /bin/echo LONG-OPTION
...

⇝ type -f echo
echo est une primitive du shell
```

help

```
⇝ help echo

echo: echo [-neE] [arg ...]
    ecrit les arguments sur la sortie standard.

    Affiche les ARGs sur la sortie standard,
                suivis d'un retour a la ligne.

Options :
  -n     ne pas ajouter de saut de ligne
  -e active l'interpretation des barres
         contre-obliques d'echappement ci-dessous
  -E supprime explicitement l'interpretation
         les barres contre-obliques d'echappement
```

enable

```
⇝ echo --help
--help

⇝ enable -n echo

⇝  echo --help
Utilisation : echo [SHORT-OPTION [STRING]
          ou : echo LONG-OPTION
```

enable, head, tail

```
⇝ enable | head -7
enable .
enable :
enable [
enable alias
enable bg
enable bind
enable break
```

enable, head, tail

```
⇝ enable | head -7
enable .
enable :
enable [
enable alias
enable bg
enable bind
enable break
```

```
⇝ enable | tail -3
enable unalias
enable unset
enable wait
```

liste des commandes internes

```
⇝ enable|sed 's/^[^ ]*//'|tr '\n' ' '|fold -sw50
. : [ alias bg bind break builtin caller cd
command compgen complete compopt continue
   declare
dirs disown echo enable eval exec exit export
false fc fg getopts hash help history jobs kill
let local logout mapfile popd printf pushd pwd
read readarray readonly return set shift shopt
source suspend test times trap true type typeset
```

liste des commandes internes

```
⇝ enable|sed 's/^[^ ]*//'|tr '\n' ' '|fold -sw50
 .  :  [  alias  bg  bind  break  builtin
   caller
 cd  command  compgen  complete  compopt
continue  declare  dirs  disown  echo  enable
eval  exec  exit  export  false  fc  fg  getopts
hash  help  history  jobs  kill  let  local
logout  mapfile  popd  printf  pushd  pwd  read
readarray  readonly  return  set  shift  shopt
source  suspend  test  times  trap  true  type
typeset  ulimit  umask  unalias  unset  wait
```

exercice

Utiliser l'aide en ligne de commande pour expliquer la différence entre :

```
⇝ echo -E '\retou\rcha\riot'
```

et

```
⇝ echo -e '\retou\rcha\riot'
```

Utiliser le manuel pour comprendre les lignes :

```
⇝ find ~/CC -name "*.c" -exec wc -l {} \; foo \
    | awk 'BEGIN {s=0} {s=s+$0} END {print s}'
```

iliv ?

```
⇝ man iliv
```

iliv ?

```
⇝ man iliv
```

```
ILIV(6)

NAME
        iliv - unix and shell programs

SYNOPSIS
        iliv [-h|-a] [-lse] [-v type] [ list ]

DESCRIPTION
  iliv  provides informations on the introduction
to GNU/linux system and Bash programs course
```

```
.\" exemple de page
.\" cours I54 USTV 2012
.TH ILIV  6 "falling 2012" "I54 Manual"
.SH NAME
iliv \- unix and shell programs
.SH SYNOPSIS
iliv
[\fB-h\fP|\fB-a\fP] [\fB-lse\fP] [\fB-v\fP \fItype\fP] [ list ]
.SH DESCRIPTION
iliv provides informations on the introduction to GNU/linux system and Bash programs courses.
.SH OPTIONS
The following options are supported
.TP
\fB-a\fP \fB-h\fP displays help
.TP
\fB-l\fP lecturer
.TP
\fB-s\fP slides
.TP
\fP-e\fP ecue
.SH LICENCE
Copyleft (!c) 2012 by Pavle Michko.
.SH AUTHOR
michko@euphoria.fr
.SH SEE ALSO
Joseph (kernel) Emmanuel (tcp/ip)
.SH BUG
please, report bugs administrative autority.
```

format ROFF

Les pages du manuels sont écrites dans le langage de formatage **roff**. Un langage à balises plus léger que LaTeX utilisé pour les pages du manuel.

```
⇝ groff -man -Tascii iliv.6
```

si OK alors

```
⇝ cp iliv.6 /usr/share/man/.
```

Pour en savoir plus,

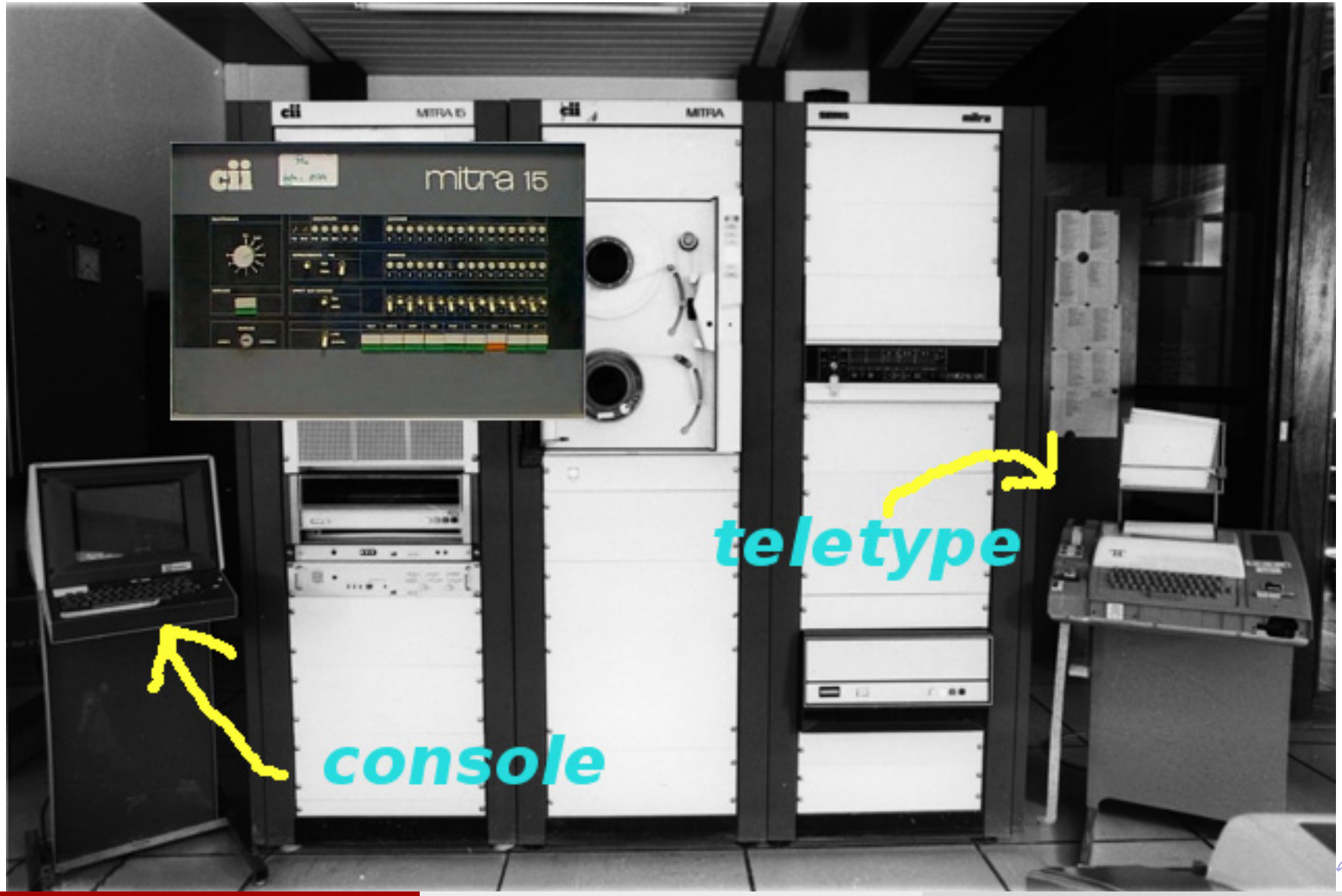
```
⇝ man 7 roff
```

4 - login

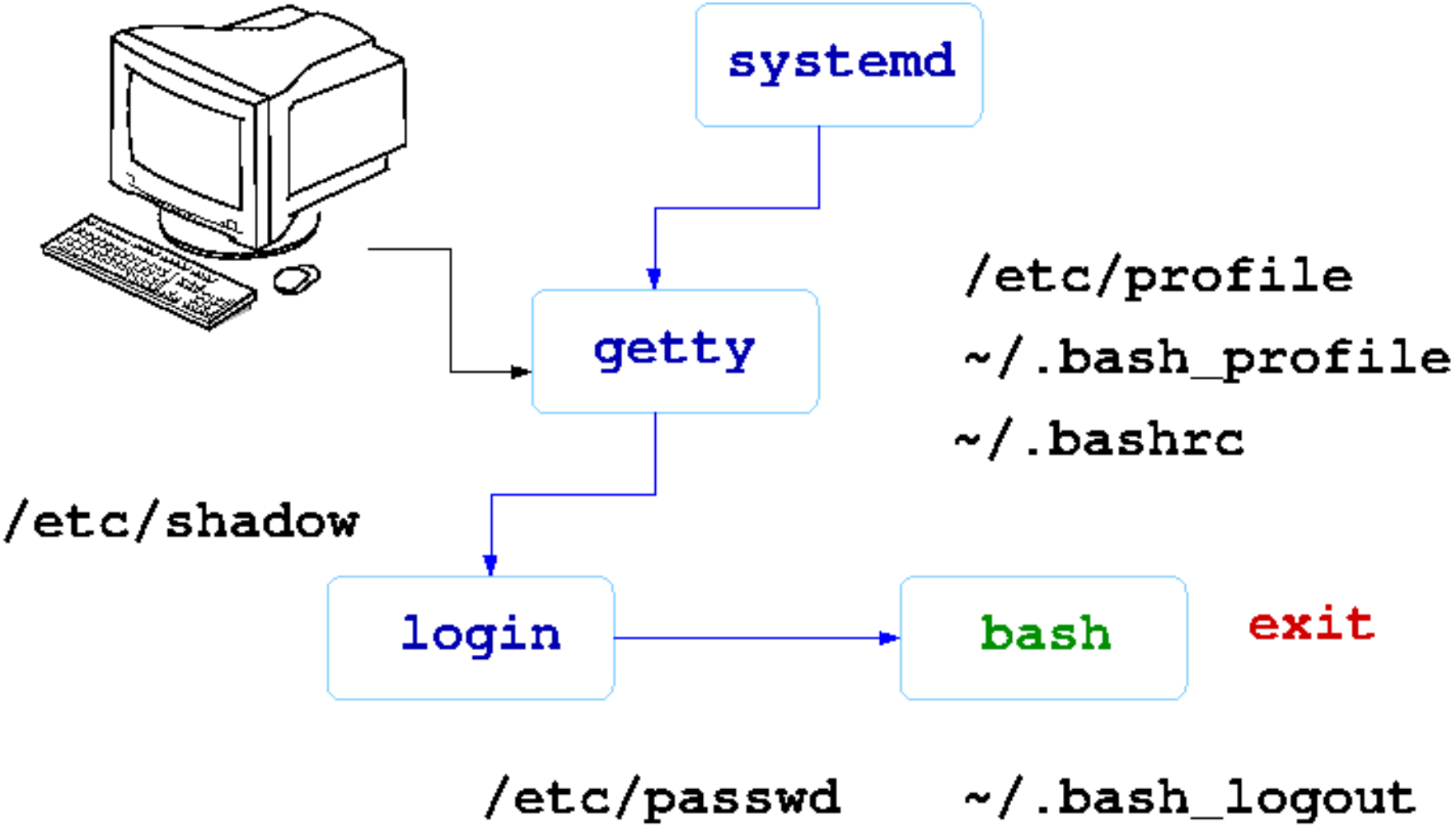
- console
- login
- user, root
- connexion à distance
- tour de l'hôte
- mot de passe

origine

MITRA 15



console réelle



login

`login` initialise les variables :

- HOME, SHELL (/etc/passwd)
- USER, LOGNAME (/etc/passwd)
- PATH (/usr/local/bin :/bin :/usr/bin)
- MAIL (répertoire)

avant l'action du **shell**.

pseudo terminal

```
⇝ tty
/dev/pts/2

⇝ fuser -av  /dev/pts/2
             UTIL.    PID ACCES  COMMANDE
/dev/pts/2:  pl       1567 F....  xfce4-terminal
             pl       3041 F....  bash

⇝ ps -oppid,comm 3041
 PPID COMMAND
 1567 bash

⇝ ps -oppid,comm 1567
 PPID COMMAND
    1 xfce4-terminal

⇝ ps -oppid,comm 1
 PPID COMMAND
    0 systemd

⇝ps -oppid,comm 0
error: process ID out of range
```

root, user

L'administrateur **root** est habilité à créer les comptes utilisateurs **user** à partir de :

- nom de login
- nom complet

Un jeu de commande d'administration permet à **root** de paramétrer les comptes utilisateurs.

- unicité du super-utilisateur pour `unix`.
- `sudo` : permet de lancer une commande **as root**.

création d'un compte

```
# useradd toto -c 'Mr Toto' -g novice
# ls -al /home/toto/
total 28
drwx------. 4 toto novice 4096
drwxr-xr-x. 9 root   root 4096
-rw-r--r--. 1 toto novice   18 .bash_logout
-rw-r--r--. 1 toto novice  193 .bash_profile
-rw-r--r--. 1 toto novice  124 .bashrc
drwxr-xr-x. 2 toto novice 4096 .gnome2
```

- `.bash{_logout,_profile,rc}` configuration du **shell**

```
VOIR AUSSI
  passwd, crypt(3), userdel(8), usermod(8).
```

.bash_profile

```
1  # .bash_profile
2  # Get the aliases and functions
3  if [ -f ~/.bashrc ]; then
4      . ~/.bashrc
5  fi
6  # User specific environment and startup programs
7  export PATH=$PATH:$HOME/bin:$HOME/script
8  export TEXINPUTS=$TEXTINPUTS:$HOME/texinput
9  export HISTFILESIZE=2048
10 export HISTSIZE=512
11 export HISTFILE=~/.compil_history
12 export PS1="\t pl@\W>"
```

.bashrc

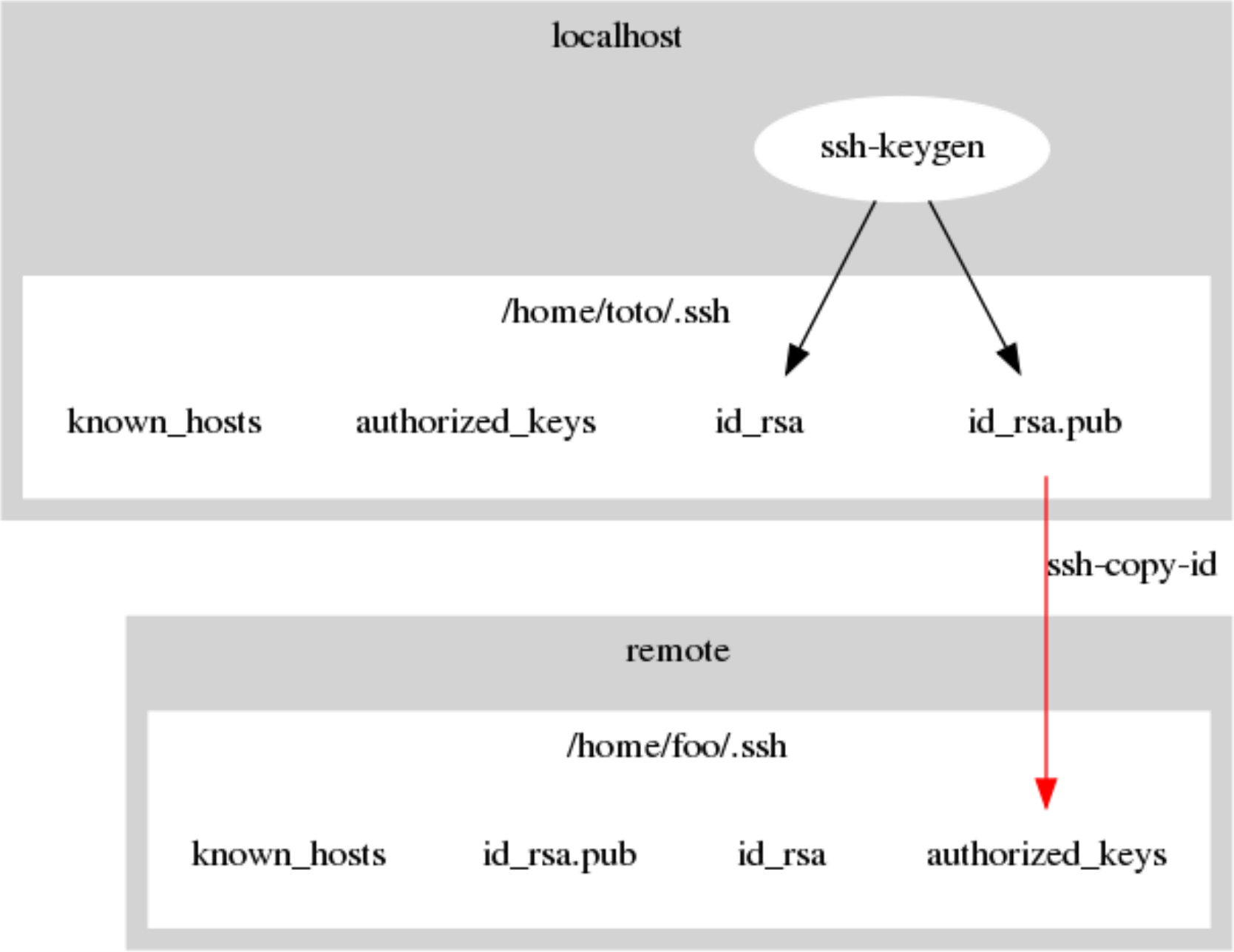
```
# .bashrc
# Source global definitions
if [ -f /etc/bashrc ]; then
 . /etc/bashrc
fi
set -o vi
alias sshnab='ssh langevin@naboo.univ-tln.fr'
alias sshgpu='ssh pl@imath-gpu.univ-tln.fr'
alias sshm1='ssh pl@serveur-imath01.univ-tln.fr'
alias sshm2='ssh pl@serveur-imath02.univ-tln.fr'
# etc...
```

/etc/passwd

```
⇝ ls -ald /home/t*
drwx------. 4 titi novice  /home/titi
drwx------. 4 toto novice  /home/toto
⇝ tail -2 /etc/passwd
toto:x:1002:1002:Mr Toto:/home/toto:/bin/bash
titi:x:1003:1002:Mr Titi:/home/titi:/bin/bash
♯ passwd toto
♯ mail toto@ouest.fr -s compte << -FIN
> salut toto,
> votre compte est active !
> bye
> -FIN
```

configuration ssh

```
[toto@ouest ~]$ ls .ssh/
authorized_keys   known_hosts

[toto@ouest ~]$ ssh-keygen

[toto@ouest ~]$ ls .ssh/
authorized_keys  id_rsa  id_rsa.pub  known_hosts

[toto@ouest ~]$ ssh-copy-id turing.euphoria.fr
```


localhost
ssh-keygen
/home/toto/.ssh
known_hosts
authorized_keys
id_rsa
id_rsa.pub
ssh-copy-id
remote
/home/foo/.ssh
known_hosts
id_rsa.pub
id_rsa
authorized_keys

configuration ssh

```
[toto@ouest ~]$ ssh turing.euphoria.nil

[toto@turing ~]$ pstree -aulpG
init,1
  |-- sshd,4197
  |    |
  |    |-- sshd,27781
  |         |
  |         |-- sshd,27783,toto
  |              |
  |              |-- bash,27784
  |                   |
  |                   |-- pstree,6718 -aulpG
```

commande distante

```
⇝ ssh toto@turing.euphoria.nil
...
[toto@turing ~]$ pstree -aulpG
init,1
  |-- sshd,4197
  |    |
  |    |-- sshd,27781
  |         |
  |         |-- sshd,27783,toto
  |              |
  |              |-- bash,27784
  |                   |
  |                   |-- pstree,6718 -aulpG
```

console virtuelle

```
⇝ ps -C mingetty -opid,cmd
  PID CMD
 1291 /sbin/mingetty /dev/tty2
 1295 /sbin/mingetty /dev/tty4
 2108 /sbin/mingetty /dev/tty3
```

ALT-CTRL-F3, ouvre une **console**.

```
login: pl passwd: ----
[pl] ps -C mingetty
  PID TTY          TIME CMD
 1291 tty2     00:00:00 mingetty
 1295 tty4     00:00:00 mingetty

[pl] tty
/dev/tty3
```

shell de login

```
[pl] bash -c 'ps -f; sleep 5'
UID PID  PPID  TTY CMD
pl  2257 2158 tty3 -bash
pl  2317 2257 tty3 bash -c ps -f; sleep 5
pl  2318 2317 tty3 ps -f
```

- le '-' indique un **shell** de login.

getty

```
[pl] pstree -ap | grep -A7 -e '|-login' > tty.
    txt
[pl] exit
```

ALT-CTRL-F1, ferme la **console**.

```
⇝ cat tty.txt
  |-login,2108
  |     `-bash,2143
  |         |-grep,2176 -A7 -e |-login
  |         `-pstree,2175 -ap
  |-mingetty,1291 /dev/tty2
  |-mingetty,1295 /dev/tty4
```

découverte de l'hôte

```
[toto@turing ~]$ uname -a
Linux turing.euphoria.fr 2.6.18-53.1.4.el5 ...
   i386 GNU/Linux

[toto@turing ~]$ grep -c proc /proc/cpuinfo
4

[toto@turing ~]$ lscpu
Architecture:        i686
CPU op-mode(s):      32-bit
CPU(s):              2 Thread(s) par coeur :  2
```

découverte de l'hôte

```
⇝ uptime
 18:00:55 up 14 days, 22:21,  1 user,
            load average: 1.37, 0.61, 0.33

⇝ free -g
     total   used free shared buffers cached
Mem:    47     30   16      0       0     29
-/+ buffers/cache:    1    46
Swap:          72     0    72
```

accès au réseau

```
[toto@turing ~]$ cat /etc/resolv.conf
search euphoria.fr
nameserver 10.1.65.1
[toto@turing ~]$ /sbin/ifconfig eth0
eth0 Link Ethernet  HWaddr 00:1A:64:34:C3:FA
inet adr:10.1.65.72 Bcast:10.1.255.255
                    Masque:255.255.0.0
adr inet6:fe80: :21a:64ff:fe34:c3fa/64 ...
[toto@turing ~]$ /sbin/route
Table de routage IP du noyau
Destination  Passerelle Genmask      I  face
10.1.0.0     *          255.255.0.0  U  eth0
default        blackjack    0.0.0.0  UG eth0
```

stockage

```
[toto@turing ~]$ df -h
Sys. de fich.  Tail. Occ. Disp. %Occ. Monte sur
/dev/mapper/rootvg-rootvol
                7,7G  6,6G 715M   91% /
/dev/mapper/lun_systemp1
                 99M   28M   66M  30% /boot
tmpfs           1014M    0 1014M   0% /dev/shm
/dev/sda2         63G  1,2G   58G  2% /var/log
/dev/mapper/bd_vg-bd_lv
                 764M  502M  263M 66% /bd
/dev/mapper/webdata_vg-webdata_lv
                  90G   46G   45G 51% /webdata
/dev/mapper/web_vg-web_lv
                 2,8G  1,8G 1015M 64% /www
```

who

```
[toto@turing ~]$ whoami
toto
[toto@turing ~]$ wc -l /etc/passwd
141 /etc/passwd
[toto@turing ~]$ w
 11:41:41 up 23 days, 16:02,  1 user,
          load average: 0,18, 0,23, 0,30
USER   TTY      FROM      LOGIN@   WHAT
toto   pts/0    sol83---  11:23    sshd: toto
[toto@turing ~]$ last -n3
toto     pts/0 euphoria Sep 10 11:23   still
michko   pts/0 sol83--- Sep 10 11:03 - 11:15
nicolas  pts/0 pc-nico  Sep  6 16:26 - 16:28
```

machine pour le calcul

```
[toto@imath ~]$ grep -c proc /proc/cpuinfo
24
[toto@imath ~]$ top
21:53:53 up 102 days, load average: 2.00, 2.00, 2.00
Tasks: 431 total,   3 running, 428 sleeping, 0 zombie
Cpu(s):  8.3%us,  0.0%sy,  0.0%ni, 91.7%id,  0.0%wa
Mem:  98986172k total, 73171416k used, 25814756k free
Swap: 76218360k total,   100900k used, 76117460k free

  PID USER     PR  NI  S %CPU %MEM    TIME+  COMMAND
14855 pv       25   0  R 100.0  0.0  10276:06 kpc
14856 pv       25   0  R  99.8  0.0  10263:15 kpc
    1 root     15   0  S  0.0  0.0  41:40.06 init
    2 root     RT  -5  S  0.0  0.0   0:00.24
    3 root     34  19  S  0.0  0.0   0:00.10
```

sécurisation des **password**

```
[toto@turing ~]$ who
toto pts/0          2012-09-01 17:54 (ouest.net)

[toto@turing ~]$ tail -2 /etc/passwd
toto:x:1002:1002:Mr Toto:/home/toto:/bin/bash
titi:x:1003:1002:Mr Titi:/home/titi:/bin/bash

[toto@turing ~]$ cat /etc/shadow
cat: /etc/shadow: Permission non accordee
```

/etc/passwd et /etc/shadow

```
⇝ ls -alt /etc/passwd

-rw-r--r--. 1 root root ... 17:13 /etc/passwd
⇝ ls -alt /etc/shadow
----------. 1 root root ... 17:13 /etc/shadow

[root@localhost ~]# tail -2 /etc/shadow
toto:$6$6gznwvTA$6cM.B8GJvlBYN0QRDnYRU6Px/QQCFoy
bKhbl3Vr4q7XmycL.LYzldXu9eBtaBNTiOGzkBmwDXkwfAj9
K4jaTT0:15584:0:99999:7:::

titi:$6$QMF37bGZ$NfVuIaWrmlCi/cuNaeM.UXHyO9ZS1yw
BieI1LXaxOhAURnW.BwtAGcy6HEpuF93yQt.mxoJd3Dq23Zu
KMkReA0:15584:0:99999:7:::
```

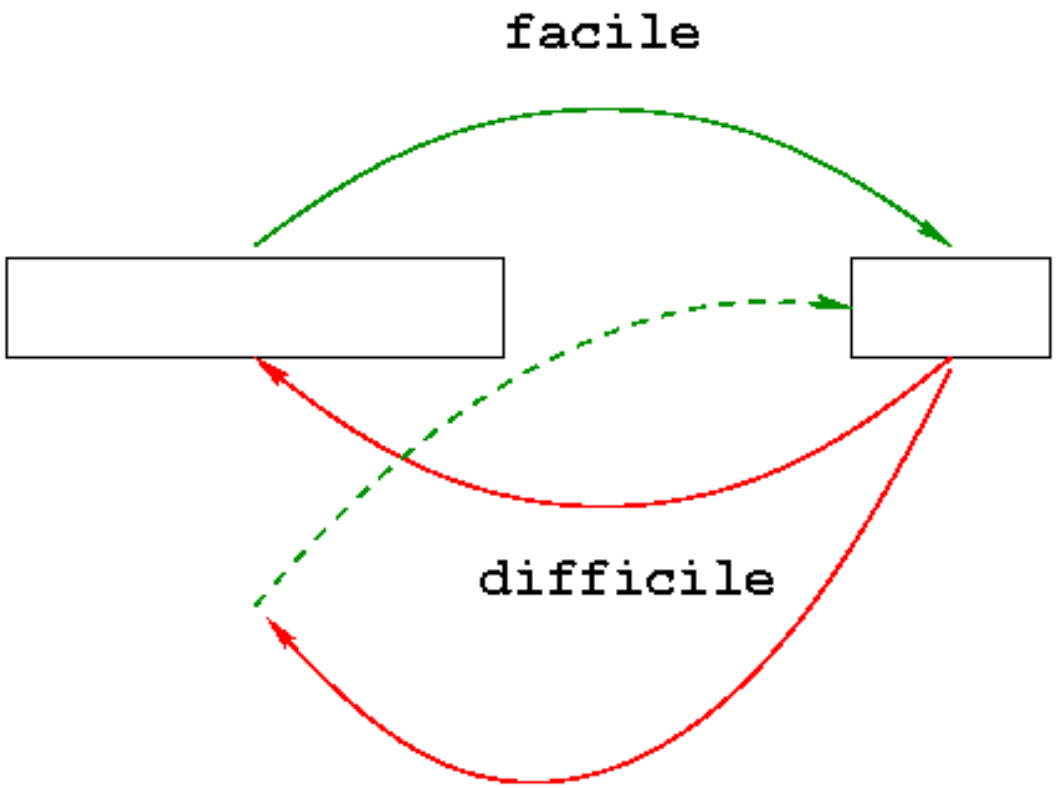
permission

```
⇝ ls -alt /etc/passwd
-rw-r--r--. 1 root root ... 17:13 /etc/passwd

⇝ ls -alt /etc/shadow
----------. 1 root root ... 17:13 /etc/shadow

[root@localhost ~]# tail -2 /etc/shadow
toto:$6$6gznwvTA$6cM.B8GJvlBYN0QRDnYRU6Px/QQCFoy
bKhbl3Vr4q7XmycL.LYzldXu9eBtaBNTiOGzkBmwDXkwfAj9
K4jaTT0:15584:0:99999:7:::

titi:$6$QMF37bGZ$NfVuIaWrmlCi/cuNaeM.UXHyO9ZS1yw
BieI1LXaxOhAURnW.BwtAGcy6HEpuF93yQt.mxoJd3Dq23Zu
KMkReA0:15584:0:99999:7:::
```

man crypt

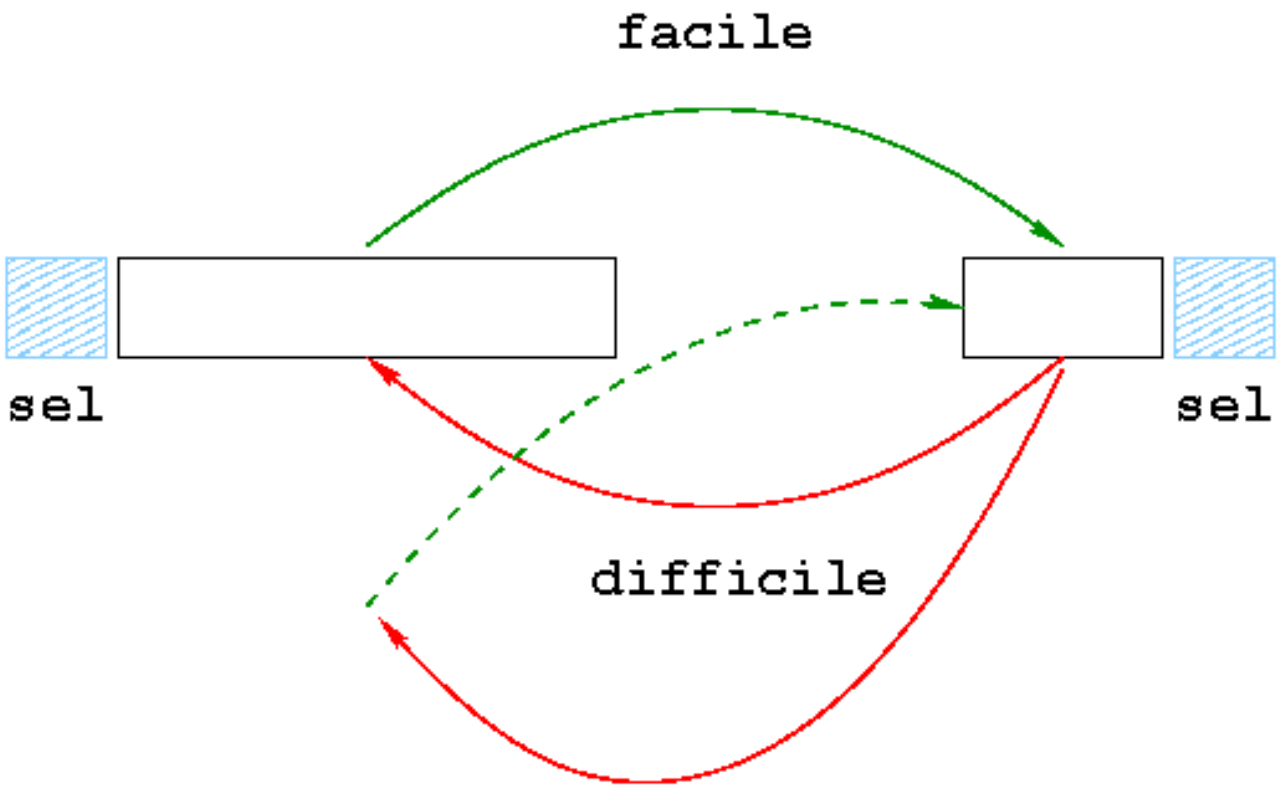
```
CRYPT(3) Manuel du programmeur Linux
NOM crypt - Chiffrage de donnees ou de mot de passe

SYNOPSIS
  #define _XOPEN_SOURCE        #include <unistd.h>
  char *crypt(const char *key, const char *salt);
  edition des liens avec l'option -lcrypt.
DESCRIPTION
       La fonction crypt est celle utilisee pour
       le cryptage des mots de passe.
NOTES
 un cryptrogramme est de la forme
                          $id$salt$encrypted :
 1 : MD5 ...  5 : SHA-256 , 6 : SHA-512
```

fonction à sens unique



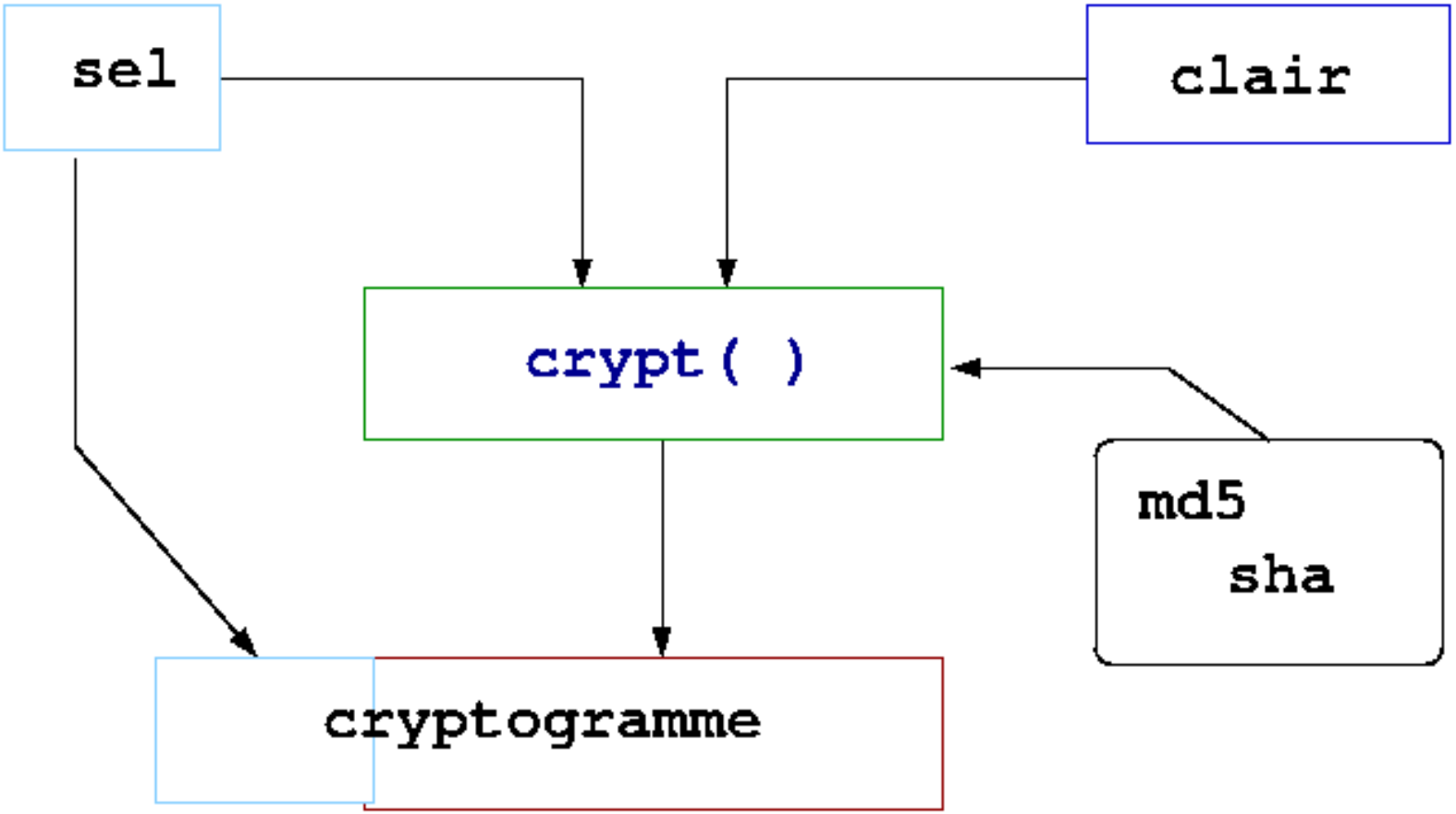
- Le système converve les cryptogrammes (empreintes)
- L'utilisateur doit fournir le clair.

salage



- Le salage aléatoire renforce la sécurité du système.
- Le même passe conduit à des empreintes différentes.

hachage



décryptage

```
1  #define _XOPEN_SOURCE        /* See feature_test_macros(7)
       */
2  #include <unistd.h>
3  #include <stdio.h>
4  #include <string.h>
5  #include <stdlib.h>
6  int main (int argc, char* argv[] )
7  {
8    char *res;
9    res = crypt( argv[1], argv[2]) ;
10   if ( strcmp( argv[2], res ))
11       return 1;
12
13 return 0;
14 }
```

décryptage

```
:: gcc -Wall pwd.c
/tmp/ccgsfZqg.o: In function 'main':
pwd.c:(.text+0x21): undefined reference to 'crypt'
collect2: ld a retourne 1 code d'etat d'execution
:: gcc -Wall pwd.c -lcrypt
:: ./a.out toto '$6$6gznwvTA$6cM.B8GJvlBYN0QRDnYRU
6Px/QQCFoybKhbl3Vr4q7XmycL.LYzldXu9eBtaBNTiOGzkBmw
DXkwfAj9K4jaTT0'

:: echo $?
0
```

- `strcmp` renvoie 0 en cas d'égalité
- commande unix : 0 = succès (true)

cassage, charte

Une fois connu le mot de passe crypté, il reste possible de casser les mots de passe faibles par la force brute ...

- **/etc/shadow** est un anti-cracker !

[John the Ripper]

déontologie : Un utilisateur `unix` signe une charte qui stipule qu'un mot de passe est strictement personnel.

5 - bash

- ligne de commande
- analyse
- quote
- expansion
- fonction et tableau
- commandes
- pipeline
- commandes séquentielles
- alternative
- test
- itérations
- choix multiple
- menus
- options

Edition

Bash utilise la bibliothèque readline qui inclut la *complétion automatique* des noms de commandes et de fichiers (tabulation) ainsi que la *navigation* dans l'historique des commandes.

- set pemet de sélectionner un éditeur : vi, emacs (défaut).
- 8 Command Line Editing

readline	emacs	vi
beginning-of-line	CTRL-a	0
end-of-line	CTRL-e	$
clear-screen	CTRL-l	
reverse-search-history	CTRL-r	
forward-search-history	CTRL-s	
kill-line	CTRL-k	d$
backward-kill-line	CTRL-x Rubout	d0
unix-line-discard	CTRL-u	dd
unix-word-rubout	CTRL-w	dw
yank	CTRL-y	p
display-shell-version	CTRL-x C-v	

- set -o vi
- set -o emacs
- bind, **.inputrc**

readline

```
1  static char *ligne = (char *) NULL;
2  char *nligne(void) {
3      if (ligne) {
4      free(ligne);
5      ligne = (char *) NULL;
6      }
7      ligne = readline("prompt:");
8      if (ligne && *ligne)
9              add_history(ligne);
10     return (ligne);
11 }
12 int main(void) {
13     while ( strcmp(nligne(), "quit"));
14     return 0;
15 }
```

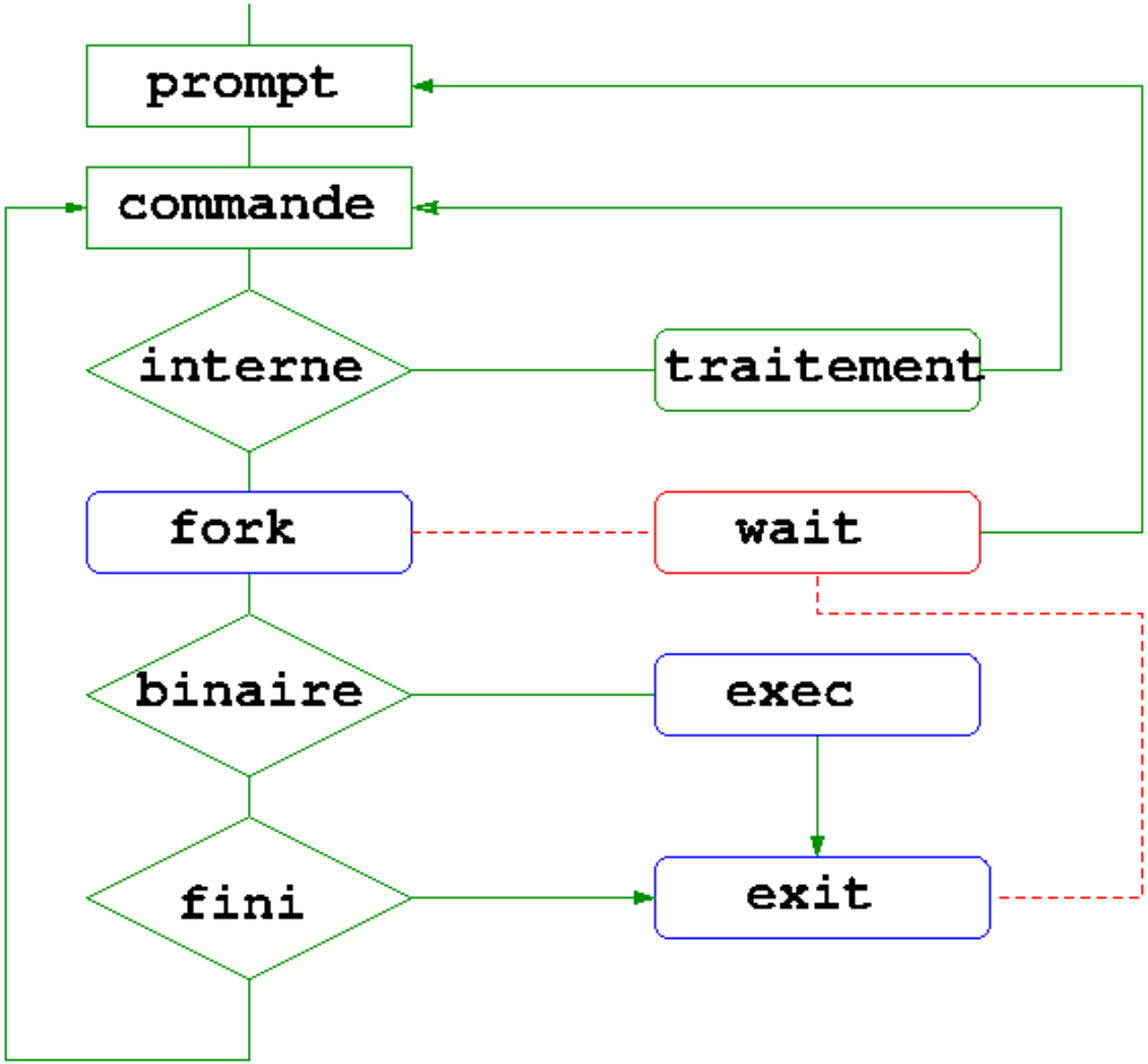
traitement des commandes

Le prompt du shell matérialise une ligne de commande invitant l'utilisateur à soumettre une instruction : une chaine de caractères analysée par le shell pour lancer des commandes.

- interne : traitée dans le processus.
- externe : `clone` + `execve`.
- complexe : `clone`

```
11:47 pl@mic/script> echo $PS1
\A pl@$MACHINE/\W>
```

boucle du shell



Interprétation des lignes

`bash` segmente la ligne en mots (tokens), le premier mot est généralement une commande :

1. alias
   - **.bashrc**
   - `alias`
2. commande interne
   - `enable`
3. binaire
   - variable **PATH**
   - `which`

les autres arguments sont des options ou paramètres de la commande.

what's for ?

Un exemple proposé par Chet Ramey :

```
1 for for in for; do for=for; done; echo $for
```

what's for ?

Un exemple proposé par Chet Ramey :

```
1 for for in for; do for=for; done; echo $for
```

Nous ne tenterons pas de percer les mystères de la grammaire du shell `bash`, tout au plus quelques points extraits du manuel :

```
⇝ wget http://www.gnu.org/software/bash/manual/
    bash.html
⇝ man bash
```

Ne sont pas vraiment abordés ici :

- les fonctions,
- les tableaux,
- les tableaux associatifs.

jeux de caractères

Méta-caractères (séparateurs) :

| & ; ( ) < >

Opérateurs de contrôle :

| |& || && ; &

Opérateurs de redirection :

< << > >> >& <<<

Paramètres spéciaux :

* @ # ? $! 0 _ ~

Mots réservés :

```
1 ! case do done elif else esac fi for function if
2 in select then until while { } time [[ ]]
```

apostrophe

Les protections permettent de forcer l'interpréteur à ignorer la signification spéciale de certains caractères ou mots. Les protections peuvent être utilisées pour empêcher le traitement des caractères spéciaux, éviter la reconnaissance des mots-réservés ou empêcher le développement des paramètres. Il y a trois mécanismes de protection :

- le caractère d'échappement (antislash)
- les apostrophes (quote)
- les guillemets (double-quote)

quoting

- \ protége le caractère qu'il précède.
- les apostrophes protégent une chaîne de caractères.

entre deux guillemets :

- $ est toujours actif.
- \ affecte

```
$    \    "    <nl>
```

cow quotes

```
⇝ cowsay -TU 'how to quote ?'
 _________________
< how to quote ? >
 -----------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
             U ||----w |
                ||     ||
```

cow explains how to quote

```
1 #!/bin/bash -v
2 x=quote; cowsay -T\\/ "how to '$x' ?"
```

```
#!/bin/bash -v
x=quote; cowsay -T\\/ "how to '$x' ?"
 ------------------
< how to 'quote' ? >
 ------------------
        \    ^__^
         \   (oo)\_______
             (__)\       )\/\
              \/ ||----w |
                 ||     ||
```

quotes

```
⇝ cowsay -f small '"$$"' "$HISTSIZE" "\$$"
 _____________
< "$$" 512 $$ >
 _____________
        \    ,__,
         \   (oo)____
             (__)    )\
                ||--|| *
```

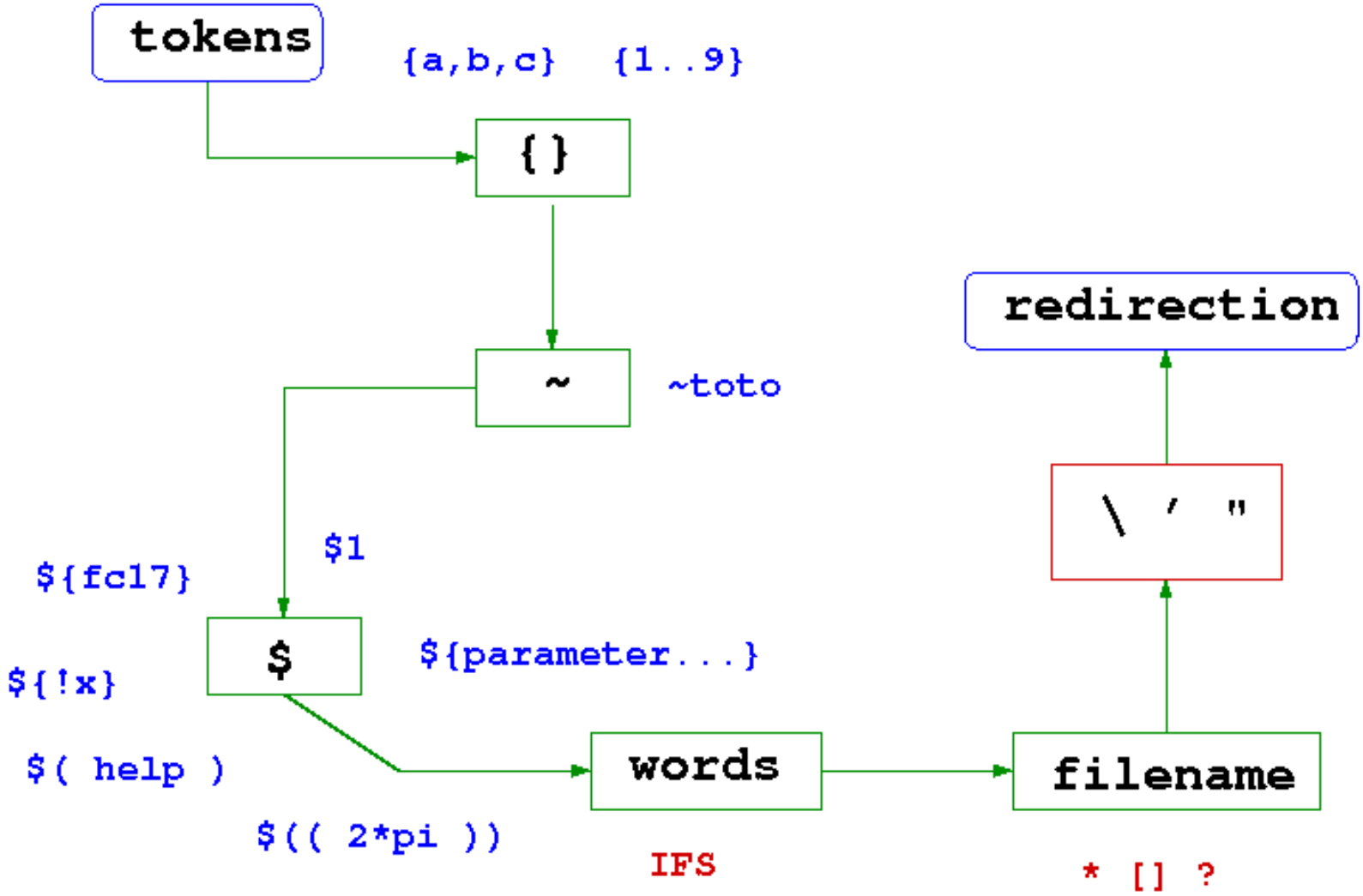
digression

```
1  #!/bin/bash -v
2    z=hello
3    printf -v z "%s $USER" $z
4    echo $z
5  hello pl
6    man ascii | grep -o '[0-9][0-9 ]*A$'
7  101   65    41    A
8    x=A
9    printf "%3o %2x %3d %s" \'$x \"$x "'$x" $x
10 101 41  65 A  z=65
11   printf -vx "0%3o" $z
12   echo -e "\\$x"
13 A
14   printf \\$(printf "%03o" $z)
15 printf "%03o" $z)
```

Analyse des commandes

1. Lecture : fichier, chaine, terminal
2. Analyse lexicale : application des quotes, suite de tokens (opérateur, mot) séparés par des métacaractères.
3. Traitement des alias.
4. Analyse syntaxique des commandes.
5. Expansion des tokens : noms de fichiers, commandes et arguments.
6. Traitement des redirections
7. Exécution des commandes.
8. Terminaison des fils, status.

expansion des tokens



{a,b,c,...} : accolade (brace)

```
1 #!/bin/bash
2 for x in {alice ,eve,bob}.{X,Y}
3 do
4     touch /tmp/$x.txt
5 done
6 ls  /tmp/[a-z]*.[XYZ]*.txt
```

```
/tmp/alice.X.txt
/tmp/alice.Y.txt
/tmp/bob.X.txt
/tmp/bob.Y.txt
/tmp/eve.X.txt
/tmp/eve.Y.txt
```

{1..n} : enumération

```
1 #!/bin/bash
2 if [ ! -f /tmp/host.txt ] ; then
3     for x in {1..254} ; do
4         host 193.49.96.$x
5         done | grep -v nat > /tmp/host.txt
6 fi
7 cut -d ' ' -f 1,5 < /tmp/host.txt | head -5
```

```
1.96.49.193.in-addr.arpa rhodes.univ-tln.fr.
2.96.49.193.in-addr.arpa mail.univ-tln.fr.
3.96.49.193.in-addr.arpa lorcane.univ-tln.fr.
4.96.49.193.in-addr.arpa dpt-info.univ-tln.fr.
5.96.49.193.in-addr.arpa serecom.univ-tln.fr.
```

tilde

```
1  #!/bin/bash -v
2
3  echo -n ~{alice,pl,bob,guest}
4  ~alice /home/pl ~bob /home/guest
5  for gus in {alice,pl,bob,guest}
6  do
7      echo ~$gus
8  done
9  ~alice
10 ~pl
11 ~bob
12 ~guest
```

argument et paramètre de position

```
1 #!/bin/bash -v
2 echo $0 : $*
3 set a b c
4 echo $*
5 while [ $# -gt 0 ]; do
6   echo $1 : $*
7   shift
8 done
```

décalage des paramètres de position

```
1  #!/bin/bash -v
2  echo $0 : $*
3  ./shift.sho :
4  set a b c
5  echo $*
6  a b c
7  while [ $# -gt 0 ]; do
8    echo $1 : $*
9    shift
10 done
11 a : a b c
12 b : b c
13 c : c
```

tri des arguments

```
1 #!/bin/bash
2 if [ $# = 0 ] ; then
3     set 2 1 4 3 6 5 8 7
4 fi
5 echo $* | tr ' ' '\n'                \
6          | sort -n | tr '\n' ' '
```

paramètres spéciaux

```
1  #!/bin/bash -v
2  echo opt=$- pid=$$
3  opt=hvB pid=2615
4  bash -c 'echo child =$$' &
5  echo hello > /tmp/spec
6  echo $?
7  0
8  echo hello > /root/tmp
9  ./spec.sho: line 6: /root/tmp: Permission non accordee
10 echo $? $!
11 1 2617
12 child =2617
```

séparateur interne de champs

```
1  #!/bin/bash -v
2  set gnu is  not  unix
3
4  echo "$*"
5  gnu is  not  unix
6  echo "$@"
7  gnu is  not  unix
8
9  IFS=:
10
11 echo "$*"
12 gnu:is : not: unix
13 echo "$@"
14 gnu is  not  unix
```

danger !

Le caractère ! est utilisé :

- opérateur logique : !=
- **pid** dernier job $!
- indirection ${ !... }
- historique !

```
39 echo hello
40 touch this
41 echo coucou
42 cat file
43 history
⇝ !39
echo hello
⇝ !-5
touch this
⇝ !ec
echo hello
⇝ !? file
cat file
⇝ ^file^foo^
cat foo
```

nom de fichier

Une expression contenant un des

```
[ * ] ?
```

donne la liste des noms de fichiers du répertoire courant qui sont en correspondance avec l'expression :

- * : chaine arbitraire
- [] : ensemble
- ? : caractère
- Absence de correspondance : attention !

filename

```
1  #!/bin/bash -v
2  if [ ! -d tmp ] ; then mkdir tmp ;fi
3  cd tmp
4  touch abc
5  touch cba
6  echo a[abc]b
7  a[abc]b
8  echo a[abc]c
9  abc
10 shopt -s failglob
11 echo a[abc]b
12 ./filename.sho: line 9: Pas de correspondance : a[abc]b
```

$(commande)

```
1 #!/bin/bash -v
2
3 man -s 1 -f bc
4
5 res=$( bc <<< "2^(2^5) + 1" )
6
7 factor $res
```

Il existe une syntaxe synomyme à base d'anticotes.

$((arithmétique))

```
1  #!/bin/bash -v
2  x=010
3  y=2
4  echo $(( x * y ))
5  16
6
7  let y++
8  echo $y
9  3
10
11 let z=x*y
12 echo $z
13 24
```

```
1  #!/bin/bash -v
2
3  echo $(( 2#1000000 ))
4  64
5  echo $(( 16#abcdef ))
6  11259375
7
8  x=$(( 16#abcdef ))
9  echo $( bc <<< "obase=16; $x"
       )
10  bc <<< "obase=16; $x" )
11  bc <<< "obase=16; $x"
12 ABCDEF
```

${parameter : [-+= ?] }

```
1  #!/bin/bash -v
2  echo ${foo:-hello world}
3  hello world
4  echo ${bar:=hello world}
5  hello world
6  echo foo=$foo bar=$bar
7  foo= bar=hello world
8  echo ${bar:+salut tout le monde}
9  salut tout le monde
10 echo ${bar:?}
11 hello world
12 echo ${foo:?}
13 ./para1.sho: line 7: foo : parametre vide ou non defini
```

chaine,sous-chaine,prefixe,suffixe

```
1  #!/bin/bash -v
2  str='Hello everybody out there using minix'
3  echo ${str:6}
4  everybody out there using minix
5  echo ${str:6:10}
6  everybody
7  # sharp prints suffixe
8  echo ${str#*y}
9  body out there using minix
10 echo ${str##*y}
11 out there using minix
12 # percent prints prefixe
13 echo ${str%%o*} ${str%o*}
14 Hell Hello everybody
```

${parameter/motif/remplace}

```
#!/bin/bash -v
claim='I do not believe  everybody should learn  to code'
echo ${claim/d*t /really\ }
I really  believe  everybody should learn  to code
echo ${claim//d*t/sometime d}
I sometime do code
echo \$\{str{^,^^,\,,\,\,} pat\}
${str^pat} ${str^^pat} ${str,pat} ${str,, pat}
claim=${claim^^*}
echo ${claim/#D*T/}
I DO NOT BELIEVE EVERYBODY SHOULD LEARN TO
    CODE
echo ${claim/#I*Y/YOU} BASH!
YOU SHOULD LEARN TO CODE BASH!
```

${!variable} ${!prefixe} : déréférencement !

```
⇝ foo=hello; ptr=foo ; echo ${!ptr}
hello
```

```
⇝ echo ${!H*}
HISTCMD HISTCONTROL HISTFILE HISTFILESIZE
    HISTSIZE HOME HOSTNAME HOSTTYPE
```

```
⇝ env | grep HIST
HISTSIZE=512
HISTFILESIZE=2048
HISTCONTROL=ignoredups
HISTFILE=/home/pl/.compil_history
```

Bash :

${!variable} ${!prefixe} : déréférencement !

```
⇝ foo=hello; ptr=foo ; echo ${!ptr}
hello
```

```
⇝ echo ${!H*}
HISTCMD HISTCONTROL HISTFILE HISTFILESIZE
    HISTSIZE HOME HOSTNAME HOSTTYPE
```

```
⇝ env | grep HIST
HISTSIZE=512
HISTFILESIZE=2048
HISTCONTROL=ignoredups
HISTFILE=/home/pl/.compil_history
```

Bash :

```
⇝for x in ${!HIST*}; do echo $x=${!x}; done
```

fonction

```
1  #!/bin/bash
2  function chr {
3    printf \\$( printf '%03o' $1 )
4  }
5  function ord {
6    printf '%d' "'$1"
7  }
8
9  for x in $* ; do
10    echo -ne \\t$x: ; ord $x
11 done
```

```
⇝ ./fonction.sho {A..E}
    A:65    B:66    C:67    D:68    E:69
```

globale par défaut

```
1  #!/bin/bash -v
2  foo=globale
3
4  function myproc {
5  local foo=locale
6  echo $foo
7  echo $bar
8  }
9
10 bar=$foo
11
12 myproc
13 locale
14 globale
```

tableau

```
1  #!/bin/bash -v
2  week=(lundi mardi mercredi jeudi  vendredi  samedi dimanche)
3  echo x=$week bad=$week[1] good=${week[1]}
4  x=lundi bad=lundi[1]  good=mardi
5  echo dim:{#week[*]} val:${week[*]}
6  dim:{#week[*]} val:lundi  mardi mercredi  jeudi  vendredi
       samedi dimanche
7  echo ind:${!week[*]}
8  ind:0  1 2 3 4 5 6
9  for i  in  ${!week[*]} ;  do
10    day=${week[i]}
11    echo -n ${day:0:1}
12 done
13 lmmjvsd
```

tableau associatif

```
1 #!/bin/bash
2 declare -A count
3 for w in $( cat $0 | sed 's/[^a-z]/\n/g' ) ; do
4  let count[$w]++
5 done
6
7 for w in ${!count[@]} ; do
8     echo -n ' '$w:${count[$w]}
9 done | fold -sw40
```

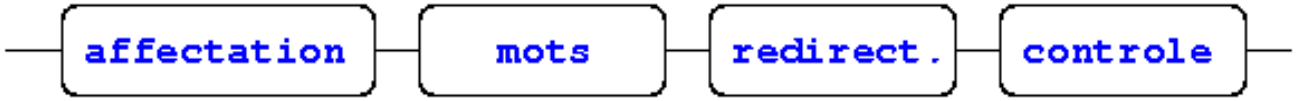
```
 do:2 let:1 declare:1 bash:1 done:2
bin:1 fold:1 count:4 echo:1 cat:1 sed:1
a:1 in:2 g:1 n:2 s:1 for:2 w:5 z:1 sw:1
```

types de commandes

Une commande composée d'arguments i.e. `echo a b c` est une *commande simple* à partir desquelles on peut construire des commandes complexes :

- pipelines (tube)
- listes (batch)
- composée (contrôle)
  - Coprocesses Two-way communication between commands.
  - GNU Parallel Running commands in parallel.

commande simple



- En général, la commande est spécifiée par le premier mot, les autres sont les arguments de la commande.
- Un signal affecte la valeur de retour :

$$128 + \text{signal}$$

[time [-p]] [!] CMD_1 [[| ou |&] CMD_2 ...]

Un *pipeline* est une suite de *commandes simples* separées par un opérateur de contrôle :

‘ | ’ ou ‘ |& ’.

- La sortie de CMD_1 est connectée par un tube vers l'entrée de CMD_2.
- Autrement dit, CMD_2 lit dans la sortie de CMD_1.
- La connexion est réalisée avant l'évaluation des redirections.
- Avec ‘ |& ’, le canal d'erreur de CMD_1 est dirigé vers l'entrée de CMD_2.

Il s'agit d'un raccourci pour :

$$\mathrm{CMD}_1\ 2{>}\&1\ \ |\ \mathrm{CMD}_2$$

Cette redirection implicite est réalisée aprés l'évaluation des redirections.

exemple de pipeline

```
⇝ man bash | sed 's/[^A-Z]/\n/g'| \
  grep -E '[A-Z]{3}'| sort | uniq -c | sort -nr
     54 BASH
     42 SHELL
     29 BUILTIN
     27 COMMANDS
     21 PATH
     21 IFS
     18 COMP
     16 POSIX
     13 EXPANSION
     13 ENV
     11 NAME
```

Liste de commandes

Une liste de commandes est un enchainement de pipelines séparés par un des opérateurs :

‘ ;’ ‘&’ ‘&&’ ‘ || ’

qui se termine éventuellement par ‘ ;’ ou ‘&’, ou ’ NL ’.

- Les opérateurs ‘&&’ et ‘ || ’ ont la même précédence, et sont prioritaires aux deux autres ‘ ;’ et ‘&’.
- Une commande qui se termine par ‘&’ est exécutée en arrière plan (exécution asynchrone) dans un sous-shell. Le shell n’attend pas la terminaison de la commande, le code est 0 (true).
- L’entrée standard est par défaut /dev/null.
- Les commandes séparés par des ‘ ;’ sont exécutées séquentiellement. Le shell attend la terminaison de chacune des commandes à tour de rôle.
- La valeur de retour correspond au status de la dernière

liste conditionnelle

Il s'agit des séquences séparées par des '&&' ou bien des ' || '.
Dans une conjonction :

$$\text{CMD}_1 \ \&\& \ \text{CMD}_2$$

CMD_2 est exécutée si et seulement si le retour de CMD_1 est OK.
Dans une disjonction :

$$\text{CMD}_1 \ \text{——} \ \text{CMD}_2$$

CMD_2 est exécutée si et seulement si le retour de CMD_1 est KO.
Dans les deux cas, la valeur de retour sera bien celle de la dernière commande exécutée.

exemple

```
1  #!/bin/bash -v
2  help time | sed -nE '/[pP]osix/s/[[: space:]]+/ /p'
3   -p  print the timing summary in the portable Posix format
4  time -p (sleep 1 && sleep 2 && sleep 3)
5  real 6.00
6  user 0.00
7  sys 0.00
8  time -p (sleep 1 || sleep 2 || sleep 3)
9  real 1.00
10 user 0.00
11 sys 0.00
12 time -p (sleep 1 & sleep 2 & sleep 3)
13 real 3.00
14 user 0.00
15 sys 0.00
```

(liste)

exécutée dans un sous-shell sans effet sur l'environnement.

```
1 #!/bin/bash -v
2 x=HELLO
3 echo x=$x s=$BASH_SUBSHELL
4 x=HELLO s=0
5 ( x=${x,,*}; echo x=$x s=$BASH_SUBSHELL; )
6 x=hello s=1
7 echo x=$x s=$BASH_SUBSHELL
8 x=HELLO s=0
```

```
⇝ ( cd /tmp; pwd ) ; pwd
/tmp
/home/pl
```

{liste ;}

exécutée avec l'environnement du shell en cours !

```
1 #!/bin/bash -v
2 x=hello
3 echo x=$x s=$BASH_SUBSHELL
4 x=hello s=0
5 { x=${x^^*}; echo x=$x s=$BASH_SUBSHELL; }
6 x=HELLO s=0
7 echo x=$x s=$BASH_SUBSHELL
8 x=HELLO s=0
```

```
1 :: { echo hello; echo world } >> message
```

if test-cmd ; then CMD_0 ;[else CMD_1 ;]fi

Si le code de retour de `test-cmd` est nul alors CMD_0 est exécutée sinon c'est CMD_1.

```
1 if [ -n HOST ] ; then
2     if ping -c1 $HOST.euphoria.fr ; then
3         ssh pl@$HOST.euphoria.fr who
4     else
5         echo $HOST is down
6     fi
7 fi
```

Les commandes [...] et [[...]] sont utilisées pour construire des tests.

- [ ] test simple, détaillé ici.
- [[ ]] voir le manuel.
- `elif` construction est possible.

help test

-d file	répertoire
-f file	fichier régulier
-p file	pipe, fifo
-x file	exécutable
-N file	modifié depuis dernier accès
-o optname	option du shell activée
-v varname	variable assignée
-z string	chaine vide
-n string	chaine non vide

test unaire

```
1  #!/bin/bash -v
2  #  0  :  SUCCES
3  #  1  :  FAILURE
4  unset s
5  [ -z "$s"    ];  echo $?
6  0
7  [ -n "$s"    ];  echo $?
8  1
9  s=initialisation
10 [ ! -z "$s"    ] && echo non vide
11 non vide
12 [ -n  "$s"    ] && echo init
13 init
```

test binaire

file1 -nt file2	file1 est plus récent que file2
file1 -ot file2	file1 est plus ancien que file2
str1 = str2	égalité des chaines
str1 != str2	chaines différentes.
str1 < str2	comparaison lexicographique
str =~ pat	correspondance
arg1 ♣ arg2	comparaison arithmétique ♣ : '-eq', '-ne', '-lt', '-le', '-gt', or '-ge'.

```
$ [ "x" = "x" ]; echo $?          —> 0
$ [ "x" < "y" ]; echo $?          —> 1
bash: y: Aucun fichier ou dossier de ce type
$ [[ "x" < "y" ]]; echo $?        —> 0
$ [[ azerty =~ a.*y ]]; echo $? —> 0
```

test binaire

```
1  #!/bin/bash -v
2  x='1234'
3  y=' 1234'
4  if [ "$x" = "$y" ] ; then
5      echo egalite
6  else
7      echo difference
8  fi
9  difference
10 if [ $x = $y ] ; then
11     echo egalite
12 else
13     echo difference
14 fi
15 egalite
```

expression logique

! expr	négation de expr
(expr)	
expr1 -a expr2	et logigue
expr1 -o expr2	ou logique

expression logique

```
1  #!/bin/bash -v
2  function transfert  {
3  ls  -lt  $2 $1/$2
4  if  [  -d $1 -a -f $2 -a  "$2" -nt "$1/$2"    ] ; then
5        cp -f $2 $1/$2
6  fi
7  }
8  echo hello  > /tmp/h
9  sleep  60
10 echo Hello > h
11 transfert    /tmp h
12 -rw-rw-r-- 1 pl pl 6 20 juin  08:24 h
13 -rw-rw-r-- 1 pl pl 6 20 juin  08:23 /tmp/h
14 transfert    /tmp h
15 -rw-rw-r-- 1 pl pl 6 20 juin  08:24 /tmp/h
```

while test-cmds ; do seq-cmds ; done

Exécute la suite de commandes tant que le code de retour de `test-commands` vaut zéro. Le code de retour est celui de la dernière commande exécutée.

```
1 while   ping -c1 -W1 192.168.0.254; do
2       echo alive
3       sleep 60
4     done
```

- les " ;" optionnel deviennent obligatoires sur une ligne !

L'inévitable factorielle !

```
1  declare -i n=1
2  r=1 ; z=1
3  # comparaison des chaines
4  while [ $r == $z ]
5  do
6      # n est entier
7      n=n+1
8      # r non entier
9      let r=r*n
10     z=$( bc <<< "$z*$n" )
11 done
12 echo n=$n $r $z
```

L'inévitable factorielle !

```
1  declare -i n=1
2  r=1 ; z=1
3  # comparaison des chaines
4  while [ $r == $z ]
5  do
6      # n est entier
7      n=n+1
8      # r non entier
9      let r=r*n
10     z=$( bc <<< "$z*$n" )
11 done
12 echo n=$n $r $z
```

```
1  n=21 -4249290049419214848 5109094217170944000 0
```

L'inévitable factorielle !

```
1  #!/bin/bash
2  z=$( \
3  bc  << -EOJ
4      r=1
5      for( n=1 ; n < 50; n++ )
6          r=r*n
7      print  r
8  -EOJ
9  )
10 echo $z
```

while et read

La commande `read` utilise le premier caractère de la variable `IFS` pour découper les lignes d'un fichier :

```
1 #!/bin/bash
2 IFS=:
3 while read user x  uid  z
4 do
5     if  [  $uid  -ge 1000 ]
6     then
7        echo $user
8     fi
9 done < /etc/passwd
```

mais une commande sait faire cela, laquelle ?

while et read

La commande `read` utilise le premier caractère de la variable `IFS` pour découper les lignes d'un fichier :

```
1 #!/bin/bash
2 IFS=:
3 while read user x  uid  z
4 do
5     if  [  $uid  -ge 1000 ]
6     then
7       echo $user
8     fi
9 done < /etc/passwd
```

mais une commande sait faire cela, laquelle ?

```
[]$  gawk -F: '{if ($3 >=1000) print $1}' /etc/
   passwd
```

for name [[in [words...]] ;] do cmds ; done

- expansion de `words`
- exécution des commandes

```
1 #!/bin/bash
2 set a b c
3 for x        ;   do  echo -n $x; done
4 for x in * ;   do  echo -n $x; done
5 for x in $( cat $0 ); do echo -n $x; done
6 for x in alpha beta gamma ;
7   do
8       echo -n $x;
9   done
```

- pas de `in words` est spécial.

for ((expr1 ; expr2 ; expr3)) ; do cmds ; done

Avec un comportement similaire à la bouble `for` du langage `C` : l'expression arithmétique `expr1` est évaluée, puis celle de `expr3` tant que `expr2` n'est pas nulle.

```
1  deb=$1
2  lim=$2
3  pas=1
4  if [ $# = 3 ] then
5      pas=$3
6  fi
7
8  for (( x=deb; x<lim ; x+=pas ));
9    do
10     echo $x;
11   done
```

Chronométrage

```
1  deb=$1 ; lim=$2 ; pas=$3 ; cmd=$4
2  if [ $# != 4 ] ; then
3      echo usage $0 : deb lim pas cmd
4      exit 1
5  fi
6  if [ -f time.log ] ; then
7      echo effacer time.log
8      exit 2
9  fi
10 for (( x=deb; x<lim ; x+=pas )) ; do
11    echo -n .$x
12    /usr/bin/time -a -o time.log --format="$x %e" $cmd $x
13    done
```

- gnuplot...

case wd in [[(] x [| x]...) cmd-list ; ;]...esac

case exécute la liste de commandes correspondant au premier motif correspondant.

- | sépare les motifs
-) termine les motifs

Chaque clause se termine par :

‘;;’ ‘;&’ ‘;;&’.

```
1 echo -n "Enter the name of an animal: "
2 read ANIMAL
3 echo -n "The $ANIMAL has "
4 case $ANIMAL in
5   horse | dog | cat) echo -n "four";;
6   man | kangaroo ) echo -n "two";;
7   *) echo -n "an unknown number of";;
8 esac
```

select name [in words...] ; do cmds ; done

Une liste d'items est contruite à partir de la suite `words`. Elle est placée sur la sortie standard. L'usager est PS3-invité à faire un choix passé dans la variale `REPLY`.

```
1  mkdir  /tmp/dir
2  rm -rf /tmp/dir/*
3  touch  /tmp/dir/{a..h}
4  select fname in /tmp/dir/*;
5  do
6      index=$REPLY
7      break;
8  done
9  echo -e "\nselection: $fname ($index)"
10 echo ---
```

```
1) /tmp/dir/a   3) /tmp/dir/c
5) /tmp/dir/e   7) /tmp/dir/g
2) /tmp/dir/b   4) /tmp/dir/d
6) /tmp/dir/f   8) /tmp/dir/h
#?
selection: /tmp/dir/g (7)
```

```
select.out : select.sh
 echo 7 | ./select.sh \
                |& fold -s -w30 > select.out

 sed -n '/select.out/,/clean/p' \
                makefile >> select.out
clean:
```

getopts optstring name [args]

La commande `getopts` permet de gérer des options passées par la ligne de commande :

- `optstring` les options à reconnaitre, avec argument ou pas.
- `name` option courante.
- `OPTIND` indice de l'argument suivant.
- `OPTARG` argument de l'option.

Traitement des erreurs

- le drapeau `OPTERR` contrôle les messages d'erreurs.
- idem ' :' préfixe `optstring`.
- `name` vaut '?' pour une option inconnue.
- `name` vaut ' :' pour un argument attendu.

utilisation de getopt

```
1  #!/bin/bash
2  while getopts ":hp:r:" opt; do
3    case $opt in
4      p) parm=$OPTARG;;
5     \?) echo "option invalide  $OPTARG";exit 1 ;;
6      :) echo "argument requis $OPTARG";exit 2 ;;
7      h) echo "usage $0 [ -p nombre ] [ fichier  ]"
8         exit 3 ;;
9    esac
10 done
11 echo $*
12 shift $((OPTIND-1))
13 [[ $parm =~ $reg ]] && echo matching
```

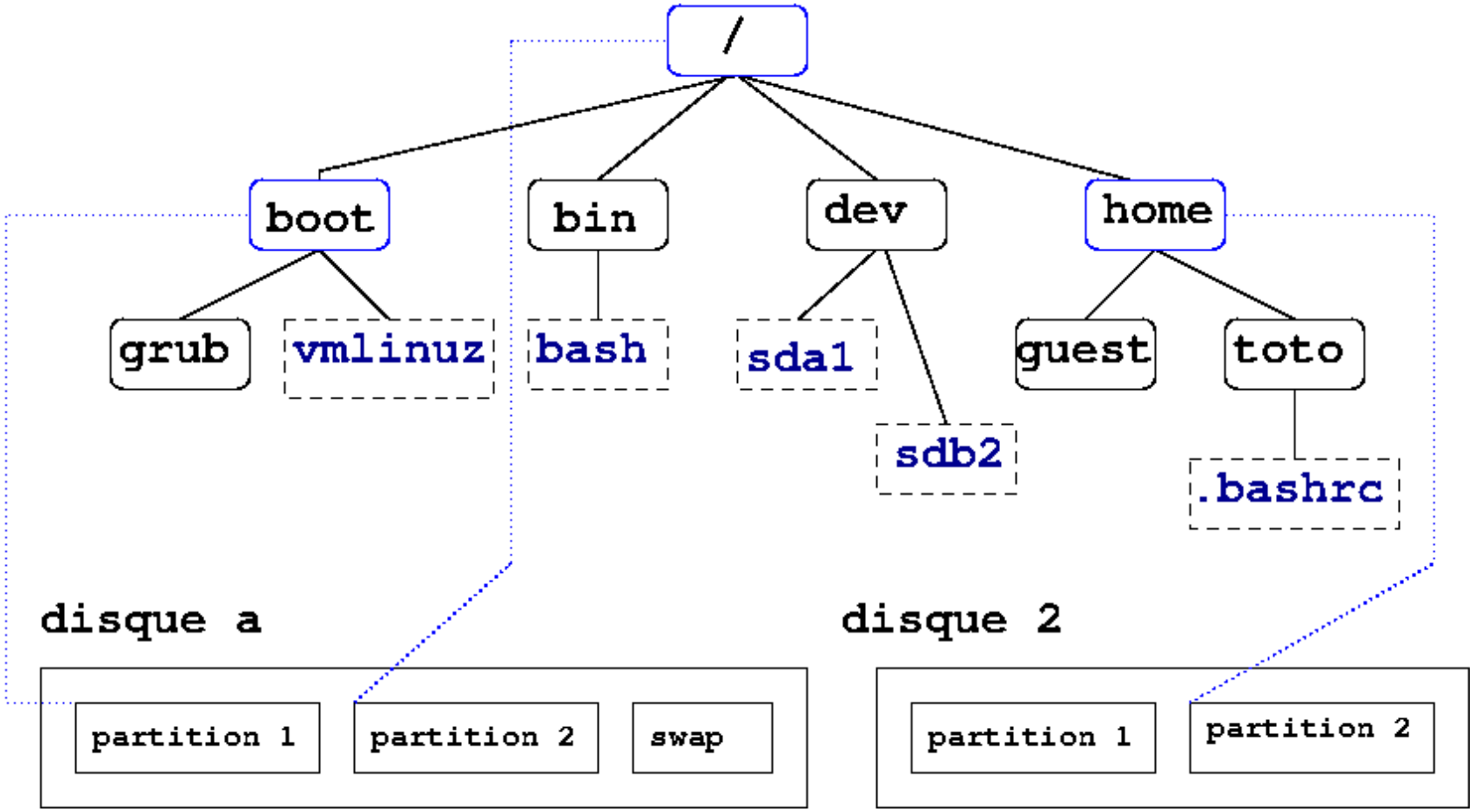
```
-p 1245 [0-5]{3}
```

6 - fichier

- structure
- l-noeud
- commande fichier
- répertoire
- tube nommé
- netcat
- commande réseau

structure générale

L'ensemble des systèmes de fichiers d'un système `unix` est un arbre enraciné dans répertoire / .



Systèmes de fichiers

- `mount` greffe des systèmes de fichiers, ces noeuds sont des points de montage.

```
FILESYSTEMS(5)  Linux Programmer's Manual
NAME
 filesystems - Linux file-system types: minix,
 ext, ext2, ext3, ext4, Reiserfs, XFS, JFS, xia,
 msdos, umsdos, vfat, ntfs, proc, nfs, iso9660,
 hpfs, sysv, smb, ncpfs
DESCRIPTION
 In order to use a file system, you have to mount
   it;
 proc        pseudo file system
 iso9660     CD-ROM file system
 nfs         network file system
```

Quelques points

```
⇝ mount
/dev/sda1 on /
    type ext4 (rw,errors=remount-ro)
proc       on /proc
    type proc (rw,noexec,nosuid,nodev)
tmpfs      on /media
    type tmpfs (rw,nosuid,nodev,noexec,relatime,
   seclabel,mode=755)
sinfo1:/home/perso on /home/perso
    type nfs (rw,soft,intr,sloppy,addr
   =10.9.185.1)
curlftpfs#ftp://freebox:----@mafreebox.freebox.
   fr/Disque%20dur/
        on /media/freebox
    type fuse (rw,nosuid,nodev,noexec,relatime
```

Organisation

bin	exécutables fondamentaux [binaries]
boot	noyau vmlinuz et fichiers de démarrage
dev	périphériques [device]
etc	fichiers de configuration des services
home	données personnelles des utilisateurs
lib	bibliothèques [libraries]
mnt	point de montage temporaire
media	point de montage automatique
proc	pseudo-fichiers des processus en cours
root	répertoire du super-utilisateur
sbin	[(system|superuser) binaries]
tmp	fichiers temporaires
usr	accessibles à tous les utilisateurs
var	fichiers variables, souvent modifiés

- FHS système de fichiers

structure d'un système de fichier

Un système de fichiers comprend trois zone :

- superbloc
- table de noeuds
- fichiers et répertoire

```
⇝ df -i
Sys. fich.      Inodes IUtil. ILibre IUti% Monte
   sur
rootfs         1122304 160548 961756   15% /
tmpfs            63474      1  63473    1% /media
/dev/sda1       128016    349 127667    1% /boot
```

super-bloc Il contient des informations globales sur le système de fichier : taille, nom, type, nom de volume, verrous, tables de gestions...

I-noeud

Les noeuds du système de fichiers sont indexés par des entiers qui les caractérisent : *I-noeud* (inode) :

```
⇝ ls -l /etc/passwd
-rw-r--r--.
    1 root root 1797  1 sept. 17:13 /etc/passwd

⇝ ls -il /home/toto/.bashrc
2883586 -rw-r--r--.
    1 toto novice 124 24 nov. 2011 /home/toto/.
   bashrc
```

attribut

La commande `stat` détaille les attributs des fichiers

```
⇝ stat  /home/toto/.bashrc
  File:  /home/toto/.bashrc
  Size: 124  Blocks: 8  IO Block: 4096  fichier
Device: fd02h/64770d  Inode: 2883586   Links: 1
Access: (0644/-rw-r--r--)
                    Uid: (1002/toto)  Gid: (1002/
   novice)
Context: unconfined_u:object_r:user_home_t:s0
Access: 2012-09-01 17:35:07.718610665 +0200
Modify: 2011-11-24 23:00:34.000000000 +0100
Change: 2012-09-01 17:05:20.838474864 +0200
 Birth: -
```

taille, bloc

Dans un terminal, lancer les commandes :

```
touch /tmp/x
watch stat /tmp/x
```

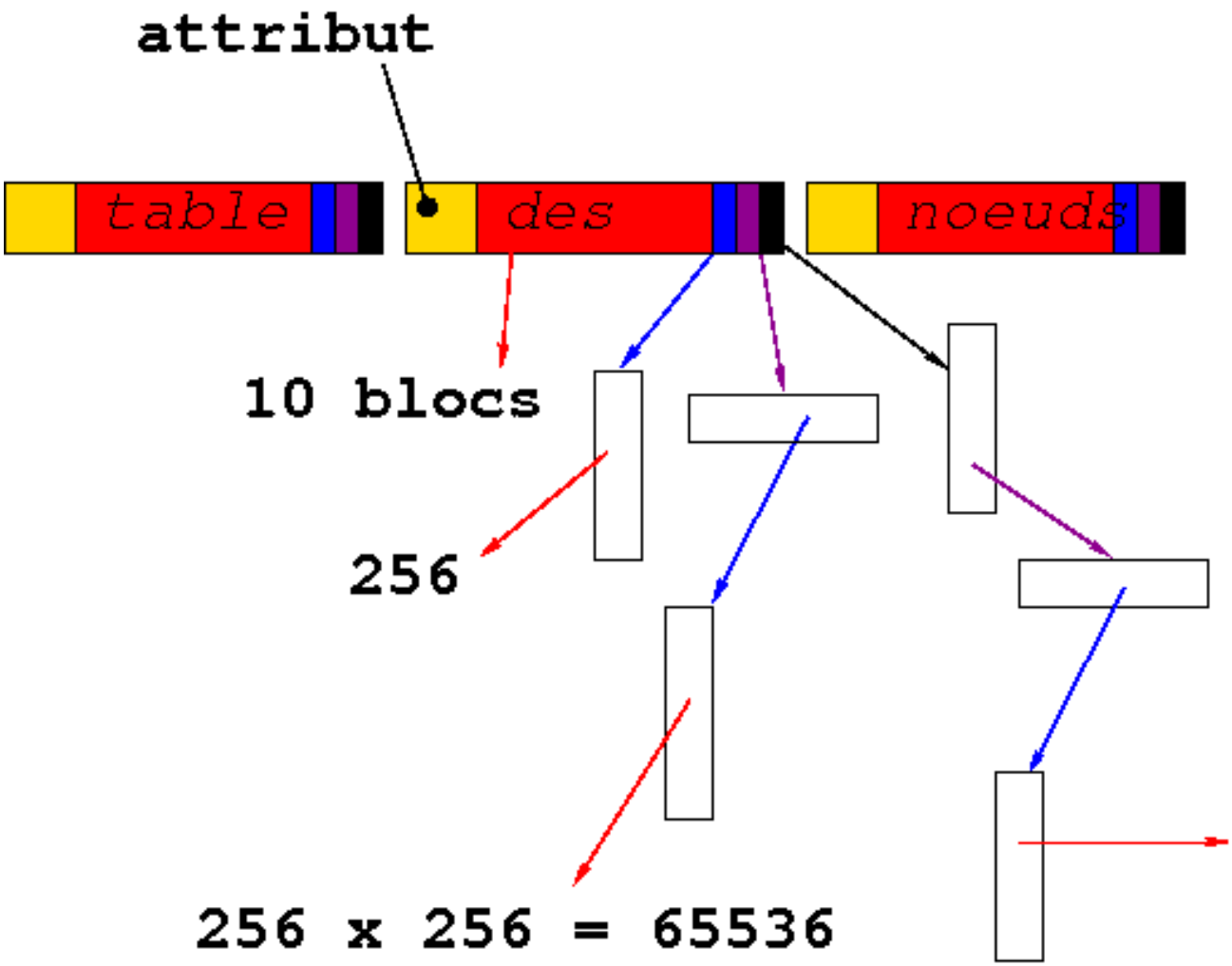
Dans un autre terminal

```
echo hello > /tmp/x
```

Observer le premier terminal. Continuer avec :

```
for(( i=0; i<12; i++)) ; do
    cat /tmp/x /tmp/x > /tmp/y ;
    mv /tmp/y /tmp/x; sleep 3;
done
```

gestion des blocs (ext)



256 x 256 x 256 = 16777216

Limite

nombre de fichiers = nombre d'inoeuds.

```
⇝ b=256
⇝ let t=10+b+b*b+b*b*b
⇝ echo $t
16843018
⇝ echo "l($t)" | bc -l
16.63944676687317765852
```

```
$ bc -l << THIS
> b=256
> t=10+b+b^2+b^3
> l(t)/l(2)
> THIS
24.00564733370382901169
```

Les types de noeud

type	english	création	destruction	
fichier	file		`rm`	
répertoire	directory	`mkdir`	`rm -r`	
lien symbolique	link	`ln -s`	`rm`	
prise	socket	`socket`	`close`	IPC
tube nommé	pipe	`mkfifo`	`rm`	IPC
périphérique	caractére			
périphérique	bloc			

Quelques noeuds

```
⇝ find /dev -type b -name 's*'
/dev/sda3
⇝ tty
/dev/pts/1
⇝ ls -l $(tty)
crw--w----. 1 drmichko tty ...  /dev/pts/1
⇝ ls -l /bin/awk
lrwxrwxrwx. 1 root root 4  ... /bin/awk -> gawk
⇝ ls -ld /*
drwxr-xr-x.  19 root root  3480  ...  /dev
drwxr-xr-x.   6 root root  4096  ...  /home
dr-xr-xr-x.  19 root root 12288  ...  /lib
drwxr-xr-x.   2 root root  4096  ...  /mnt
dr-xr-xr-x. 146 root root     0  ...  /proc
```

commande fichier

- ls, find, locate
- mkdir, cd, popd, pushd, dir
- basename, dirname, pwd, which
- touch, cp, mv
- tar : archivage
- rsync, unison : sauvegarde, git

des statistiques :

- df,du : stat. systèmes de fichiers, disque.
- lsof : liste des fichiers ouverts
- fuser : usage des fichiers

administratives :

- ln : lien
- mount : montage
- fdisk, parted : manipuler les partitions
- fsck : contrôler, réparer

commande fichier

- file, type
- cmp, comm, diff
- cat, tac, tee
- more, less
- tail, head
- nl, wc,
- grep, strings
- od, hexdump
- sort, uniq
- split, csplit
- join, paste, fold
- col, tr, iconv, cut, awk, sed
- gedit, vi, emacs , aspell

compilation

- `indent` :
- `make` : maintenance
- `gcc` : compilateur gnu
- `grof` : profileur
- `gcov` : couverture
- `gdb` : debugguer
- `valgrind` : fuite mémoire
- `flex` : analyseur lexical
- `bison` : compilateur de compilateur

[abc du c]

Les répertoires

Les noms de fichiers sont stockés dans les répertoires :

inode	taille	nom
3804070	12	.
3538945	12	..
3804071	20	hello.txt
3804072	4052	Hello.txt

```
⇝ mkdir tmp
⇝ echo 'hello world' > tmp/hello.txt
⇝ echo 'hello world' > tmp/Hello.txt
♯ echo -e 'ls /drmichko/tmp\nquit'
      | debugfs /dev/mapper/VolGroup-lv_home
debugfs: ls /drmichko/tmp
 3804070 (12) .    3538945 (12) ..    3804071
    (20) hello.txt
```

ajout, suppression des entrées

On ajoute un fichier :

```
⇝ echo 'hello world' > tmp/nouveau.txt



♯ echo -e 'ls /drmichko/tmp\nquit'
      | debugfs /dev/mapper/VolGroup-lv_home
debugfs: ls /drmichko/tmp
 3804070 (12) .    3538945 (12) ..    3804071
   (20) hello.txt
 3804072 (20) Hello.txt    3804067 (4032)
   nouveau.txt
```

On efface un autre :

```
⇝ rm tmp/hello.txt
```

src/dir.c

```
1  #include <string.h>
2  int main()
3  {   int count;
4      struct dirent *fic ;
5      DIR *dir ;
6      char *path, *tok;
7      path=getenv( "PATH" );
8      tok = strtok(path, ":");
9      do {
10         dir = opendir( tok );
11         count=0;
12         while ((fic = readdir(dir)) )
13             if ( DT_REG == fic->d_type) count++;
14     printf ("%s -> %d files\n", tok, count);
15     closedir(dir);
```

src/dir.c

```
/usr/kerberos/sbin -> 5 files
/usr/kerberos/bin -> 14 files
/usr/local/bin -> 4 files
/usr/bin -> 2273 files
/bin -> 115 files
/usr/local/sbin -> 0 files
/usr/sbin -> 362 files
/sbin -> 207 files
/home/pl/bin -> 53 files
/home/pl/script -> 36 files
```

dir.sh

```
1 #!/bin/bash
2 echo $PATH | tr ':' '\n' | while read path
3       do
4    count=$( ls -l $path | grep -cE '^-' )
5    echo $path '->' $count files
6       done
```

- `man 3 readdir`
- `man tr`, `grep`
- `help echo`, `help while`
- 2 solutions parmi d'autres !

[comparaisons] : `python`, `perl`, `bash`, `C`.

Les tubes nommés

```
⇝ whoami
michko
⇝ mkfifo /tmp/canal
⇝ ls -l /tmp/canal
prw-rw-r--. 1 michko michko 0  3 sept. 18:55 /
    tmp/canal
⇝ cat > /tmp/canal
```

michko écrit dans le tube...

```
⇝ whoami
toto
⇝ cat < /tmp/canal
```

toto lit dans le tube.

netcat

La commande `nc`, le couteau suisse du **net**, permet d'écouter sur une *prise* réseau :

```
⇝ nc -l 31415
```

ou bien d'envoyer des données vers une *socket* distante :

```
⇝ nc turing.euphoria.fr 31415
```

dig, nc

```
⇝ nc -lu 31415 | hexdump -C
```

```
⇝ dig -p 31415 www.euphoria.fr @localhost
```

dig, nc

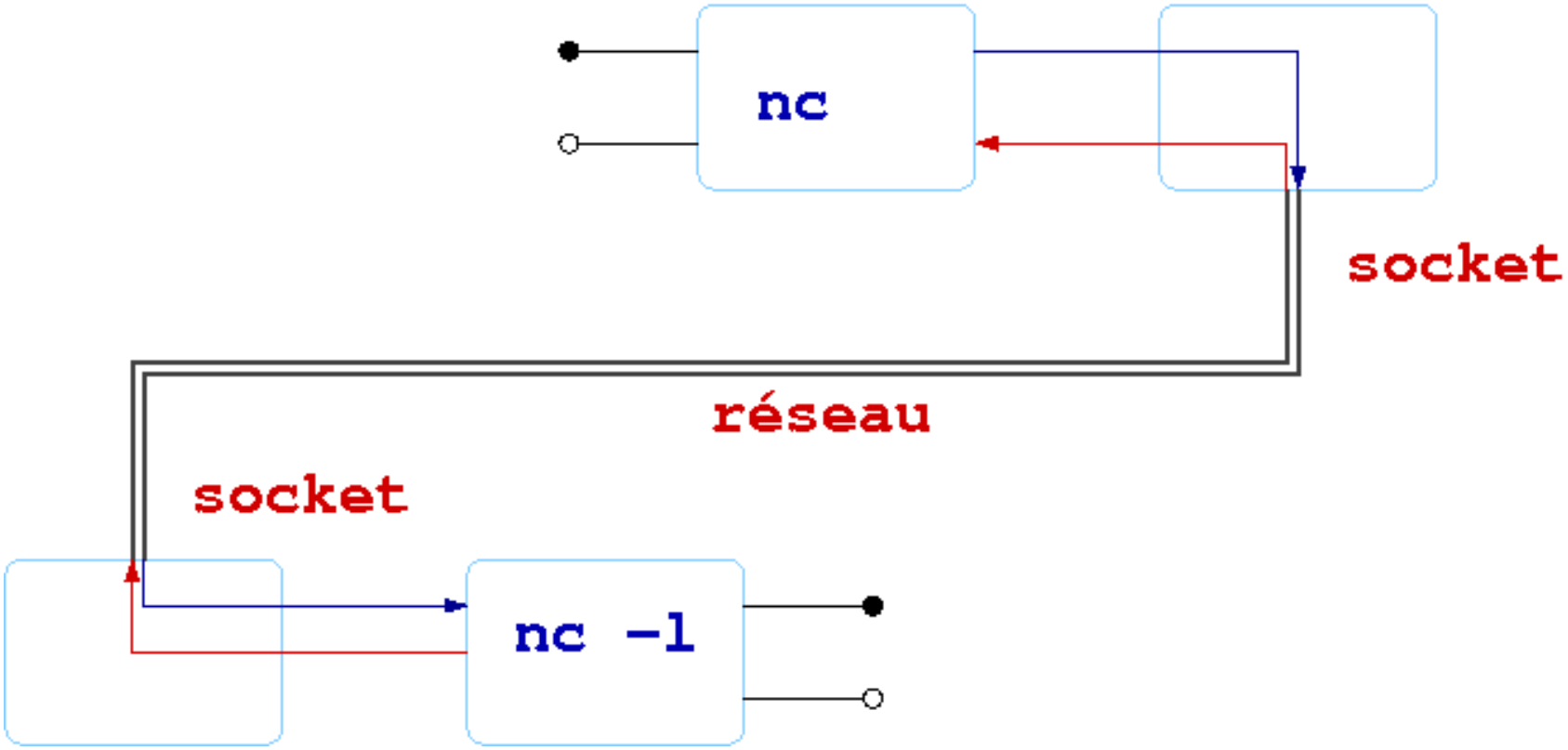
```
⇝ nc -lu 31415 | hexdump -C
```

```
⇝ dig -p 31415 www.euphoria.fr @localhost
```

```
00000000  6a 48 01 00 00 01 00 00
          00 00 00 00 03 77 77 77
          |jH.........www|
00000010  08 65 75 70 68 6f 72 69
          61 02 66 72 00 00 01 00
          |.euphoria.fr....|
```

```
; <<>> DiG 9.7.3-P1-RedHat-9.7.3-2.P1.fc13
  <<>> -p31415 www.euphoria.fr @localhost
;; global options: +cmd
;; connection timed out; no servers could be
   reached
```

client/serveur



gnuchess

```
⇝ gnuchess
GNU Chess 5.08
Adjusting HashSize to 1024 slots
Transposition table:  Entries=1K Size=40K
Pawn hash table: Entries=0K Size=28K
White (1) : e2e4
1. e2e4

black  KQkq  e3
r n b q k b n r
p p p p p p p p
. . . . . . . .
. . . . . . . .
. . . . P . . .
```

gnu chess service

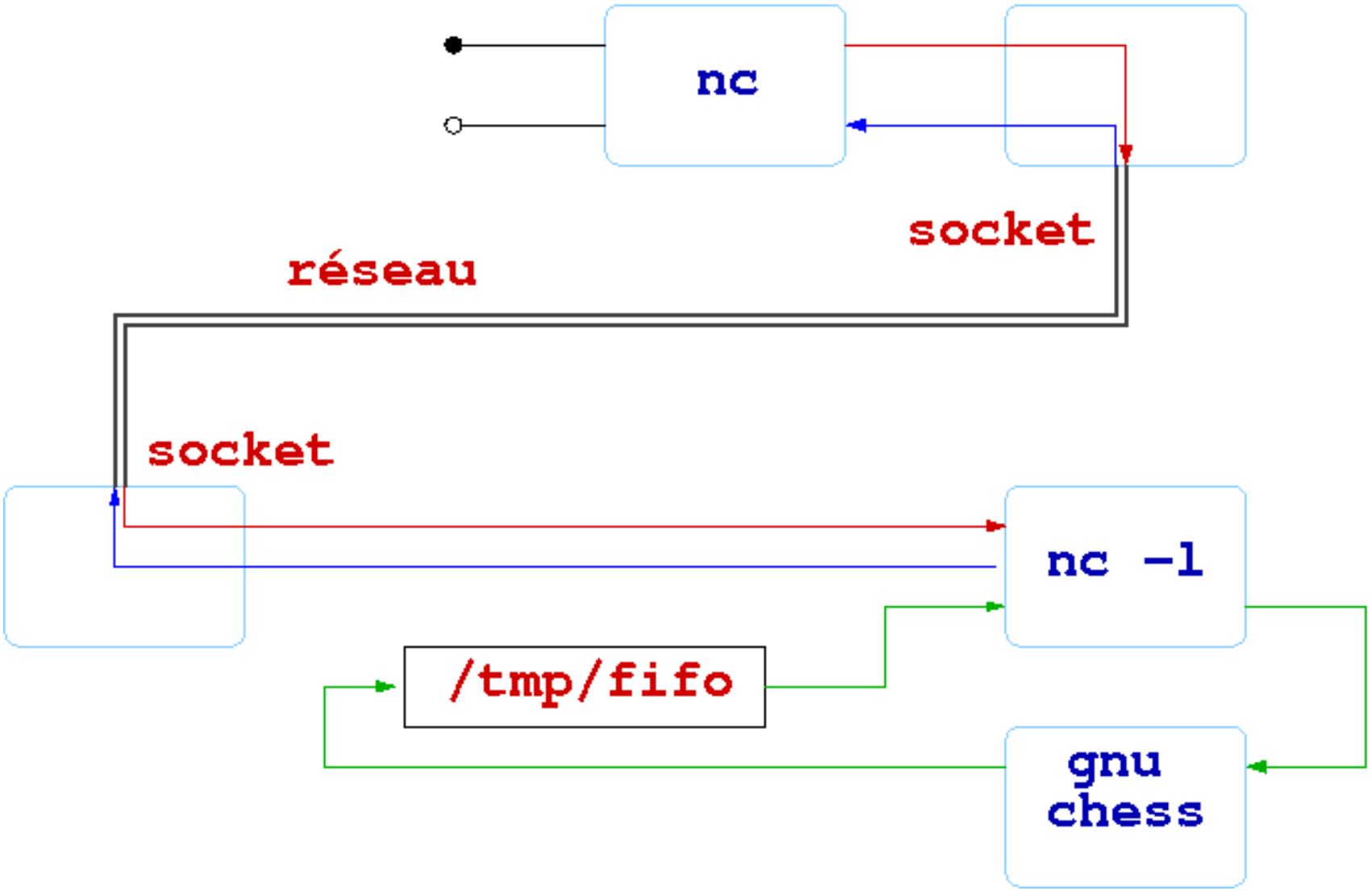
Installation du serveur :

```
⇝ mkfifo /tmp/fifo
⇝ nc -l 31415 < /tmp/fifo
                | gnuchess > /tmp/fifo
```

connexion d'un client :

```
⇝ nc 31415
```

gnu chess service



client

```
⇝ nc turing.euphoria.fr 31415
GNU Chess 5.08
Adjusting HashSize to 1024 slots
Transposition table: Entries=1K Size=40K
Pawn hash table: Entries=0K Size=28K
White (1) : e2e4
1. e2e4

black KQkq e3
r n b q k b n r
p p p p p p p p
. . . . . . . .
. . . . . . . .
. . . . P . . .
```

commande réseau

- `ftp`, `sftp`, `scp`
- `ssh`
- `telnet`
- `wget`, `curl`
- `lynx`, `links`

diagnostic :

- `ping`, `host`, `dig`
- `route`, `ifconfig`,`ip`
- `traceroute`, `tracepath`
- `netstat`

7 - permission

- droit fichier
- changement
- droit répertoire
- droits spéciaux
- setuid bit
- faille

permission

Un i-noeud est la propriété d'un utilisateur et d'un groupe :

```
⇝ chown toto:novice /tmp/canal
⇝ ls -l /tmp/canal
prw-rw-r--. 1 toto novice 0  ... /tmp/canal
⇝ stat /tmp/canal
  File: /tmp/canal
  Size: 0 Blocks: 0 IO Block: 4096 FIFO
Device: fd01h/64769d Inode: 787380 Links: 1
Access: (0664/prw-rw-r--)
   Uid: ( 1002/toto)   Gid: ( 1002/novice)
```

drapeaux

Les permissions sont décrites par un nombre de 4 chiffres en base octale et forment 12 drapeaux de droit.

s	S	t	u	u	u	g	g	g	o	o	o
0	0	0	1	1	1	1	0	1	1	0	0
			r	w	x	r	-	x	r	-	-

TABLE – droit 0754

Le chiffre dominant concerne les droits spéciaux, les suivant les droits utilisateur, groupe et autre.

affichage

s	S	t	u	u	u	g	g	g	o	o	o
1			r	w	s	r	w	x	r	w	x
	1		r	w	x	r	w	s	r	w	x
		1	r	w	x	r	w	x	r	w	t

TABLE – affichage des droits spéciaux

```
⇝ ls -l /bin/passwd
-rwsr-xr-x. 1 root root 2715 22 jui 02:51 /bin/
    passwd
⇝ ls -l /usr/bin/wall
-r-xr-sr-x. 1 root tty 10588 18 fev 2013 /usr/
    bin/wall
⇝ ls -ld /tmp
drwxrwxrwt 14 root root 480 18 sept. 18:22 /tmp
```

droit fichier

Les drapeaux **rwx** sont définis par le propriétaire du fichier pour lui, le groupe et le reste du monde. Le propriétaire peut changer les permissions.

- **r** : fichier ouvert en lecture, copiable
- **w** : fichier ouvert en écriture, vidable
- **x** : exécutable

Les droits **suid** et **sgid** affectent ces droits.

manipulation

La commande chmod permet d'ajuster les droits :

```
⇝ chmod u+x go-x /tmp/canal
```

de les sceller numériquement.

```
⇝ chmod 700 /tmp/canal
```

mode récursif :

```
⇝ chmod -R a-w /home/toto/pub
```

priorité ugo

```
# useradd -g bar foo;useradd drm;useradd ken
# ls -l /home
drwx------   4 drm   drm     4096  ... drm
drwx------   4 foo   bar     4096  ... foo
drwx------   4 ken   ken     4096  ... ken
# usermod -a -G bar drm    ;  groups drm
drm : drm bar
# echo world > /tmp/hello
# chown foo:bar /tmp/hello ; ls -l /tmp/hello
-rw-r--r-- 1 foo bar 6  ... /tmp/hello
# chmod ou-r /tmp/hello
```

```
# sudo -u ken cat /tmp/hello
cat: /tmp/hello: Permission non accordee
# sudo -u foo cat /tmp/hello
cat: /tmp/hello: Permission non accordee
# sudo -u drm cat /tmp/hello
world
```

droit répertoire

- **r** : lecture du contenu du répertoire.
- **w** : permet la création, la suppression et le changement de noms
- **x** : autorise l'accès (ou ouverture) au répertoire.

Le **sticky bit** et le **sgid** affectent ces droits.

x/dir

```
1 #!/bin/bash -v
2 mkdir -p /tmp/x/y
3 echo hello > /tmp/x/y/z.txt
4 cat /tmp/x/y/z.txt
5 hello
6 chmod u-x /tmp/x
7 cat /tmp/x/y/z.txt
8 cat: /tmp/x/y/z.txt: Permission non accordee
9 chmod u+x /tmp/x
```

w/dir

```
⇝ su pat ; ls -ld /tmp/x/y
drwxrwxr-x 2 pl pl 4096 31 juil. 09:50 /tmp/x/y
⇝ yes | rm /tmp/x/y/z.txt
rm: impossible de supprimer /tmp/x/y/z.txt
```

w/dir

```
⇝ su pat ; ls -ld /tmp/x/y
drwxrwxr-x 2 pl pl 4096 31 juil. 09:50 /tmp/x/y
⇝ yes | rm /tmp/x/y/z.txt
rm: impossible de supprimer /tmp/x/y/z.txt
```

```
⇝ chmod o+w /tmp/x/y
⇝ ls -al /tmp/x/y
drwxrwxrwx 2 pl pl 4096 31 juil. 09:58 .
drwxrwxr-x 3 pl pl 4096 31 juil. 09:40 ..
-rw-rw-r-- 1 pl pl    6 31 juil. 09:58 z.txt
```

w/dir

```
⇝ su pat ; ls -ld /tmp/x/y
drwxrwxr-x 2 pl pl 4096 31 juil. 09:50 /tmp/x/y
⇝ yes | rm /tmp/x/y/z.txt
rm: impossible de supprimer /tmp/x/y/z.txt
```

```
⇝ chmod o+w /tmp/x/y
⇝ ls -al /tmp/x/y
drwxrwxrwx 2 pl pl 4096 31 juil. 09:58 .
drwxrwxr-x 3 pl pl 4096 31 juil. 09:40 ..
-rw-rw-r-- 1 pl pl    6 31 juil. 09:58 z.txt
```

```
$ echo boom > /tmp/x/y/z.txt
bash: /tmp/x/y/z.txt: Permission non accordee
$ rm -f  /tmp/x/y/z.txt
$ ls -a  /tmp/x/y                    (OK)
```

masque de création

Un *masque* est appliqué par défaut lors de la création des fichiers. Il indique les droits à retirer (masque) ou à conserver (logique) :

```
⇝ umask ; umask -S
0002
u=rwx,g=rwx,o=rx
⇝ echo test > /tmp/test
⇝ ls -l test
-rw-rw-r--. 1 pl pl 5 3 sept. 19:45 /tmp/test
⇝ echo $(( 666 - 666 & 002 ))
664
⇝ stat --format="%a %A" /tmp/test
664 -rw-rw-r--
```

masque de création

```
1  #!/bin/bash -v
2  echo 'int main(void) {return 0;}' > vrai.c
3  umask -S
4  u=rwx,g=rwx,o=rx
5  gcc vrai.c
6  ls -l a.out
7  -rwxrwxr-x 1 pl pl 4542 31 juil . 08:25 a.out
8  umask 077
9  gcc vrai.c
10 ls -l a.out
11 -rwx------ 1 pl pl 4542 31 juil. 08:25 a.out
```

bit collant

A l'origine, le sticky bit indique au système de conserver en mémoire le code. Cette terminologie historique est réutilisée pour qualifier les doits sur un répertoire :

```
⇝ ls -ld /tmp
drwxrwxrw.t 13 root root 4096  5 sept. 12:36 /
    tmp/

⇝ stat --format="%F %a %A" /tmp
repertoire 1777 drwxrwxrwt
```

sticky bit du répertoire allumé

Un utilisateur peut manipuler une arborescence dans ce répertoire : créer et effacer les fichiers qui lui appartiennent.

sticky bit

```
⇝ ls -ld /{tmp,TMP}
drwxrwxrwt. 103 root root 12288 ... /tmp
drwxrwxrwx    2 root root  4096 ... /TMP
⇝ echo hello > /tmp/msg.txt
⇝ echo hello > /TMP/msg.txt
⇝ ls -l /{tmp,TMP}/msg.txt
-rw-rw-r-- 1 pl pl 6 ... /tmp/msg.txt
-rw-rw-r-- 1 pl pl 6 ... /TMP/msg.txt
⇝ su pat
$ rm /TMP/msg.txt
$ rm /tmp/msg.txt
rm : operation non permise
```

droit suid

Dans certaines circonstances, un utilisateur doit pouvoir exécuter des binaires avec des privilèges **root**. Typiquement, **toto** peut changer son mot de passe avec `passwd` et donc ecrire dans **/etc/shadow** ! ?

```
⇝ ls -l /etc/shadow
----------. 1 root root 1164  ... /etc/shadow
⇝ ls -l /usr/bin/passwd
-rwsr-xr-x. 1 root root 25992 ... /usr/bin/
    passwd
⇝ stat --format="%F %a %A" /usr/bin/passwd
fichier 4755 -rwsr-xr-x
```

- `passwd` est un binaire **suid**
- ouvert en exécution
- fournit la capacité root (euid).

binaires suid

```
⇝ find /usr/bin -perm +4000
/usr/bin/chsh
/usr/bin/chage
/usr/bin/passwd
/usr/bin/mount
/usr/bin/crontab
/usr/bin/sudo
/usr/bin/umount
/usr/bin/at
/usr/bin/su
```

droit sgid répertoire

Le bit **sgid** sur un fichier exécutable s'applique au groupe : s masque les droits x du groupe, S sinon.

- Dans la cas d'un partage de dossier, tous les membres d'un groupe souhaitent modifier ou supprimer les contributions des membres du groupe.
- Tout fichier et répertoire créé dans un répertoire **sgid** appartient au groupe de celui-ci et non au groupe de l'utilisateur qui crée l'élément.
- Les sous-répertoires héritent du bit **sgid**, sans rétroaction !

```
♯ groupadd test ; mkdir /home/partage
♯ chown root:test /home/partage
♯ usermod -a -G test toto
⇝ groups
 novice test
⇝ echo hello > /home/partage/toto
⇝ ls -al  /home/partage *o*
-rw-r--r--. 1 toto novice   18  16:52 toto
♯ chmod g+s /home/partage
⇝ stat --format="%F %a %A" /home/partage
repertoire 2775 drwxrwsr-x
⇝ echo hello  > /home/partage/world
⇝ ls -al  /home/partage *o*
-rw-r--r--. 1 toto novice   18  16:52 toto
-rw-r--r--. 1 toto test      6  16:53 world
```

uid, euid

Un processus possède un **uid** réel et un **uid** effectif sur lequel sont basé les privilèges du processus. En général

$$UID = EUID$$

- l'**uid** réel est celui du processus qui lance la commande.
- l'**uid** effectif est (sous condition) celui du propriétaire de la commande.

Les permissions sont basées sur l'**euid**.

system vs execve

```
1  int main()
2  {
3  char *argv[2];
4  argv[0] = "id";
5  argv[1] = 0;
6
7  system("id");
8
9  execve("/usr/bin/id", argv, 0);
10
11 return 0;
12 }
```

- `system` exécution par le shell.
- `execve` appel système.

system vs execve

```
sudo gcc -Wall setuidexec.c -o setuidexec.exe
./setuidexec.exe | cut -d ',' -f1,2
uid=501(pl) gid=501(pl) groupes=501(pl),10(wheel
    )
uid=501(pl) gid=501(pl) groups=501(pl),10(wheel)
sudo chmod u+s setuidexec.exe
./setuidexec.exe | cut -d ',' -f1,2
uid=501(pl) gid=501(pl) groupes=501(pl),10(wheel
    )
uid=501(pl) gid=501(pl) euid=0(root) groups=0(
    root),10(wheel)
```

- `system` euid $\rightsquigarrow$ uid (`bash` n'est pas **suid**).
- `execve` euid = root. (appel système)

euid.c

```
1  #include <unistd.h>
2  #include <stdlib.h>
3  int main( void ) {
4  FILE *src = NULL;
5  system("ls -alt euid.exe");
6  printf ("\nuid=%d euid=%d\n",getuid (),geteuid ());
7  system("whoami");
8  system("wc -L /etc/shadow");
9                                  // -L : plus longue ligne ?
10 src = fopen("/etc/shadow", "r");
11 printf ("%p", src );
12 fclose ( src );
13 return 0;
14 }
```

system, execve

```
⇝ sudo gcc -Wall euid.c -o euid.exe
⇝ sudo chmod u+s euid.exe
⇝ ./euid.exe
-rwsr-xr-x 1 root root 5326 21:37 euid.exe

uid=501 euid=0
pl
wc: /etc/shadow: Permission non accordee
0x859a008
```

- `bash` : perte de **euid**.
- `fopen` : exécuté avec **euid** root.

uid, euid

```
1  #include <sys/types.h>
2  #include <stdio.h>
3  #include <unistd.h>
4  #include <stdlib.h>
5  int main( void )
6  {
7  FILE *src;
8  printf ("\nuid=%d euid=%d\n",getuid (),geteuid ());
9  src = fopen( "uid.txt", "r" );
10 if ( ! src )
11     perror ( "interdit " );
12 fclose (src);
13 return 0;
14 }
```

uid, euid

```
⇝ gcc -Wall uid.c -o uid.exe
⇝ sudo chown nobody:nobody uid.exe
⇝ sudo cp -p uid.exe uid.sss
⇝ sudo chmod u+s  uid.sss
⇝ touch uid.txt ; ls -l uid.[est]*
-rwxrwxr-x  1 nobody nobody 5237 ...  uid.exe
-rwsrwxr-x. 1 nobody nobody 5237 ... uid.sss
-rw--w----. 1 pl     pl        0 ...  uid.txt
⇝ id
uid=501(pl) gid=501(pl) groupes=501(pl),10(wheel)
⇝ ./uid.exe
uid=501 euid=501
⇝ ./uid.sss
uid=501 euid=99
interdit: Permission denied
```

setuid.c

```
1  #include <sys/types.h>
2  #include <stdio.h>
3  #include <unistd.h>
4  #include <stdlib.h>
5  int main( void ) {
6  system("ls -alt setuid.exe");
7  printf ("\nuid=%d euid=%d\n",getuid (),geteuid ());
8  system("whoami");
9  system("ls /root");
10 setuid(0);
11 printf ("\nuid=%d euid=%d\n",getuid (),geteuid ());
12 system("whoami");
13 system("ls /root");
14 return 0;
15 }
```

setuid.exe

```
⇝ sudo gcc -Wall setuid.c -o setuid.exe
⇝ sudo chmod u+s setuid.exe
./setuid.exe
-rwsr-xr-x 1 root root 5187 ...  setuid.exe
uid=501 euid=0
pl
ls: impossible d'ouvrir le repertoire
    root: Permission non accordee
uid=0   euid=0
root
anaconda-ks.cfg  Desktop  install.log
```

`setuid` : le processus utilise son **euid** pour changer **uid** !

faille

```
1  #include <sys/types.h>
2  #include <stdio.h>
3  #include <unistd.h>
4  #include <stdlib.h>
5
6  int main( int argc, char *argv[] )
7  {
8  char cmd[64];
9
10 sprintf ( cmd, "wc -l %s", argv[1] );
11 setuid(0);
12 system( cmd );
13 return 0;
14 }
```

exploit

```
⇝ ./boom.exe /etc/passwd
35 /etc/passwd
```

exploit

```
⇝ ./boom.exe /etc/passwd
35 /etc/passwd
```

```
⇝ ./boom.exe /etc/shadow
35 /etc/shadow
```

exploit

```
⇝ ./boom.exe /etc/passwd
35 /etc/passwd
```

```
⇝ ./boom.exe /etc/shadow
35 /etc/shadow
```

```
⇝ ./boom.exe "/etc/passwd; grep pl /etc/shadow"
35 /etc/passwd
pl:$6$uHL8DbWBehKEl0Rb$dZZ/3O2uhDlMkkrnjlG/q8Lo.
    J8
CZKPqZAc7j7EFFBo6o4QTw/UlWK.YWhjvgKRqquMo.
    aKJEtlG3
YEYPyg5T.:15439:0:99999:7: : :
passwd
```

Injection de code

faille L'absence de contrôle sur les données passées au exécutable pose toujours des soucis, dans le cas des *suid* c'est un sérieux problème de sécurité !

Injection de code

faille L'absence de contrôle sur les données passées au exécutable pose toujours des soucis, dans le cas des *suid* c'est un sérieux problème de sécurité !

```
⇝ ./boom.exe 'boom.c;/bin/sh'
```

Injection de code

faille L'absence de contrôle sur les données passées au exécutable pose toujours des soucis, dans le cas des *suid* c'est un sérieux problème de sécurité !

```
⇝ ./boom.exe 'boom.c;/bin/sh'
```

```
15 boom.c
sh-4.2#
```

Injection de code

faille L'absence de contrôle sur les données passées au exécutable pose toujours des soucis, dans le cas des *suid* c'est un sérieux problème de sécurité !

```
⇝ ./boom.exe 'boom.c;/bin/sh'
```

```
15 boom.c
sh-4.2#
```

```
sh-4.2# whoami
root
```

- shell root !

attaque sushi

```
#!/bin/bash
echo $UID $EUID
⇝ sudo chown root:root  suid.sh
⇝ sudo chmod u+s suid.sh
⇝ ls -l suid.sh
-rwsrwxr-x 1 root root 28 19 sept. 06:10 suid.sh
⇝ ./suid.sh
501 501
```

Les possibilités de failles dans un script **suid** seraient encore plus nombreuses : attaques **sushi**. Pour des raisons de sécurité :

- dans un script **shell** : **euid** := **uid**
- modification de fichier $\Longrightarrow$ perte du **suid**-bit

shellshock

```
1  #include <sys/types.h>
2  #include <stdio.h>
3  #include <unistd.h>
4  #include <stdlib.h>
5  // shellshockable : facile
6  int main( void )
7  {
8  char tmp[128], cmd[128];
9  sprintf (tmp, "sed -n '/:%d:/s/:.*//p' /etc/passwd", getuid
       () );
10 sprintf (cmd, "grep $( %s ) /etc/shadow", tmp);
11 puts("cryptogramme:");
12 setuid(0);
13 system( cmd );
14 return 0;
```

shellshock

```
SYSTEM(3)          Manuel du programmeur Linux
NOM
       system - Executer une commande shell
SYNOPSIS
       #include <stdlib.h>
       int system(const char *command);
DESCRIPTION
La fonction system() execute la commande indiquee
dans command en appelant /bin/sh -c command, et
revient apres l'execution complete de la commande.
NOTES
N'utilisez jamais system() dans un programme
avec les privileges Set-UID ou Set-GID. Des
variables d'env. avec des valeurs etranges peuvent
etre utilisees pour corrompre le systeme.
```

8 - processus

- exécutable vs processus
- environnement
- variable du shell
- contexte d'exécution
- commande processus
- gentillesse
- ressource
- sous-processus
- job
- processus léger

processus ?

Il s'agit d'un fichier binaire en cours d'exécution ! Il y a un processus actif par processeur, le noyau partage et ordonne leur exécution.

processus ?

Il s'agit d'un fichier binaire en cours d'exécution ! Il y a un processus actif par processeur, le noyau partage et ordonne leur exécution.

- `bg`, `jobs`, `time`, `kill`
- `ps`, `pstree`, `/usr/bin/time`, `top`

ps

La commande `ps` renseigne sur les attributs

PID, PPID, TID (SPID), UID, EUID ...

et l'état d'un processus

	état
D	Uninterruptible sleep (usually IO)
R	Running or runnable (on run queue)
S	Interruptible sleep
T	Stopped, traced
X	dead (should never be seen)
Z	Defunct ("zombie") process

format BSD : $<$ haute priorité, N basse, L pages vérouillées, s session leader, l multithreadé, $+$ en avant plan.

exemple

```
⇝ ps -e -ostate,pid,cmd | grep -v S
R 12447 ps -e -ostate,pid,cmd
```

exemple

```
⇝ ps -e -ostate,pid,cmd | grep -v S
R 12447 ps -e -ostate,pid,cmd
```

```
⇝ ps --ppid 1 -oppid,pid,user,cmd
1   338 root /usr/lib/udev/udevd
1   339 root /usr/lib/systemd/systemd-journald
1   531 root /usr/lib/systemd/systemd-logind
1   538 root /usr/sbin/atd -f
1   540 root /usr/sbin/lxdm-binary -nodaemon
1   577 root /usr/sbin/crond -n
1   867 root /sbin/dhclient -H dellpl -1 ...
1   934 root /usr/sbin/sshd -D
1  1187 pl   xterm
1  1256 root sendmail: accepting connections
1  1272 smmsp sendmail: Queue runner@01:00:00 ...
1  1439 pl ssh -Nf pl@nil.fr -L2222:10.2.73.86:22
```

Environnement

Un processus hérite des variables d'environnement. Elles sont passées par la pile, tout comme le résultat de la commande.

```
1 ...
2 int main( int argc, char *argv[], char* env[] )
3 {
4 ...
5 return SUCCES;
6 }
```

- code de retour = 0 : SUCCES

capture de l'environnement

```
1  #include <stdio.h>
2  int main( int arg, char** argv , char** envr )
3  {
4  int res = 0;
5  while ( *argv )
6      printf ("\narg:%s", *argv++);
7  while ( *envr ) {
8      printf ("\nenv:%s", *envr++);
9      res++;
10     }
11 return res;
12 }
```

code de retour

```
:: gcc -Wall env.c
:: ./a.out hello
arg:./ a.out
arg: hello
env:HOSTNAME=dellpl
env:TERM=xterm
env:SHELL=/bin/bash
env:HISTSIZE=1000
env:USER=pl
...
:: echo $?
41
:: env -i ./a.out hello
arg:./ a.out
arg: hello
```

variable usuelle

- std : LOGNAME, HOME PATH, SHELL, TERM, USER, UID, GID,...
- `bash` : PS1, PS2, HISTFILE, SHELLOPT, BASH_VERSION,...
- paramètres de `bash` : $, 0–9, −, ♯, ∗, @, !, ?, _

```
⇝ help variable | grep -E '(PS|SH)'
    BASH_VERSION Version information for this
   Bash.
    PS1  The primary prompt string.
    PS2  The secondary prompt string.
    SHELLOPTS A colon-separated list of enabled
   shell options.
```

```
⇝ env | grep -e '^P'
PATH=/usr/kerberos/sbin:/usr/kerberos/bin:/usr/
   local/bin:/usr/bin:/bin:/usr/local/sbin:/usr/
```

transmission

- les variables du shell sont transmises aux sous-shell.
- `export` permet la transmission aux processus fils.

```
1 XYZ='hello world'
2 env | grep XYZ
3 export XYZ
4 env | grep XYZ
5 XYZ=hello world
6 export -p XYZ
7 export -n XYZ
8 env | grep XYZ
```

commandes

- `help variable` : variables usuelles.
- `env` : environnement d'exécution
- `export` : transmission automatique
- `unset` : démonte une variable du shell.
- `which`
- `declare`

```
⇝ grep PATH ~/.bash_profile
PATH=$PATH:$HOME/bin:$HOME/script
export MANPATH=$MANPATH:$HOME/texinput
```

- `set` : configuration de `bash`

declare

```
⇝ X=(1 2 3)
⇝ declare -p X
declare -a X='([0]="1" [1]="2" [2]="3")'
declare z=15
⇝ declare -p z
declare -i z="15"
⇝ declare str=hello
⇝ declare -p str
declare -- str="hello"
⇝ declare -A t
⇝ t['un']=1 ; t['deux']=2; t['trois']=3
declare -p t
declare -A t='([un]="1" [deux]="2" [trois]="3"
  )'
```

option du shell

```
1 #!/bin/bash
2 PS4=::
3 help set | grep -E '^[ ]*-(e|x)'
4 set -x
5 echo foo > /etc/shadow
6 set -e
7 echo bar > /etc/shadow
8 echo hello world
```

option du shell

```
1 #!/bin/bash
2 PS4=::
3 help set | grep -E '^[ ]*-(e|x)'
4 set -x
5 echo foo > /etc/shadow
6 set -e
7 echo bar > /etc/shadow
8 echo hello world
```

```
1         -e  Exit immediately if a command exits with a non-
    zero status.
2         -x  Print commands and their arguments as they are
    executed.
3 :: echo foo
4 ./set.sho: line 5: /etc/shadow: Permission non accordee
```

contexte d'exécution

Le contexte d'exécution,

- registre
- mémoire
- descripteur

et l'ensemble des données concernant un processus sont disponibles dans le répertoire `/proc/self` ou `/proc/pid`.

proc.sho

```
1 #!/bin/bash -v
2 ls  -d /proc/self/* | sed 's/\/proc.self \///g'\
3     | tr '\n' ' ' | fold -w 54
4 #
5 cat /proc/self /environ \
6   | tr '\000' '\n' | grep -i name
7 cat /proc/self /status
```

proc.out

```
1  #!/bin/bash -v
2  ls  -d /proc/self/* | sed 's/\/proc.self \///g'\
3      | tr '\n' ' ' | fold -w 54
4  attr auxv cgroup clear_refs cmdline comm coredump_filt
5  er cpuset cwd environ exe fd fdinfo io latency limits
6  loginuid maps mem mountinfo mounts mountstats net oom_
7  adj oom_score pagemap personality root sched schedstat
8   sessionid smaps stack stat statm status syscall task
9  wchan #
10 cat /proc/self /environ \
11   | tr '\000' '\n' | grep -i name
12 HOSTNAME=microbe.rts.fr
13 USERNAME=pl
14 LOGNAME=pl
15 G_BROKEN_FILENAMES=1
```

status.out

```
1  #!/bin/bash -v
2  cat /proc/self /status
3  Name: cat
4  State:    R (running)
5  Tgid: 2286
6  Pid:  2286
7  PPid: 2284
8  TracerPid:   0
9  Uid:  501   501   501   501
10 Gid:  501   501   501   501
11 Utrace:   0
12 FDSize:   256
13 Groups:   10 501
14 VmPeak:      4084 kB
15 VmSize:      4084 kB
```

/proc/pid

```
1  #include <sys/types.h>
2  #include <stdio.h>
3  #include <unistd.h>
4  #include <stdlib.h>
5
6  int main( int argc, char *argv[] )
7  {
8  char cmd[64];
9  if ( argc > 1 )
10   sprintf (cmd, "cat /proc/%d/%s", getpid() , argv[1]) ;
11 else
12   sprintf (cmd, "ls -al /proc/%d", getpid() );
13 system( cmd );
14 putchar( '\n');
15 return 0;
```

```
⇝ gcc -Wall proc.c -o proc.exe ; ./proc.exe
dr-xr-xr-x.   2 pl pl0  3 sept. 12:19 attr
-rw-r--r--.   1 pl pl 0  3 sept. 12:19 autogroup
-r--------.   1 pl pl 0  3 sept. 12:19 auxv
-r--r--r--.   1 pl pl 0  3 sept. 12:19 cgroup
--w-------.   1 pl pl 0  3 sept. 12:19
   clear_refs
-r--r--r--.   1 pl pl 0  3 sept. 12:19 cmdline
-rw-r--r--.   1 pl pl 0  3 sept. 12:19 comm
-rw-r--r--.   1 pl pl 0  3 sept. 12:19
   coredump_filter
-r--r--r--.   1 pl pl 0  3 sept. 12:19 cpuset
lrwxrwxrwx.   1 pl pl 0  3 sept. 12:19 cwd -> /
   home/pl/UPS/doc
-r--------.   1 pl pl 0  3 sept. 12:19 environ
lrwxrwxrwx.   1 pl pl 0  3 sept. 12:19 exe -> /
   home/pl/UPS/doc/proc.exe
```

Carte mémoire

```
⇝ ./proc.exe maps
08048000-08049000 r-xp 00000000 fd:01 793772 /
    home/pl/UPS/doc/a.out
08049000-0804a000 rw-p 00000000 fd:01 793772 /
    home/pl/UPS/doc/a.out
42c92000-42cb1000 r-xp 00000000 fd:01 163870 /
    usr/lib/ld-2.15.so
42cb1000-42cb2000 r--p 0001e000 fd:01 163870 /
    usr/lib/ld-2.15.so
42cb2000-42cb3000 rw-p 0001f000 fd:01 163870 /
    usr/lib/ld-2.15.so
42cb5000-42e5d000 r-xp 00000000 fd:01 163871 /
    usr/lib/libc-2.15.so
42e5d000-42e5f000 r--p 001a8000 fd:01 163871 /
    usr/lib/libc-2.15.so
```

commande processus

- ps, pstree, top, uptime
- time, time
- nice, ulimit
- gdb, ltrace, strace
- kill, killall
- wait
- jobs, bg, fg
- at, batch
- nohup
- disown

calcul

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  int main( int argc, char*argv[] )
4  {
5  int x = atoi( argv[1] );
6  int y = 1;
7  while ( --x )
8      y = (x * y) % 23;
9  return y;
10 }
```

calc.exe

```
#!/bin/bash -v
gcc -Wall -O2 calc.c -o calc.exe
./calc.exe 100000000 &
sleep 1
ps   -o cmd,pcpu
CMD                                %CPU
evince proc.pdf                     2.7
bash                                0.0
make -B pcpu.out                    0.0
/bin/sh -c ./pcpu.sh >& pcp         0.0
/bin/bash -v ./pcpu.sh              0.0
./calc.exe 100000000                102
ps -o cmd,pcpu                      0.0
```

charge/wait

```
1  #/bin/bash -v
2  cpt=$1
3  uptime | sed 's/load/\nload/'
4  while [ $cpt -gt 0 ]
5  do
6      /usr/bin/time --format="r=%U c=%c t=%E" \
7          ./calc.exe 500000000 &
8      let cpt--
9  done
10 wait
11 date
```

charge/wait

```
 14:11:40 up 7 days, 21:23,  5 users,
load average: 0,15, 0,53, 0,34
r=12.54 c=265 t=0:12.80
r=12.52 c=162 t=0:12.90
r=12.52 c=644 t=0:13.08
r=12.47 c=797 t=0:13.68
ven. sept. 20 14:11:54 CEST 2013
 14:11:54 up 7 days, 21:23,  5 users,
load average: 0,80, 0,66, 0,39
r=12.54 c=958 t=0:19.10
r=12.53 c=928 t=0:19.27
r=12.54 c=1080 t=0:19.30
r=12.53 c=1061 t=0:19.38
r=12.51 c=833 t=0:19.55
r=12.48 c=866 t=0:19.72
```

gentillesse

```
1  #/bin/bash -v
2  cpt=$1
3  gtle =0
4  while [ $cpt -gt 0 ]
5  do
6      /usr/bin/time --format="nice=$gtle cpu=%P ctx=%c
       total=%E" \
7          nice -n $gtle ./calc.exe 500000000 &
8      let cpt--
9      let gtle +=5
10 done
11 lscpu | grep '(s)' | iconv -f UTF-8 -t ASCII -c
12 ps -opid,nice,psr,cmd -C calc.exe
```

gentillesse

```
Mode(s) opratoire(s) des processeurs: 32-bit,
   64-bit
Processeur(s):          4
Liste de processeur(s) en ligne: 0-3
Thread(s) par cur: 2
Cur(s) par socket: 2
Socket(s):              1
  PID   NI PSR CMD
20555    5   3 ./calc.exe 500000000
20557    0   1 ./calc.exe 500000000
20560   10   3 ./calc.exe 500000000
20561   15   3 ./calc.exe 500000000
20562   19   0 ./calc.exe 500000000
nice=0 cpu=97% ctx=107 total=0:12.71
nice=5 cpu=91% ctx=1039 total=0:13.63
```

limites

```
⇝ ./proc.exe limits| sed -E 's/unlimited/*/g;s/[
    \t]+/ /g'
Limit                     Soft   Hard Units
Max cpu time                 *      * seconds
Max file size                *      * bytes
Max data size                *      * bytes
Max stack size        10485760      * bytes
Max core file size           0      * bytes
Max resident set             *      * bytes
Max processes             1024   7911 processes
Max open files            1024   1024 files
Max locked memory        65536  65536 bytes
Max address space            *      * bytes
Max file locks               *      * locks
Max pending signals       7911   7911 signals
```

limit.c

```
1  #include <sys/time.h>
2  #include <sys/resource.h>
3  #include <stdio.h>
4  int main( void )
5  {
6  struct rlimit z;
7  printf ("limites :      souple      maximum\n");
8  getrlimit ( RLIMIT_AS, &z );
9  printf ("mem    :%10lu %10lu\n", z.rlim_cur, z.rlim_max);
10 getrlimit ( RLIMIT_NOFILE, &z );
11 printf ("fic      :%10lu %10lu\n", z.rlim_cur , z.rlim_max);
12 getrlimit ( RLIMIT_NPROC, &z );
13 printf ("proc    :%10lu %10lu\n", z.rlim_cur , z.rlim_max);
14 getrlimit ( RLIMIT_STACK, &z );
15 printf ("pile    :%10lu %10lu\n", z.rlim_cur , z.rlim_max);
```

```
./limit.exe
limites:     souple     maximum
mem     :4294967295 4294967295
fic     :       1024       1024
proc    :       1024       7911
pile    :   10485760 4294967295
cpu     :4294967295 4294967295

Temps UCT limite expire
```

stack.c

```
1  #include <signal.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4  int  n = 0;
5  void go( void )
6  {
7   //printf(" %d\n", n );
8   n++;
9   go( );
10 }
11
12 int main( int argc, char*argv[] )
13 {
14 go( );
15 return 0;
```

stack.out

```
#!/bin/bash -v
ulimit -s
10240
echo -e 'display n\nrun\nquit' > /tmp/gdb.cmd
gdb -q -x /tmp/gdb.cmd ../src/stack.exe\
    | grep SIG -A3
Program received signal SIGSEGV, Segmentation fault.
go () at stack.c:9
9    go( );
1: n = 786223
ulimit -s 0
gdb -q -x /tmp/gdb.cmd ../src/stack.exe\
    | grep SIG -A3
Program received signal SIGSEGV, Segmentation fault.
go () at stack.c:9
```

sous-processus

Un processus peut lancer des sous-processus de plusieurs manières :

- `fork` : le sous-processus hérite d'une copie contexte.
- `clone` : partage de variables.
- `execve` : exécute un binaire ou un script.
- `pthread` : processus léger

Dans les cas lourds/léger, un processus parent attend (normalité) la fin des sous-processus, et les sous-processus signalent leur terminaison.

job

Les sous-processus du shell (**jobs**) peuvent être en :

- avant plan
- arrière plan

La commande interne `jobs` donne la liste des sous-processus du shell.

```
1  #!/bin/bash -v
2  sleep 10 &
3  sleep  5 &
4  jobs
5  [1]-  Running                 sleep 10 &
6  [2]+  Running                 sleep 5 &
7  ps -p $!
8    PID TTY          TIME CMD
9   2812 pts/1    00:00:00 sleep
10 jobs -l %1
11 [1]-  2811 Running                 sleep 10 &
```

tâche en arrière plan

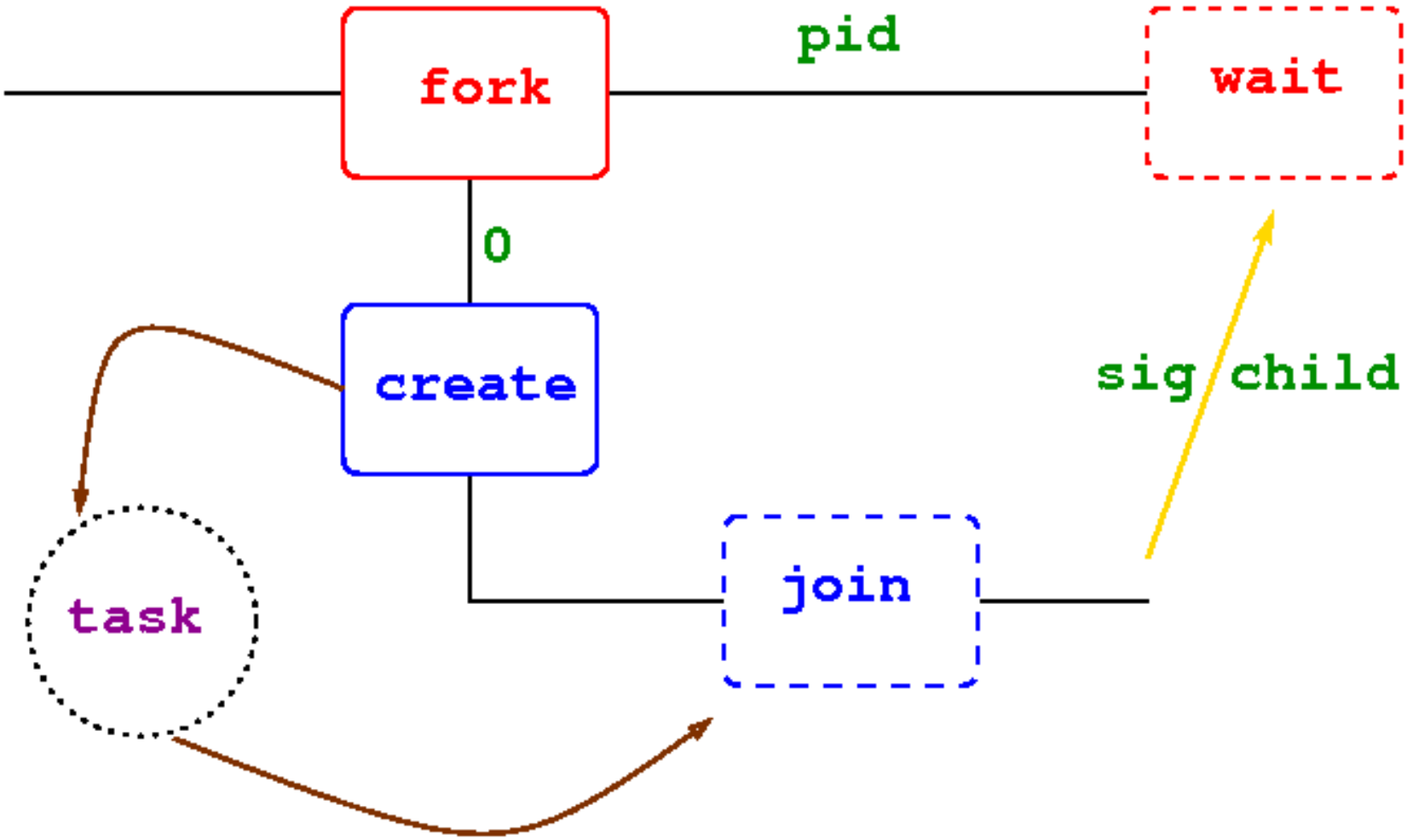
```
⇝ sleep 120 & sleep 60 & sleep 30 &
[1] 3649
[2] 3650
[3] 3651
⇝ kill -s SIGSTOP %2
⇝ jobs
[1]   En cours d'execution   sleep 120 &
[2]+  Stoppe                 sleep 60
[3]-  En cours d'execution   sleep 30  &
⇝ kill -s SIGKILL  %1
[1]   Processus arrete       sleep 120
⇝ jobs
[2]+  Stoppe                 sleep 60
[3]-  En cours d'execution   sleep 30  &
[3]-  Fini                   sleep 30
```

thread POSIX

```
1  #include <unistd.h>
2  #include <sys/types.h>
3  #include <sys/wait.h>
4  #include <stdio.h>
5  #include <errno.h>
6  #include <stdlib.h>
7  #include <pthread.h>
8
9  void * task (void *argv)
10 {
11 int x = *((int *) argv);
12 sleep (x);
13 pthread_exit (0);
14 }
```

bibliotèque pthread

```
1  int main (int argc, char *argv[] )
2  {
3  pid_t pid;
4  pthread_t th;
5  int status, x = atoi (argv[1]) ;
6  if ((pid = fork ()) < 0)
7      perror ("fork"), exit (1);
8
9  if ( pid )
10     wait (&status);
11 else {
12   if (pthread_create (&th, NULL, task, &x) < 0)
13     perror ("thread"), exit (2);
14     pthread_join (th, NULL);
15 }
```


fork
pid
wait
0
create
sig child
task
join

fork, pthread_create

```
1 #!/bin/bash -v
2 ./fils.exe 30 &
3 sleep 5
4 ps -T -o pid,ppid,spid,cmd
5 kill $!
6 ps -T -o pid,ppid,spid,cmd
```

fils.sh

processus lourds et légers de `make -B fils.out` ?

```
1 fils.exe : fils.c
2         gcc -Wall fils.c -o fils.exe -lpthread
3
4 fils.out : fils.exe fils.sh
5         ./fils.sh >& fils.out
```

```
./fils.exe 30 &
sleep 5
ps -T -o pid,ppid,spid,user,cmd
 1462  1415  1462 pl        /bin/bash
 2119  1462  2119 pl        make -B fils.out
 2125  2119  2125 pl        /bin/sh -c ./fils.sh
   >&  fils.out
 2126  2125  2126 pl        /bin/bash -v ./fils.
   sh
 2127  2126  2127 pl        ./fils.exe 30
 2129  2127  2129 pl        ./fils.exe 30
 2129  2127  2130 pl        ./fils.exe 30
 2131  2126  2131 pl        ps -T -o pid,ppid,
   spid,user,cmd
kill $!
ps -T -o pid,ppid,spid,user,cmd
 1462  1415  1462 pl        /bin/bash
```

9 - redirection

- flux vs descripteur
- redirection
- tracer un processus
- descripteur de fichier
- diagnostic
- substitution de processus

fichier C

La bibliothéque 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 gérer 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 gérer 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 êre 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 êre 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 êre 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 êre redirigés vers des descripteurs ou des fichiers

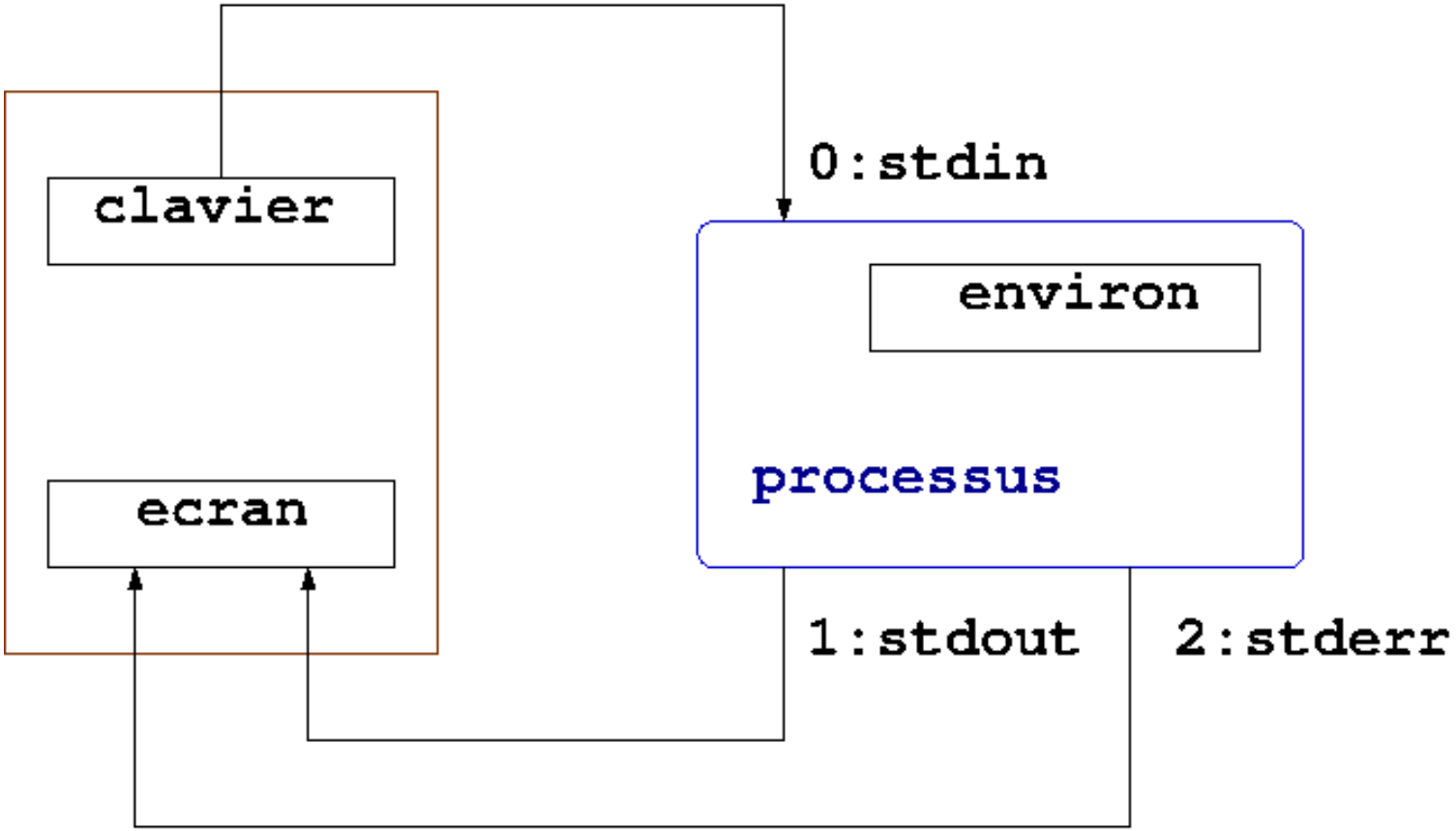
```
⇝ cat >fichier
Hello World          <- clavier
```

```
⇝ cat fichier
Hello World
```

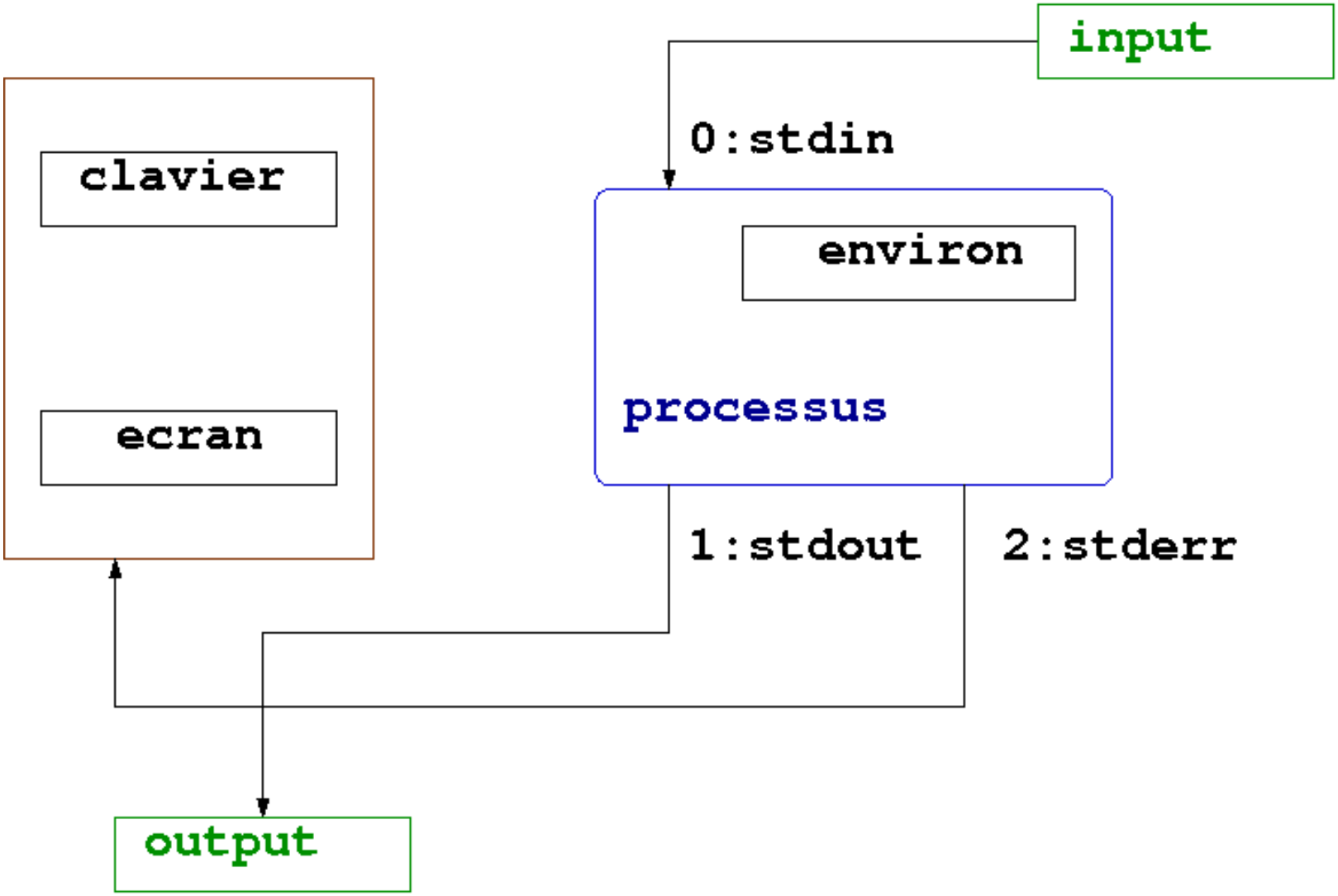
```
⇝ cat <fichier
Hello World
```

```
⇝ cat <fichier >output
```

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'aprés 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
```

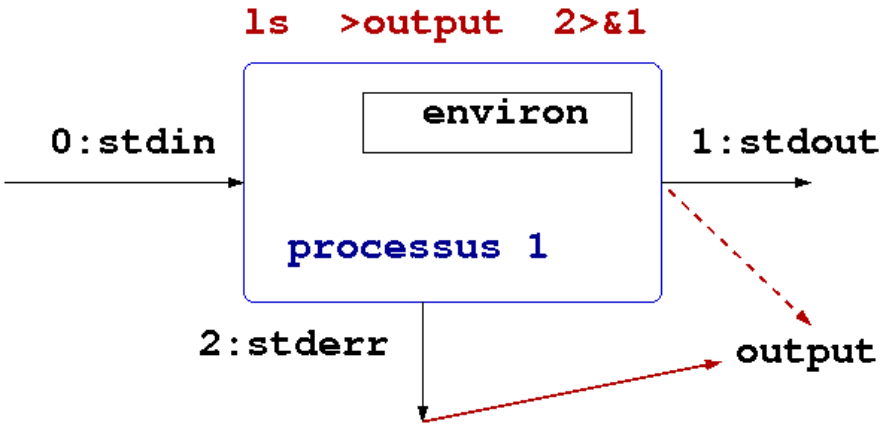
En effet

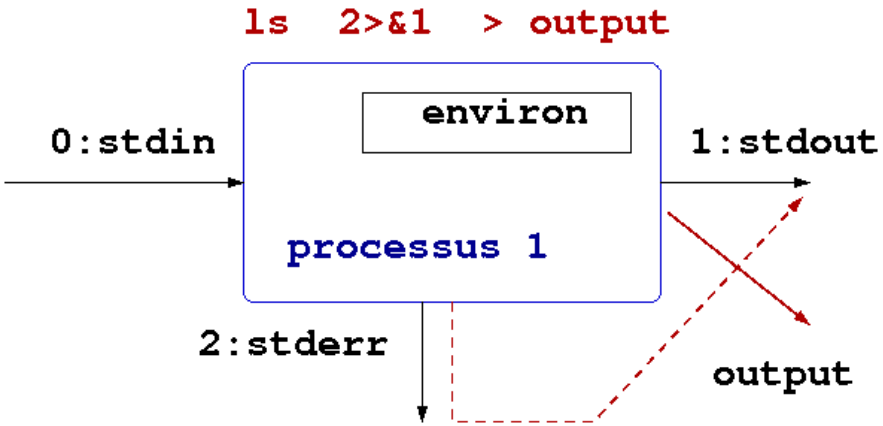
```
1 :: cat direct.sh erreur 2>&1 > /tmp/foo
2 cat: erreur : Aucun fichier ou dossier de ce type
3
4 :: cat direct.sh erreur > /tmp/bar 2>&1
```

```
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 prefere generalement la premiere, equivalente
    a:
                        >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 etre identifies , les infos
       sur les processus
5  non possedes ne seront pas affichees , vous devez etre 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.

```
1 hello.out : hello.exe
2         strace -o hello.sys \
3          -e read,write,open,execve,close,stat ./hello.exe
4         ltrace -o hello.lib ./hello.exe
5         ltrace -c -o hello.lc ./hello.exe
6         strace -c -o hello.tmp ./hello.exe
7         sed 's/[ ]\+/ /g' hello.tmp > hello.sc
8         awk '/hello.out/,/@@@/' makefile > hello.out
```

makefile

[ptrace]

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[style=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   1   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 systeme `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 systeme `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	__mempcpy_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	mempcpy
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
    }
```

memo 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\0\3\0\3\0\1\0\0\0\220\347\31
    B4\0\0\0"..., 512) = 512
close(3)                                        = 0
```

memo 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
```

memo 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\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
```

10 - pipeline

- tracer des commandes
- pipeline
- tracer un pipeline

traitement des commandes

Le prompt du shell matérialise une ligne de commande invitant l'utilisateur à soumettre une instruction : une chaine de caractères analysée par le shell pour lancer des commandes.

- interne : traitée dans le processus.
- externe : `clone` + `execve`.
- complexe : `clone`

trace des commandes externes

```
1 #!/bin/bash
2 /bin/echo hello
3 echo HELLO
4 /bin/echo allo
```

echos.sh

```
echos.out:echos.sh
 strace -f -e trace=execve,clone,waitpid,write\
            -o echos.str -e signal=SIGCHLD ./
   echos.sh

 sed -n '/echos.out/,/HERE/p' makefile > echos.
   out
```

echos.make

echos.strace

```
19330 execve("./echos.sh", ["./echos.sh"], [/* 54
    vars */]) = 0
19330 clone(child_stack=0, flags=
    CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
     child_tidptr=0x7fef183379d0) = 19331
19331 execve("/bin/echo", ["/bin/echo", "hello"],
    [/* 53 vars */]) = 0
19331 write(1, "hello\n", 6)                    = 6
19330 --- SIGCHLD (Child exited) @ 0 (0) ---
19330 write(1, "HELLO\n", 6)                    = 6
19330 clone(child_stack=0, flags=
    CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
     child_tidptr=0x7fef183379d0) = 19332
19332 execve("/bin/echo", ["/bin/echo", "allo"],
    [/* 53 vars */]) = 0
19332 write(1, "allo\n", 5)                     = 5
```

pipeline

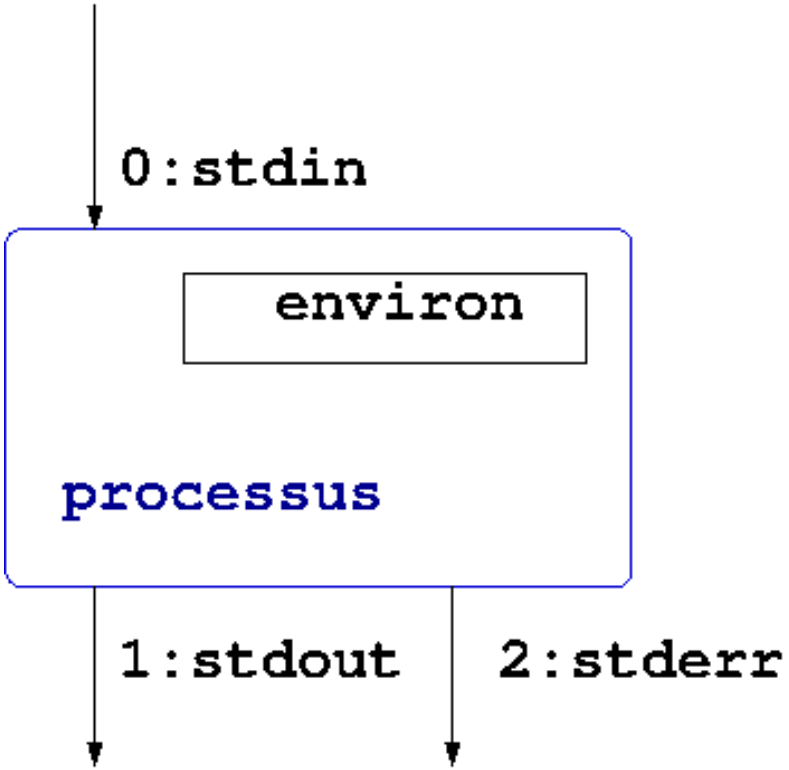
Le pipeline permet de chaîner des processus de sorte que la sortie d'un processus (stdout) alimente directement l'entrée (stdin) du suivant.

```
1 ::   ps -eoppid | sort | uniq -c | sort -gr
2          | head -3 | sed 's/[ ]\+/:/g'
3            | cut -d':' -f3 | xargs ps -p
```

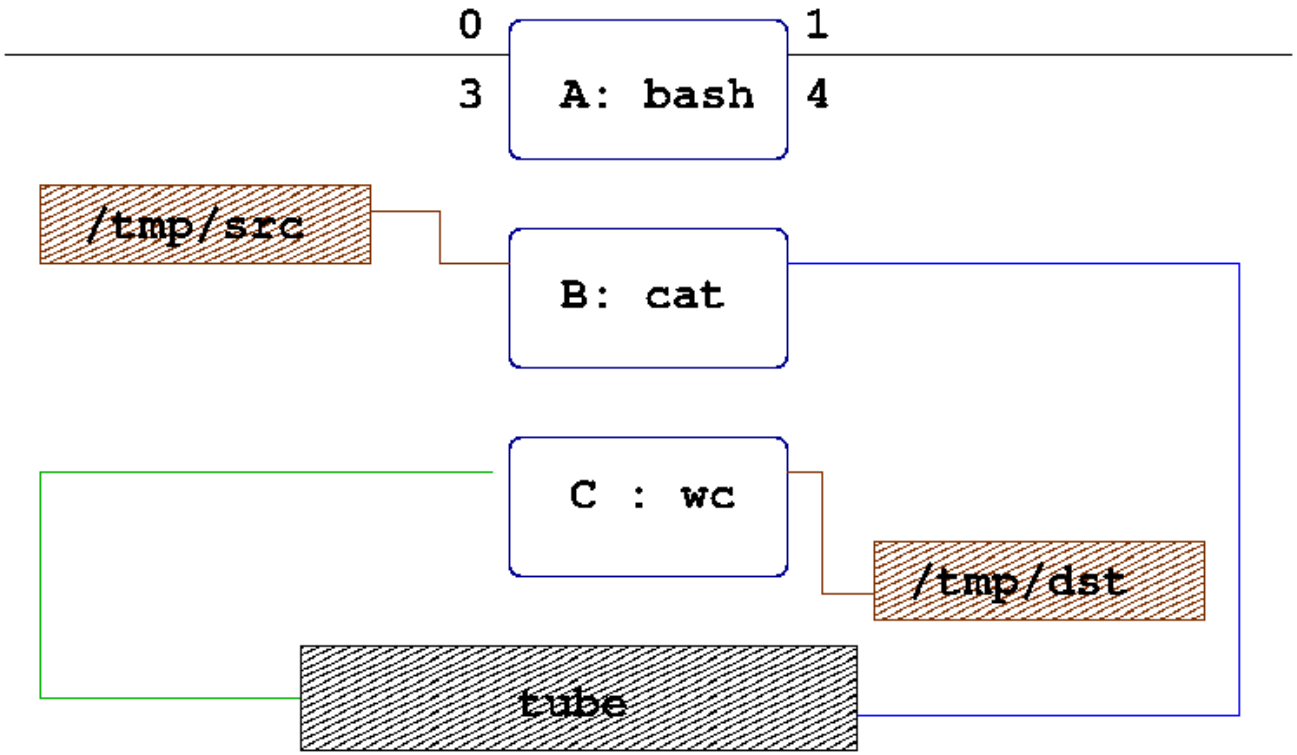
La mise en oeuvre d'un pipeline s'appuie sur les appels système :

- `fork` / `clone`
- `pipe`, `dup2`
- `execve`

processus



cat </tmp/src | wc -l > /tmp/dst



bash pipeline

```
1 echo hello > /tmp/src
2 call =open,close,pipe,dup2,execve,clone,fork
3 strace -o pipe.out -f -e trace=$call -p $$ &
4 sleep 1
5 cat </tmp/src | wc -l >/tmp/dst
```

pipe.sh

```
./pipe.sh > pipe.out
Process 2291 attached
Process 2298 attached
Process 2299 attached
./pipefilter.sh pipe.out
```

trace brute

```
5377  --- SIGCHLD {si_signo=SIGCHLD, si_code=
   CLD_EXITED, si_pid=5379, si_uid=1000,
   si_status=0, si_utime=0, si_stime=0} ---
5377  pipe([3, 4])                          = 0
5377  clone(child_stack=0, flags=
   CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|
   SIGCHLD, child_tidptr=0xb778a728) = 5384
5384  close(255)                            = 0
5377  close(4)                              = 0
5377  close(4)                              = -1
   EBADF (Bad file descriptor)
5377  clone( <unfinished ...>
5384  close(3)                              = 0
5384  dup2(4, 1 <unfinished ...>
5377  <... clone resumed> child_stack=0, flags=
```

filtre

```
1  #!/bin/bash
2
3  grep -Ev '(locale | cache| BAD|\-1|255|exited|libc)' pipe.
       out \
4      | sed -r 's/(clone).*=(.*)/\1 := \2/' \
5      | uniq | tr -d '[A-Z]' | cut -c1-44 > mypipe.out
6
7  set $(grep -oE '^[0-9]+' mypipe.out | sort | uniq )
8  sed -i -e "s/$1/& (A)/g" \
9         -e "s/$2/& (B)/g" \
10        -e "s/$3/& (C)/g" mypipe.out
```

trace filtrée (1/3)

```
5377 (A)  ---  {si_signo=, si_code=_, si_pid=537
5377 (A)  pipe(3, 4)                          = 0
5377 (A)  clone := 5384 (B)
5377 (A)  close(4)                            = 0
5377 (A)  clone( <unfinished ...>
5384 (B)  close(3)                            = 0
5384 (B)  dup2(4, 1 <unfinished ...>
5377 (A)  <... clone := 5385 (C)
5384 (B)  <... dup2 resumed> )                = 1
5384 (B)  close(4)                            = 0
5377 (A)  close(3)                            = 0
5385 (C)  dup2(3, 0)                          = 0
```

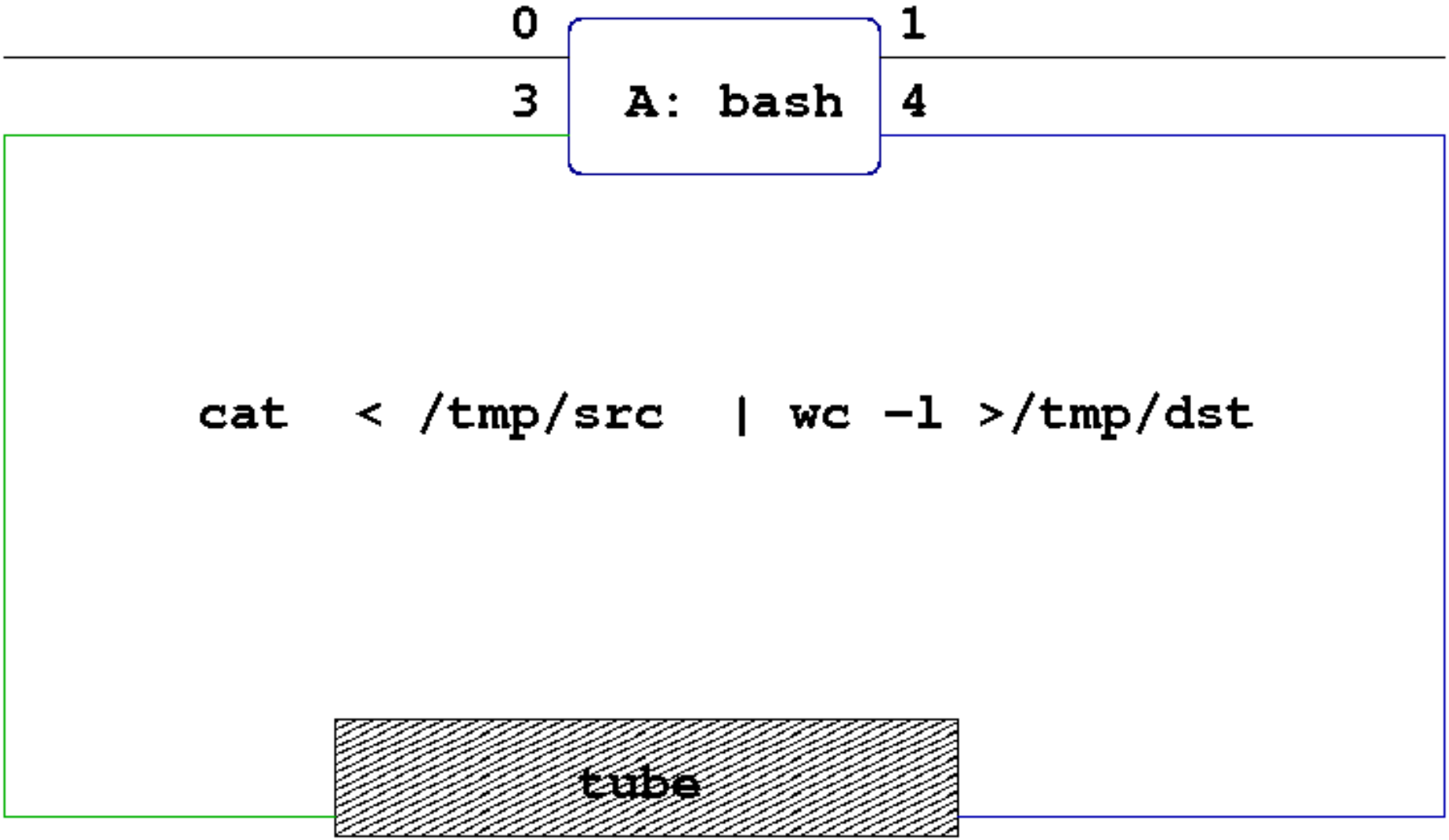
trace filtrée (2/3)

```
5385 (C)  close(3)                                  = 0
5384 (B)  open("/tmp/src", _|_) = 3
5384 (B)  dup2(3, 0)                                = 0
5384 (B)  close(3)                                  = 0
5384 (B)  execve("/usr/bin/cat", "cat", /* 55 va
5385 (C)  open("/tmp/dst", _|_|_|_, 0666) = 3
5385 (C)  dup2(3, 1)                                = 1
5385 (C)  close(3)                                  = 0
5384 (B)  <... execve resumed> )                    = 0
5385 (C)  execve("/usr/bin/wc", "wc", "-l", /* 5
5385 (C)  <... execve resumed> )                    = 0
5384 (B)  <... open resumed> )                      = 3
5384 (B)  close(3)                                  = 0
```

trace filtrée (2/3)

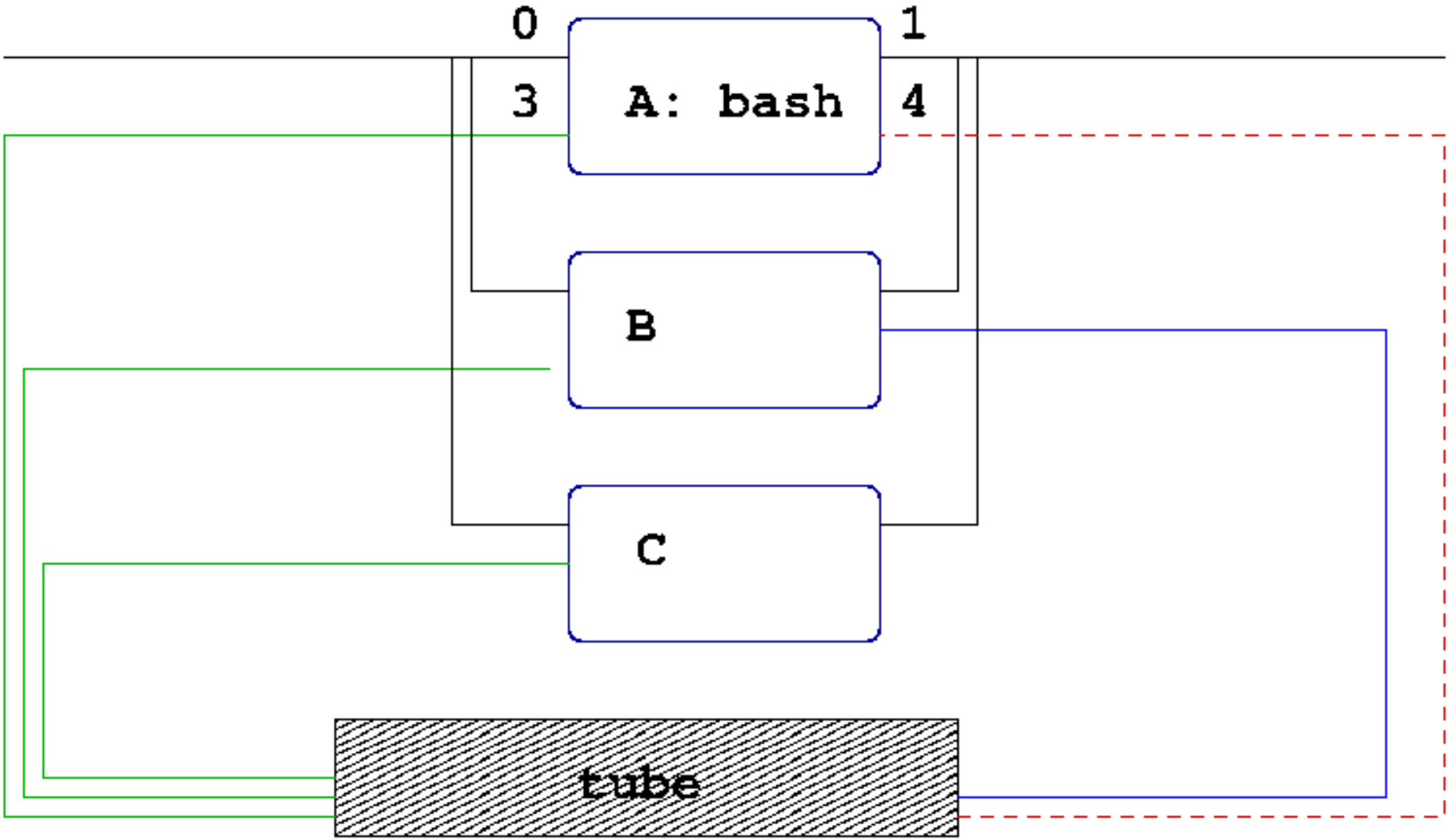
```
5385 (C)   <... open resumed> )                    = 3
5384 (B)   <... open resumed> )                    = 3
5385 (C)   close(3)                                = 0
5384 (B)   close(3)                                = 0
5385 (C)   close(3)                                = 0
5384 (B)   close(3)                                = 0
5385 (C)   close(3)                                = 0
5384 (B)   close(0)                                = 0
5384 (B)   close(1 <unfinished ...>
5385 (C)   close(3 <unfinished ...>
5384 (B)   <... close resumed> )                   = 0
5385 (C)   <... close resumed> )                   = 0
5384 (B)   close(2)                                = 0
5385 (C)   close(3)                                = 0
5385 (C)   close(0)                                = 0
5385 (C)   close(1)                                = 0
```

A : pipe(3,4)



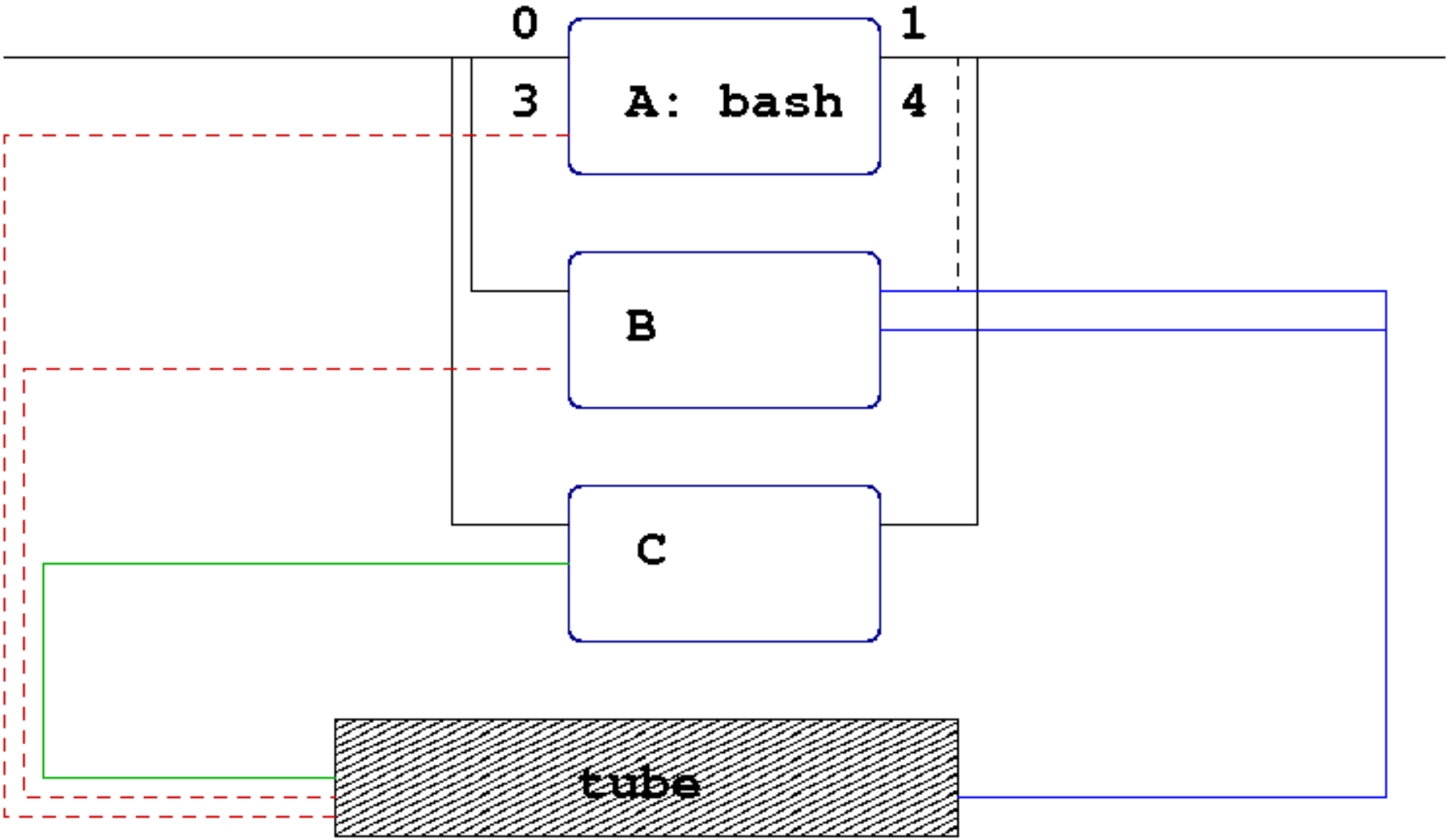
A : clone,close(4), clone

A : clone, close(4), clone



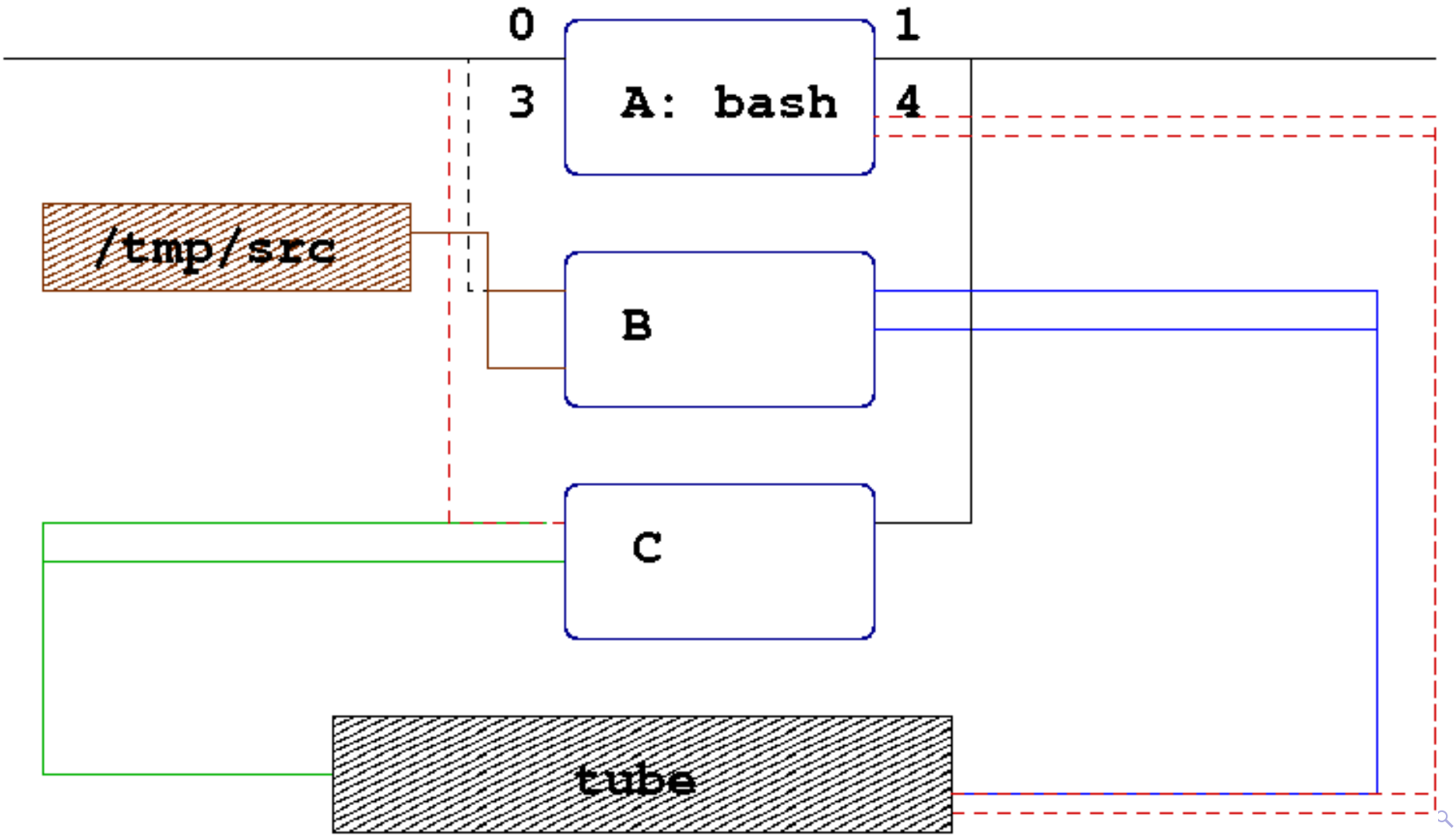
B : close(3), dup2(4,1), A : close(3)

B : close(3), dup2(4,1), A : close(3)

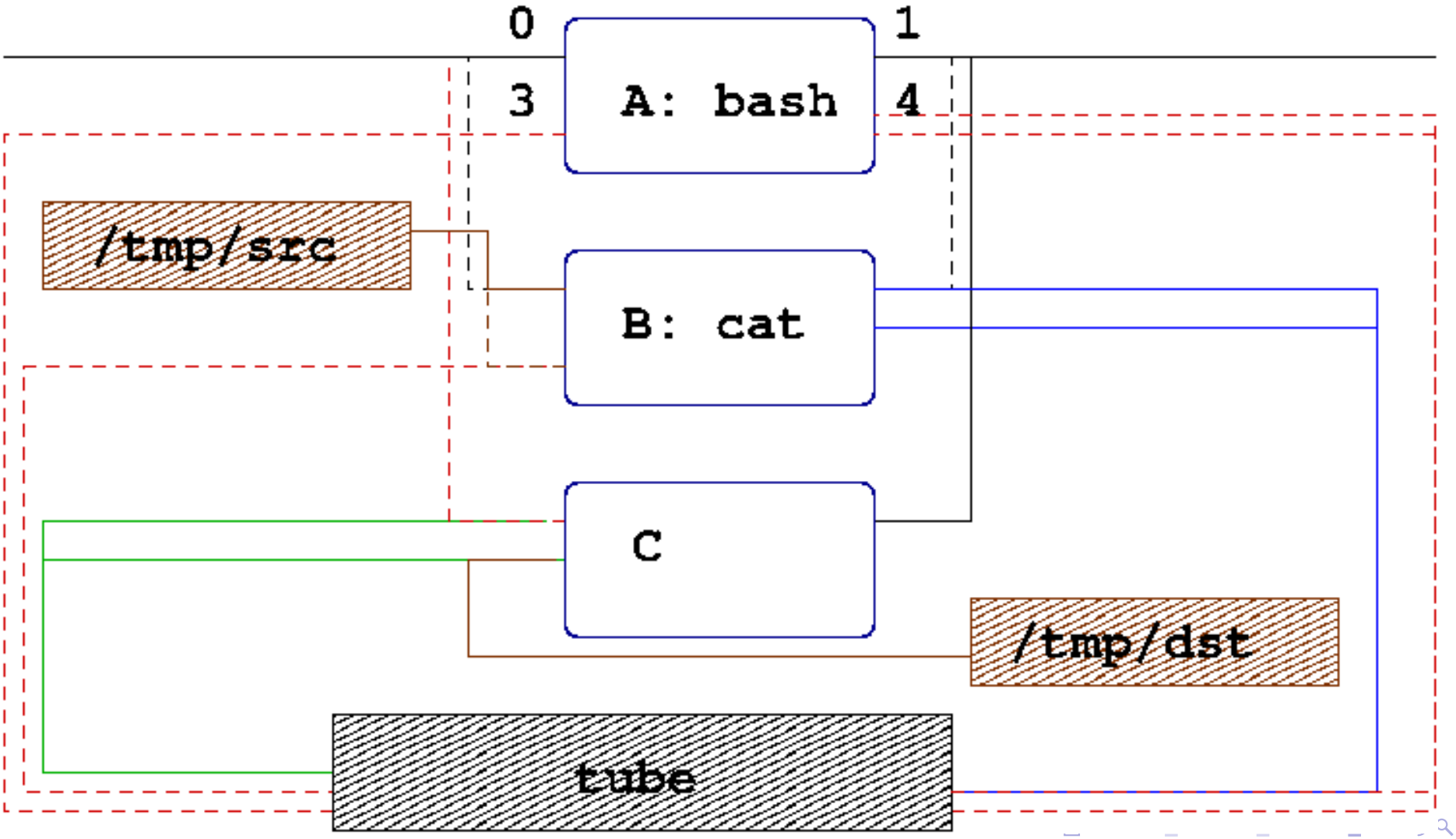


C : dup2(3,0), B : open(/tmp/src) = 3, B : dup2(3,0)

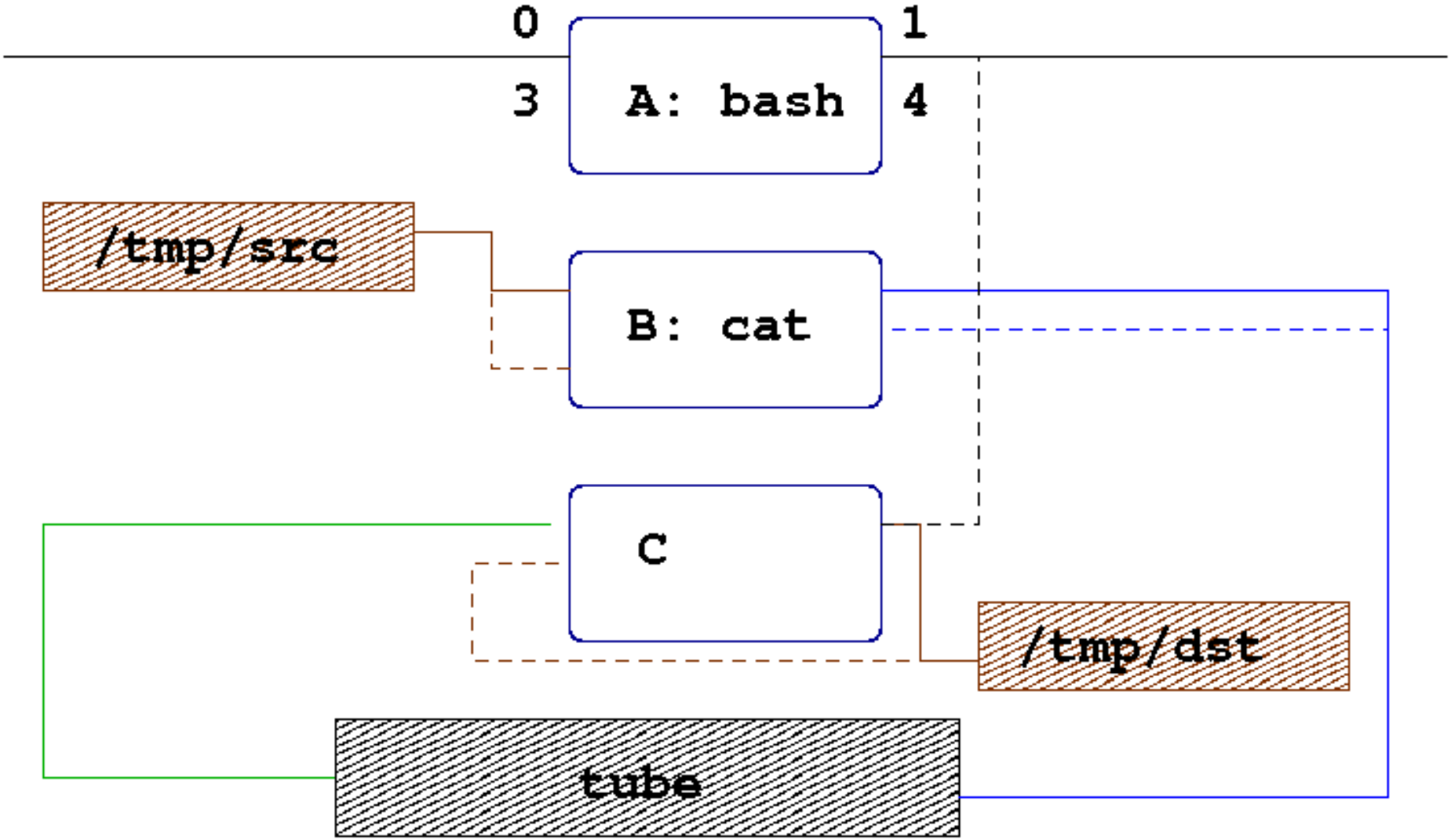
C : dup2(3,0), B : open(/tmp/src)=3, B : dup2(3,0)



B : close(3), C : open(/tmp/dst)=3, B : execve(cat)



C : dup2(3,1), close(3), execve(wc)



11 - outils

- regex
- grep
- find
- sed
- awk
- vi

outils

La liste des commandes utiles s'allonge d'années en années. Mais la croissance n'est pas exponentielle, et il est encore possible de se mettre à jour ! Dans vos TPs, vous travaillerez les incontournables :

- `make`, `gcc`, `gdb`, `gprof` etc...
- `gnuplot`, `dot`, `tex`, `latex`...
- `flex`, `bison`

utiliserez régulièrement les

- `ps`, `sort`, `wc`
- `cut`, `tr`
- `ssh`, `scp`, `sftp`

quelques démos

Il s'agit de faire quelques démonstrations concernant la bibliothèque `regex` et les commandes `grep` , `sed` , `find` , `awk` et `vi`

quelques démos

Il s'agit de faire quelques démonstrations concernant la bibliothèque `regex` et les commandes `grep` , `sed` , `find` , `awk` et `vi`

A très peu de choses prés, tout est prévu dans ces commandes pour les manipulation des fichiers textes. Il s'agit plus de savoir comment retrouver rapidement une option utile avec son mode d'emploi !

quelques démos

Il s'agit de faire quelques démonstrations concernant la bibliothèque `regex` et les commandes `grep` , `sed` , `find` , `awk` et `vi`

A très peu de choses prés, tout est prévu dans ces commandes pour les manipulation des fichiers textes. Il s'agit plus de savoir comment retrouver rapidement une option utile avec son mode d'emploi !

La plupart du temps, les expressions régulières jouent un rôle clef dans le traitement des fichiers.

rexexp

```
REGEX(7)          Linux Programmer's Manual
NAME
        regex – POSIX.2 regular expressions
DESCRIPTION
 Regular expressions ("RE"s), as defined in POSIX2,
come in 2 forms:modern REs (roughly those of egrep;
POSIX.2 calls these "extended" REs) and obsolete
REs (roughly those of ed(1); POSIX.2 "basic" REs).

Obsolete REs mostly exist for backward compatible.
in some old  programs; they  will  be discussed at
the end. POSIX.2 leaves some aspects of RE syntax
and semantics open; "(!)" marks decisions on these
aspects that may not be fully portable to other
POSIX.2 implementations.
```

rexexp

```
1   regcomp (&r, argv[1], REG_EXTENDED) != 0);
2   src = fopen (argv[2], "r");
3   while ( ! feof (src) ) {
4       fgets (line , 1024, src );
5       s = 0;
6       while (0 == regexec (&r, &(line[s]), 1, &pos, 0))
7     {
8       // traitement
9     }
10    }
11  regfree (&r);
```

grep

origine

mars 1973, par Ken Thompson.

La version GNU de `grep` utilise l'algorithme de recherche de motif de Boyer-Moore (1977).

- oxford dictionary
- wiki grep
- gnu grep
- voir aussi : `ngrep`

grep

```
NAME
    grep - print lines matching a pattern

SYNOPSIS
    grep [OPTIONS] PATTERN [FILE...]
    grep [OPTIONS] [-e PATTERN | -f FILE] [FILE...]

DESCRIPTION
    grep  searches the named input FILEs for lines
containing  a  match to the given PATTERN.  By
default, grep prints the matching lines.
```

grep -e vs -E

```
⇝ grep -rio  --include="*.tex"
             -e '\(emacs\|vim\)' ~/doc

⇝ grep -rio  --include="*.tex"
             -E '(emacs|vim)' ~/doc
```

grep -e vs -E

```
⇝ grep -rio  --include="*.tex"
             -e '\(emacs\|vim\)' ~/doc

⇝ grep -rio  --include="*.tex"
             -E '(emacs|vim)' ~/doc
```

- recherche les fichiers TEX, dans l'arborescence **doc**, qui mentionnent un des éditeurs `vi` ou `emacs`

grep -e vs -E

```
⇝ grep -rio  --include="*.tex"
             -e '\(emacs\|vim\)' ~/doc

⇝ grep -rio  --include="*.tex"
             -E '(emacs|vim)' ~/doc
```

- recherche les fichiers TEX, dans l'arborescence **doc**, qui mentionnent un des éditeurs `vi` ou `emacs`

```
/home/pl/UPS/doc/tools.tex:emacs
/home/pl/UPS/doc/tools.tex:vim
/home/pl/UPS/doc/prologue.tex:emacs
/home/pl/UPS/doc/prologue.tex:emacs
```

grep -iownE '[-a-z]+regexp' doc/*.tex

```
Matcher Selection
        -E, --extended-regexp
Matching Control
        -e PATTERN, --regexp=PATTERN
        -i , --ignore-case
        -v , --invert-match
        -w, --word-regexp
        -x , --line-regexp
General Output Control
        -c , --count print  a count of matching
        -L , --files-without-match
        -l , --files-with-matches
        -m NUM, --max-count=NUM
        -o , --only-matching
        -q , --quiet , --silent
```

grep -Rli –include=*rc path /etc 2>/dev/null

```
Output Line Prefix Control
        -b, --byte-offset
        -H, --with-filename
        -h, --no-filename
        -n, --line-number
Context Line Control
        -A NUM, --after-context=NUM
        -B NUM, --before-context=NUM
        -C NUM, -NUM, --context=NUM
File and Directory Selection
        --exclude=GLOB
        --exclude-from=FILE
        --exclude-dir=DIR
        --include=GLOB
        -R, -r, --recursive
```

exemple

```
⇝ grep -iownE '[-a-z]+regexp' doc/*.tex
doc/tools.tex:126:--extended-regexp
doc/tools.tex:128:--regexp
doc/tools.tex:131:--word-regexp
doc/tools.tex:132:--line-regexp
doc/tools.tex:319:--regexp
```

```
⇝ grep -iwlER '[-a-z]+regexp' --include=*.tex ~
/home/pl/web-docs/cours/UPS/doc/tools.tex
```

retour de grep

0 au moins une occurrence
1 aucune occurrence
2 erreur d'accès, erreur de syntaxe

```
for file in *
do
    if grep -qE '[0-9]+' $file
        then
            echo liste de $file :
            grep -oE '[0-9]+' | sort | uniq -c
        else
            echo $file pas de nombre
    fi
done
```

find

origine

Dick Haight (?)

`find` est une commande puissante pour trouver les fichiers en fonction de critères variés.

- sur -print
- wiki find
- gnu findutils
- voir aussi : `locate`, `xargs`

find

La même chose que précédemment, avec `find`, sans le support récursif de `grep`

```
⇝ find ~/doc/tools.tex -name "*.tex"
    -exec grep -Hio -E '(emacs|vim)' {}\;
```

ou encore

```
⇝ find ~/doc/tools.tex -name "*.tex" |
            xargs grep -Hio -E '(emacs|vim)'
```

man find | wc -l ... 1615 !

```
NAME
   find - search for files in a directory hierarchy
SYNOPSIS
   find [-H][-L][-P][-Dx][-Ox][path...] [expression]

DESCRIPTION
   GNU find searches the directory tree rooted at
each given file name by evaluating the given
expression from left to right, according to the
rules of precedence (see section OPERATORS),
until the outcome is known...
```

option

-maxdepth -mindepth -mount -regextype

```
$ find / -maxdepth 2 -name "$$"          2>/dev/
    null
/proc/2105
$ find / -mount -maxdepth 2 -name "$$" 2>/dev/
    null
```

```
$ find / -maxdepth 2 -regex ".*/[0-2]{1}" 2>/dev
    /null
$ find / -regextype posix-extended -maxdepth 2 \
         -regex ".*/[0-2]{1}" 2>/dev/null
/proc/1
/proc/2
```

```
$ find ~ -mindepth 6 -name '\.*'   -prune \
```

test

`-name, -iname`	nom de base	
`-wholename`	nom complet	
`-regex, -iregex`	nom de base	
`-type`	type	f, d, p, c, l, b
`-user`	propriété	
`-group`	groupe	
`-perm`	permission	
`-links`	nombre de liens	
`-size`	taille	char, word, bloc, k, M, G
`-atime,-amin`	date d'accès	
`-mtime,-mmin`	modification	
`-ctime,-cmin`	création	
`-newer`	comparaison des dates	

inoeud faible

```
$ find / -inum -3 -printf "%i %u %p\n"
2 root /
1 pl /home/pl/.gvfs
1 root /sys
2 root /sys/fs
1 root /proc
1 root /proc/sys/fs/binfmt_misc
2 root /proc/sys/fs/binfmt_misc/status
2 root /boot
1 root /dev/pts
2 root /dev/pts/ptmx
```

A propos de `find` et de -**print**

. . . You know, like why is the -**i** option for `grep` mean ignore case, and the -**f** option for `sort` mean ignore case, and so on. . . Well, the instructor of the course decided to chime in and said something like this :

"Here's another good example of this problem with UNIX. Take the `find` command for example. WHAT idiot would program a command so that you have to say -**print** *to print the output to the screen. What IDIOT would make a command like this and not have the output go to the screen by default"*

And the instructor went on and on, and vented his spleen...

A propos de `find` et de **-print**

The next morning, one of the ladies in the class raised here hand, the instructor called on her, and she proceeded to say something like this :

"*The reason my father programmed the find command that way, was because he was told to do so in his specifications.*"

A propos de `find` et de **-print**

The next morning, one of the ladies in the class raised here hand, the instructor called on her, and she proceeded to say something like this :

"*The reason my father programmed the find command that way, was because he was told to do so in his specifications.*"

- D'après ce courrier, Dick Haight est à l'origine de `find`.

action

```
⇝ find ~ -size +1M -ctime -10 -perm 0664
                -name "*.JPG" -user pl -ok delete
    \;
< delete ... /home/pl/pics/IMGP0968.JPG > ? n
< delete ... /home/pl/pics/IMGP0514.JPG > ? n
< delete ... /home/pl/pics/IMGP0965.JPG > ? n
```

```
-delete
-exec command ;
-execdir command ;
-ls
-ok command ;
-okdir command;
-print
-print0
-printf
```

Editeur de flux : sed

```
NAME
   sed - stream editor for filter-transforming text
SYNOPSIS
   sed [OPTION]... {script} [input-file]...
DESCRIPTION
 sed is a stream editor. A stream editor is
used to perform basic text transformations on an
input stream
OPTION
   -n, --quiet, --silent
   -e script, --expression=script
   -r, --regexp-extended
   -f script-file, --file=script-file
   -i[SUFFIX], --in-place[=SUFFIX]
   -l N, --line-length=N
```

sed

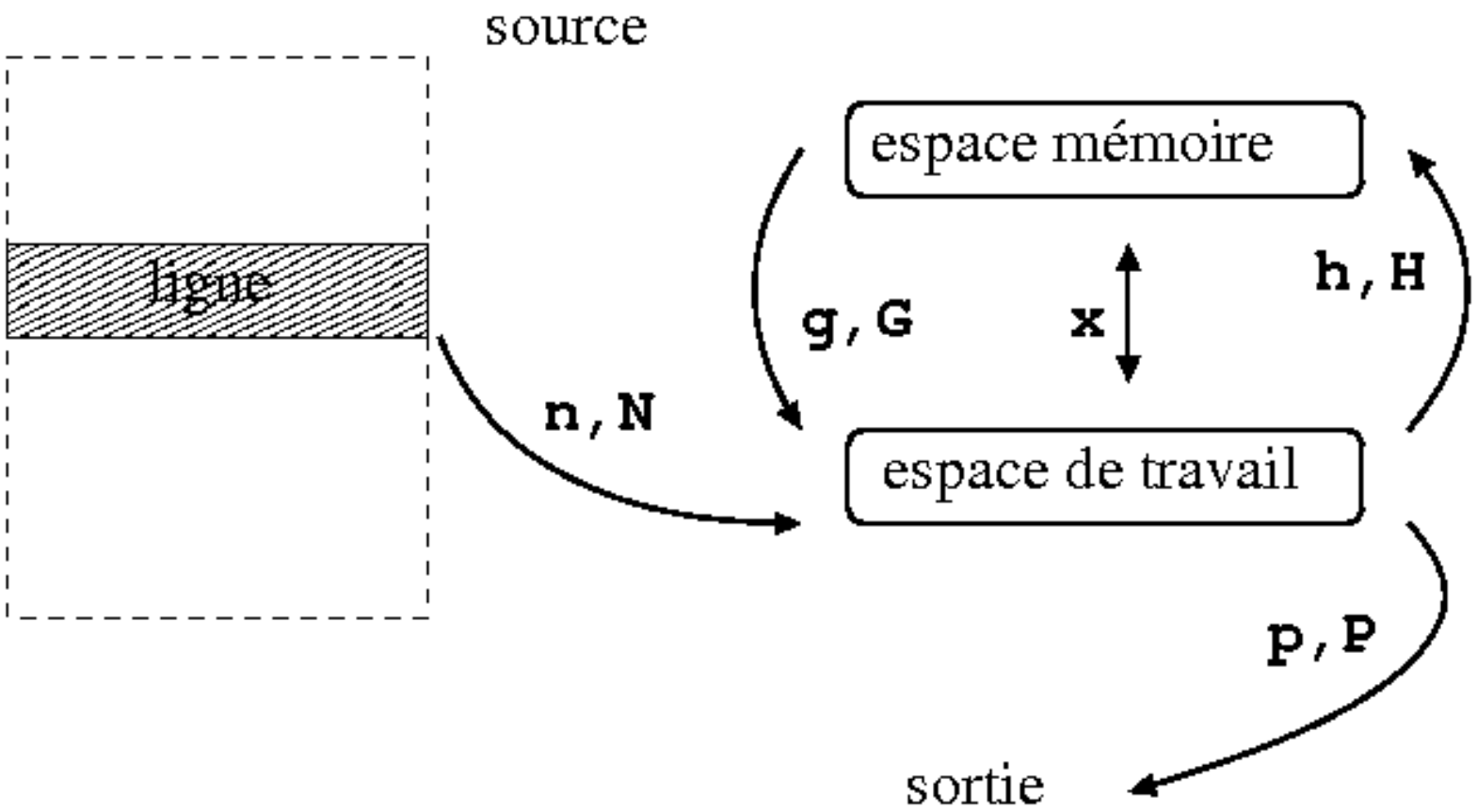
origine

1973-74, par Lee McMahon, Bell labs.

- wiki sed
- gnu sed

principe

sed charge les lignes d'un fichier correspondant à une adresse dans un espace de travail pour lui appliquer des commandes d'édition.



adresse : motif, ligne

```
1 #!/bin/bash -v
2 sed '/nologin/d'  /etc/passwd
3 root:x:0:0: root:/root:/bin/bash
4 sync:x:5:0: sync:/sbin:/bin/sync
5 shutdown:x:6:0: shutdown:/sbin:/sbin/shutdown
6 halt:x:7:0: halt:/sbin:/sbin/halt
7 pl:x:501:501::/home/pl:/bin/bash
8 guest:x:502:502::/home/guest:/bin/bash
```

adresse : intervalle

```
1  #!/bin/bash -v
2  sed -rn '/dae.*n/,7{=;p}'  /etc/passwd
3  3
4  daemon:x:2:2:daemon:/sbin:/sbin/nologin
5  4
6  adm:x:3:4:adm:/var/adm:/sbin/nologin
7  5
8  lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
9  6
10 sync:x:5:0:sync:/sbin:/bin/sync
11 7
12 shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
13 24
14 avahi:x:70:70:Avahi mDNS/DNS-SD Stack:/var/run/avahi-
       daemon:/sbin/nologin
```

commande

q	quitter
d,D	effacer l'espace des motifs
s	substitution
y	remplacement
p, P	affiche l'espace des motifs
n, N	chargement d'une ligne
h, H	mémoriser
g,G	charger la mémoire
x	échange les contenus mémoires
{}	groupement des commandes
=	numéro de ligne
:	étiquette
b	saut

```
#!/bin/bash -v
echo {1..6} | sed 'y/ /\n/' > /tmp/foo
sed 'N;s/\n/-/' /tmp/foo
1-2
3-4
5-6
sed -n 'h;n;G;s/\n/-/p' /tmp/foo
2-1
4-3
6-5
sed -r '/[3]/{ h;n;G;s/(^|\n)/-/g}' /tmp/foo
1
2
3
-4-3
5
6
```

exemples

```
1  #!/bin/bash -v
2
3  sed -rne '/root/s/[ ]*(:|^) [ ]*.\ U&/gp' /etc/passwd
4  Root:X:0:0:Root:/root:/bin/bash
5  Operator:X:11:0:Operator:/root:/sbin/nologin
6  echo abcdefgh | sed 's/..\ u&-/g'
7  Ab-Cd-Ef-Gh-
8
9  echo abcde | sed 's/./&\n/g;' > /tmp/abcd
10 sed = /tmp/abcd | sed -n '1~2N;s\s/-/p'
11 1-a
12 2-b
13 3-c
14 4-d
15 5-e
```

exemples

```
1  #!/bin/bash -v
2  echo -e '1\n2\n3\n4\n5\n6\n7' > /tmp/nbr
3  sed -ne 'N;N;s/\n/:/gp' /tmp/nbr
4  1:2:3
5  4:5:6
6  sed -e 'x;n;' /tmp/nbr | tr '\n' '/'; echo
7  /2/1/4/3/6/5/
8  sed -e '3,5d' /tmp/nbr | tr '\n' '/'; echo
9  1/2/6/7/
10 sed -ne '3,5p' /tmp/nbr | tr '\n' '/'; echo
11 3/4/5/
12 sed -ne '3,5!p' /tmp/nbr | tr '\n' '/'; echo
13 1/2/6/7/
```

s/regexp/replacement/flags

```
1  #!/bin/bash -v
2
3  sed -ne '/sh$/s/:.*:/ --> /p' /etc/passwd
4  root --> /bin/bash
5  pat --> /bin/bash
6  pl --> /bin/bash
7  juju --> /bin/bash
8  sed -ne '/sh$/s/\([^:]*\):.*:\([^:]*\) /\2 <-- \1/p' /etc/
       passwd
9  /bin/bash <-- root
10 /bin/bash <-- pat
11 /bin/bash <-- pl
12 /bin/bash <-- juju
```

awk

Un mini langage pour manipuler les colonnes d'un fichier. `awk` a été créé aux Bell Labs dans les années 70s par :

- Alfred Aho
- Peter Weinberger
- Brian Kernighan.

```
⇝ cat /etc/passwd | awk -F: '{ print $1 }'
```

C'est un des langages qui a inspiré `perl`.

- wiki awk
- gawk
- voir aussi : `locate`, `xargs`

awk

```
⇝ awk '/^p.*l/ {print $1}' /etc/passwd
pulse:x:998:996:PulseAudio
pl:x:1000:1000:pl:/home/pl:/bin/bash
⇝ awk -F: '/^p.*l/ {print $2+$3+$4}' /etc/passwd
1994
2000
⇝ awk --posix -F ':' '$4 ~ /5[0-9]{2}/ { print
    $1, $4}' /etc/passwd
pl 500
pat 501
```

Perl combines (in the author's opinion, anyway) some of the best features of C, sed, awk, and sh, so people familiar with those languages should have little difficulty with it. Larry Wall

Un exemple tiré de linux magazine

```
#!/bin/bash
# Grab a list of users from the /etc/passwd file
cat /etc/passwd |grep sh |grep home | awk -F: '{
    print $1}'
```

pipeline overdose

En général, un abus de pipelines traduit des maladresses !

Un exemple tiré de linux magazine

```
#!/bin/bash
# Grab a list of users from the /etc/passwd file
cat /etc/passwd |grep sh |grep home | awk -F: '{
    print $1}'
```

pipeline overdose

En général, un abus de pipelines traduit des maladresses !

```
grep -E '(sh|home)' | awk -F: '{print $1}'
```

Un exemple tiré de linux magazine

```
#!/bin/bash
# Grab a list of users from the /etc/passwd file
cat /etc/passwd |grep sh |grep home | awk -F: '{
    print $1}'
```

pipeline overdose

En général, un abus de pipelines traduit des maladresses !

```
grep -E '(sh|home)' | awk -F: '{print $1}'
```

```
awk -F: '/home/ && /sh/ {print $1}' /etc/passwd
```

Un exemple tiré de linux magazine

```
#!/bin/bash
# Grab a list of users from the /etc/passwd file
cat /etc/passwd |grep sh |grep home | awk -F: '{
    print $1}'
```

pipeline overdose

En général, un abus de pipelines traduit des maladresses !

```
grep -E '(sh|home)' | awk -F: '{print $1}'
```

```
awk -F: '/home/ && /sh/ {print $1}' /etc/passwd
```

```
awk -F: '$6 ~ /home/ && $7 ~ /sh/ {print $1}' /
    etc/passwd
```

awkword

```
lynx --dump $1 | awk '
    BEGIN { FS="[^a-zA-Z]+" }

    {
        for (i=1; i<=NF; i++) {
            word = tolower($i)
            words[word]++
        }
    }

    END {
        for (w in words)
            printf ("%3d %s\n", words[w], w)
    }
' | grep -E '[a-z]{3}' | sort -rn | head -12
```

vi

L'éditeur visuel `vi` (William Joy, BSD) est un bon laboratoire pour pratiquer les expressions régulières utiles pour l'éditeur non visuel `sed`.

```
⇝ vi test.txt
```

Vous comprendrez que `vi` démarre en mode commande et qu'il fonctionne principalement en trois modes :

- insertion
- commande
- visuel

CTRL-c bascule en mode commande. CTRL-v bascule en mode visuel

vimbook

commande

commande	action	exemple
:help	aide	:help yank
:w	sauvegarde	:w file.tex
:q	quitte	
:q !	quitte sans sauver	
:wq	modifie et quitte	
:sh	lancer un shell	:sh ... `exit`
:make	lancer `make`	
:ab	abréviation	:ab anti anticonstitutionnellement
:split	nouveau cadre	:split common.c

bouger

commande	commentaire	portée
w	suivant	mot
b	précédent	mot
^	début	ligne
0	début	ligne
$	fin	ligne
:0	début fichier	
:5	ligne	fichier
:$	fin	fichier
CTRL-W	changer de cadre	

rechercher

commande	commentaire
:/*motif*	rechercher le motif
n	suivant
N	précédent

insérer

commande	action	commentaire
i	insérer	avant le curseur
a		aprés le curseur
A		fin de ligne
I		début de ligne
o		ligne suivante
O		ligne précédente
:r *file*	un fichier	:r foo.txt
:r !*commande*	une sortie	:r ! `wc` bar.txt

diverse

commande	action	commentaire
u	annuler	undo
U	annuler	undo line
.	répéter	
m	marquer	bloc a, b
CTRL-V	marquer	visuel
J	concaténer	des lignes
~	casser	lettre
g~w	casser	mot
gU$	majusculer	curseur-EOL
guw	minusculer	mot
CTRL-A	incrémenter	nombre
CTRL-X	décrémenter	nombre

couper

commande	portée
x	caractère
5x	5 caractères
dw	mot
5dw	5 mots
dd	ligne
5dd	5 lignes
D	curseur-EOL
d´a	marque a
:5,7d	lignes 5,6,7

copier

commande	commentaire
yw	copier un mot
y3w	3 mots
yy	une ligne
y7	7 lignes
y'z	la marque z
:10,20y	les lignes 10-20.
:'<,'>y	le bloc visuel.

coller

commande	action	commentaire
p	coller	sous le curseur
P	coller	au dessus
:10,20t 100	copier	lignes 10-20 vers 100
:10,20m 100	deplacer	lignes 10-20
:100,$m 50	deplacer	100-EOF vers 50.

remplacer

commande	action	portée
r	remplacer	caractère
cw	changer	mot
cc	changer	ligne
C	changer	curseur-EOL

substituer

commande	action	portée
:%s/motif/nouveau/g	substituer	fichier
:%s/motif/nouveau		une fois par ligne
:%s/motif/nouveau/gc	cofirmation	fichier
:10,20s/motif/nouveau/g		10 a 20.
:s/.*/U&/	masjusculer	ligne courante.
:s/motif/nouveau&/g		fichier
:/motif/-2,+4d	efface	contexte
:s/[ABC]/[abc]/g	A devient a	B devient

action contextuelle

commande :g/chaine/commande	action
:g/target/s/0/ZERO/g	remplace 0 par ZERO dans les targets
:g/^[ ]*$/d	supprime toutes les lignes vides
:g/target/t7	les lignes cibles en 7
:g/target/cd 80	centre les lignes ciblées
:g !/target/d	efface les lignes
:v/target/d	idem
:g/re/p	as `grep` !

oxford dictionary