UPMC

Master P&A/SDUEE

UE MP050

Méthodes Numériques et Informatiques - A

Introduction à l'environnement Unix

Jacques.Lefrere@aero.jussieu.fr

Sofian.Teber@lpthe.jussieu.fr

Albert Hertzog

2014–2015

Table des matières

1 Introduction au système UNIX

1.1 Système d'exploitation

– ensemble de programmes d'un ordinateur servant d'interface entre le matériel et les logiciels applicatifs
– abrégé SE (en anglais *operating system*, abrégé OS)
– exemples : MS-DOS, Windows (XP, 7, ...), famille UNIX (linux, Mac OS, ...)

Linux aujourd'hui dominant dans le calcul intensif :

plus de 90% des calculateurs du TOP 500

`http://i.top500.org/stats`

1.2 Historique

- depuis les années 1970 (MS-DOS : années 1980)
- grande diffusion assurée grâce à la portabilité du langage C
- plusieurs branches de développement (BSD et System V)
- rôle essentiel des milieux universitaires dans la diffusion d'unix
- normalisation POSIX (*Portable Operating System Interface*)
- système ouvert : implémentations sur diverses architectures

  du portable au super-calculateur
  - propriétaires (Mac OS, aix, hp-ux, solaris, ...)
  - libres (**linux**, net-bsd, free-bsd, ...)

    plusieurs distributions linux : debian, ubuntu,

    Red-Hat, mandriva, puis mageia, scientific-linux...

$\Rightarrow$ quelques différences dans les commandes (ex. : `ps`, impression `lpr`/`lp`, ...)

mais surtout au niveau administration (gestion des packages par ex.)

1.3 Principales caractéristiques du système UNIX

- interactif
- multi-tâches
- multi-utilisateur (dont le super-utilisateur)

  $\Rightarrow$ système de droits d'accès aux fichiers
- documentation en ligne (`man`, `info`, ...)
- intégration dans le réseau
  - partage de ressources (fichiers, authentification, ...)
  - applications réparties
- chaînage des processus par les tubes (pipes)

  $\Rightarrow$ assemblage d'outils élémentaires pour accomplir des tâches complexes
- l'interpréteur (**shell**) intègre un **langage de programmation**

  $\Rightarrow$ programmes interprétés en shell = shell-scripts

1.4 L'interpréteur de commandes ou shell (sh, bash, zsh, ...)

Le shell est l'interface utilisateur du système d'exploitation.

Deux familles (liées aux 2 branches d'unix) avec deux syntaxes différentes (en particulier dans la programmation) et des fichiers de configuration différents :

	versions libres	fichier de configuration
sh, ksh (Korn)	pdksh	.profile .kshrc
	bash (linux)	**.bash_profile .bash_login**
		.profile .bashrc
	zsh	.zprofile .zlogin .zshrc
csh	tcsh (ancien mac-os-X)	.login .[t]cshrc

sh : Bourne Shell (historique). bash : Bourne Again Shell (amélioration de sh).

1.5 Compte utilisateur

- identifiant (ou *login*)
- mot de passe (ou *password*)
- un groupe parmi ceux définis sur la machine
- un répertoire d'accueil personnel (ou *home directory*)
- un « interpréteur de commandes » (ou *shell*) :
  sh, `ksh`, **bash**, **csh**, **tcsh** ou **zsh**.

L'ensemble de ces informations est stocké dans un fichier système (souvent `/etc/passwd`).

Ressources limitées, par exemple par quota sur le disque

$\Rightarrow$ problème de connexion en mode graphique si quota atteint.

1.6 Sessions unix

- deux types de session :
  - mode texte (console, accès distant, ...) : ligne de commande
  - mode graphique (multi-fenêtres) : icônes et menus pour lancer les applications

    (dont les consoles `konsole` et `xterm` par exemple)

    gestionnaires de fenêtres : `fvwm`, **`kde`**, `gnome`, **`icewm`**...
- point commun
  - identification (*login*)
  - authentification (*password*)

Sous linux, en cas de problème en mode graphique,

passage en mode texte par frappe simultanée de

Ctrl | Alt | F1 (6 consoles de `F1` à `F6`).

Retour en mode graphique par Ctrl | Alt | F7 ou Ctrl | Alt | F8

2 Le shell : introduction

Le shell est un programme qui interprète les commandes saisies dans le terminal.

2.1 Syntaxe de la ligne de commandes

Le shell découpe la ligne de commande en **mots** séparés par des blancs

plus généralement par l'`IFS` (***I**nput **F**ield **S**eperator*)

- premier mot = **la commande**
- mots suivants = **les paramètres**
- paramètres optionnels introduits par « `-` »

`mkdir`	`-v`	`Rep`
commande	option	autre paramètre

Exemples de commandes élémentaires :

commande	affichage
date	de la date
whoami	du login
hostname	du nom de la machine
who	de la liste des utilisateurs connectés
echo "chaine de caracteres"	d'une chaîne
id	du numéro d'utilisateur
uname	du nom du système d'exploitation

Le shell

– distingue les **majuscules** des **minuscules**

– interprète certains **caractères** dits **spéciaux**

(éviter les blancs dans les noms de fichiers et de répertoires)

2.2 Caractères spéciaux pour le shell (*, ?, [. . .], ...)

– Générateurs de noms de fichiers (jokers)

?	un caractère quelconque dans un nom de fichier
*****	une chaîne de caractères quelconque dans le nom d'un fichier (y compris une chaîne vide)
[...**]**	**un** caractère quelconque pris dans la liste entre crochets
[c_1**−**c_2**]**	**un** caractère quelconque entre c_1 et c_2 dans l'ordre lexicographique
[!...**]**	**un** caractère quelconque pris *hors* de la liste

– Caractères de contrôle

^C	interruption du processus en cours
^Z	suspension du processus en cours
^? ou **^H**	effacement du dernier caractère (choix par `stty erase ^?`)
^D	fermeture du flux d'entrée (fin de session en shell)

– plus beaucoup d'autres (voir chapitres suivants)

Exemples de caractères jokers

`*`	tous les fichiers du répertoire courant
`*.c`	tous les fichiers dont le nom finit par **`.c`**
`*.*`	tous les fichiers dont le nom comporte un point
`data??`	tous les fichiers dont le nom est **`data`** suivi de deux caractères
`f.[abc]`	les fichiers **`f.a`**, **`f.b`**, et **`f.c`** s'ils existent
`f.[0-9]`	les fichiers dont le nom s'écrit **`f.`** suivi d'un chiffre
`f.[!0-9]`	les fichiers dont le nom s'écrit **`f.`** suivi d'un caractère qui ne soit pas un chiffre
`*.[ch]`	tous les fichiers dont le nom finit par **`.c`** ou **`.h`** s'ils existent

2.3 Historique des commandes

Accès à l'historique des commandes avec les **flèches haut et bas**.

Modification d'une commande avec les flèches **gauche et droite**.

2.4 Complétion de noms de fichier ou de commande

La touche Tab permet :

- de **compléter automatiquement** les noms de commande ou de fichiers en l'absence d'ambigüité
- de **visualiser les différentes possibilités** en cas d'ambigüité (en pressant 2 fois Tab)
- de **s'apercevoir d'une erreur** en cas de non réponse du shell

2.5 Documentation en ligne (man)

Différents moyens d'accéder à la documentation en ligne :

– **man** cmd : affiche la page du manuel de la commande cmd

  Pour se déplacer dans le manuel : flèches haut et bas

  Pour sortir : touche q

  Pour rechercher un motif : \motif

– cmd --help : affiche l'aide en ligne

– info cmd : complémentaire à man

Rechercher quelle commande utiliser pour une opération : man **-k** *motclef*

Autre source d'information : usage **averti** d'un moteur de recherche sur le web

3 Hiérarchie des fichiers unix

3.1 Arborescence

L'ensemble des fichiers est structuré sous la forme d'une hiérarchie de répertoires (*directories*) et de fichiers (*files*) constituant un arbre unique.

– / est la **racine** (*root*) : le répertoire qui contient tous les autres fichiers ;
– ses **nœuds** sont des sous-répertoires...
– ses **feuilles** sont les fichiers ordinaires (en général).
– le **séparateur** de niveaux est la barre oblique (*slash*)

Remarques concernant Windows :

– le séparateur est la contre-oblique \ (*antislash*).
  C : \ est la racine du disque dur,
  D : \ est la racine du lecteur CD, ...
– terminologie : dossier (*folder*) à la place de répertoire.

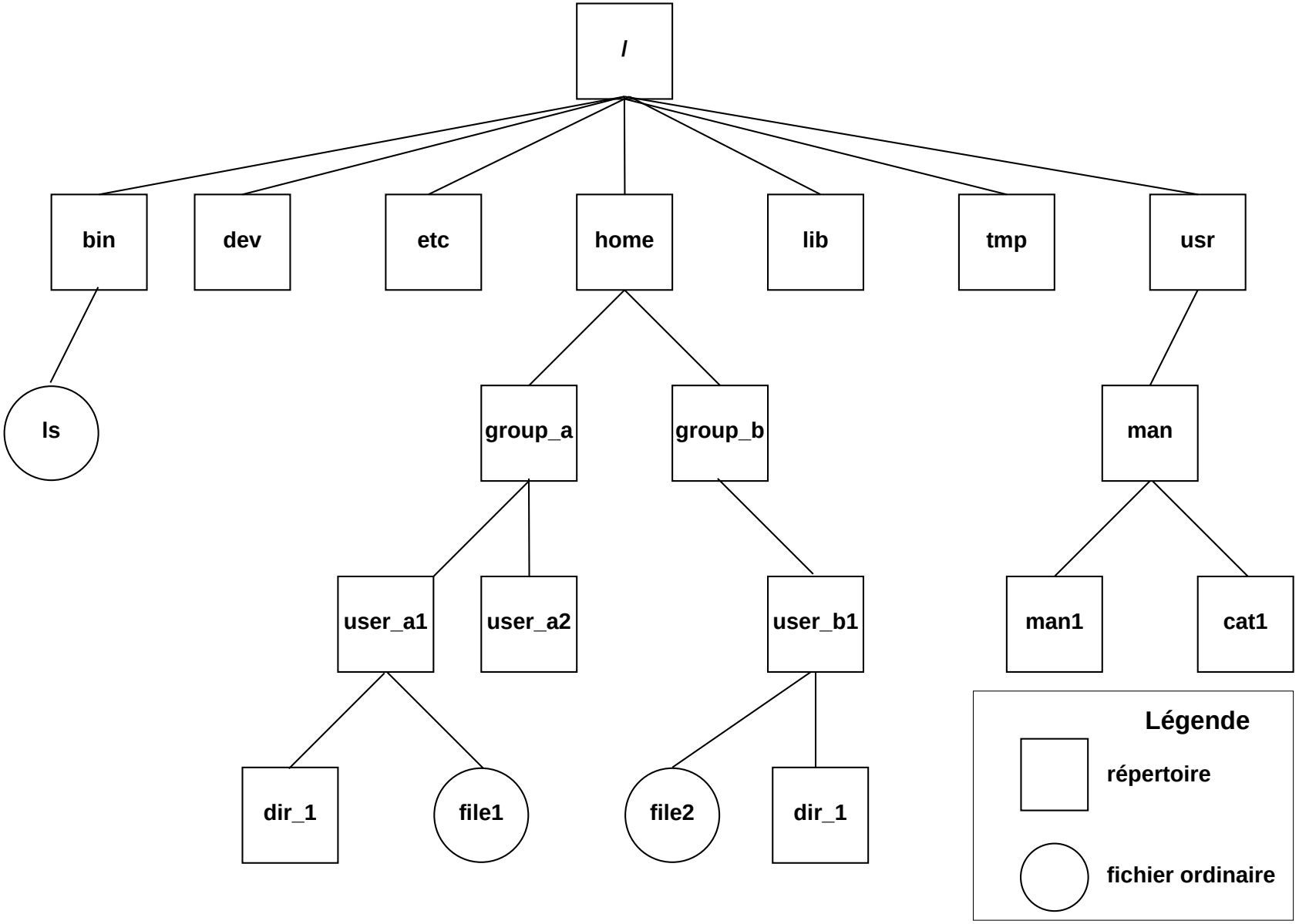

FIG. 1 – Arborescence des fichiers UNIX

3.2 Chemin d'accès (*path*)

– **chemin absolu** : commence toujours par `/` et comporte la liste complète des répertoires traversés depuis la racine,

Exemple : **répertoire d'accueil** ou **répertoire personnel** de `user_a1` : `/home/group_a/user_a1`,

– **un chemin relatif** : comporte la liste des répertoires à parcourir **depuis le répertoire courant** jusqu'au fichier ou répertoire choisi.

Il ne commence jamais par `/` et doit passer par un nœud commun à la branche de départ (répertoire courant) et la branche d'arrivée.

- `.` **répertoire courant**
- `..` **répertoire père**

Exemples, partant de `/home/group_a/user_a1` :
`dir_1`, `../`, `../user_a2`, `../../group_b/user_b1`,

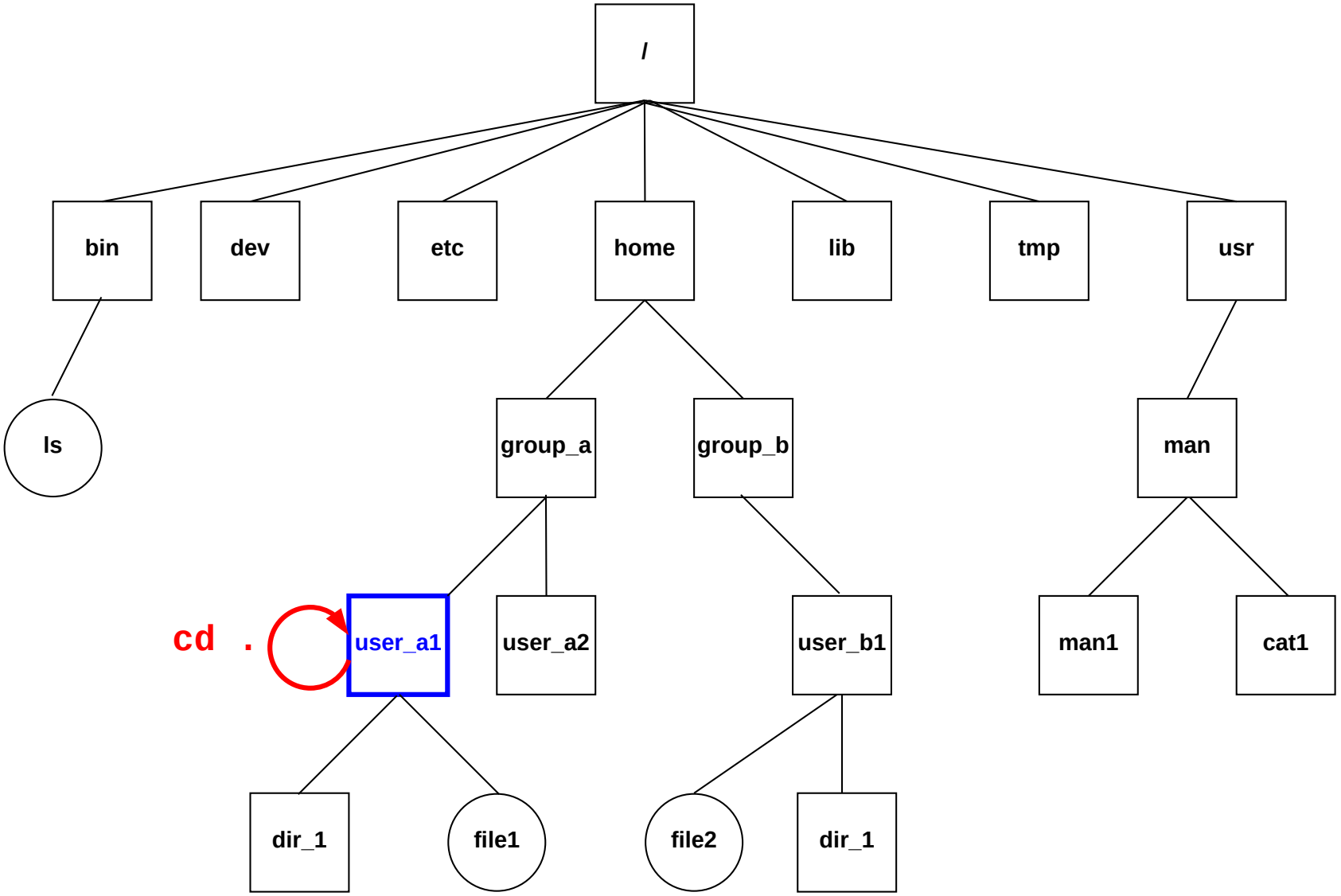

FIG. 2 – La commande `cd .` laisse dans le répertoire courant `/home/group_a/user_a1`.

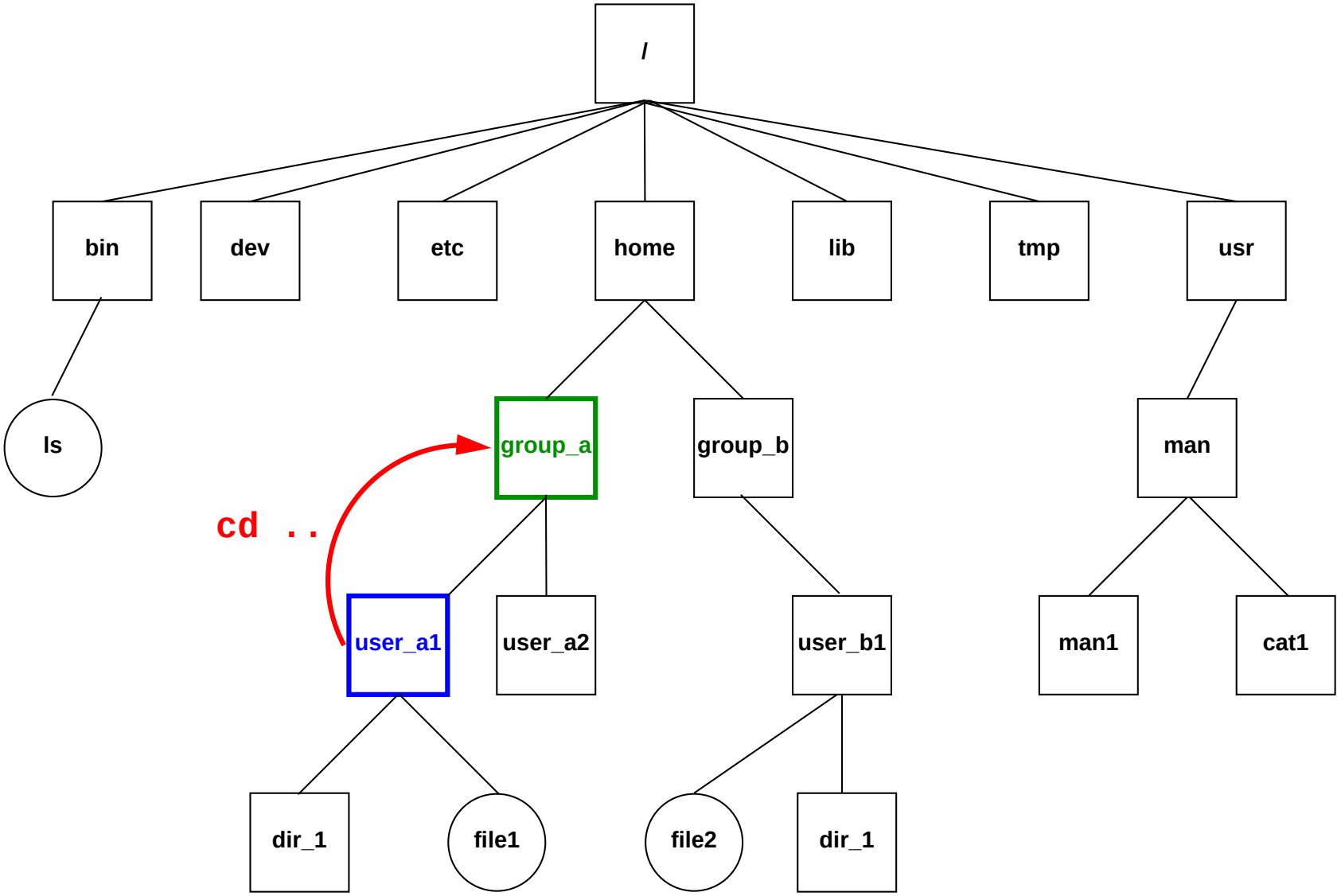

FIG. 3 – La commande `cd ..` déplace dans le répertoire père `group_a`.

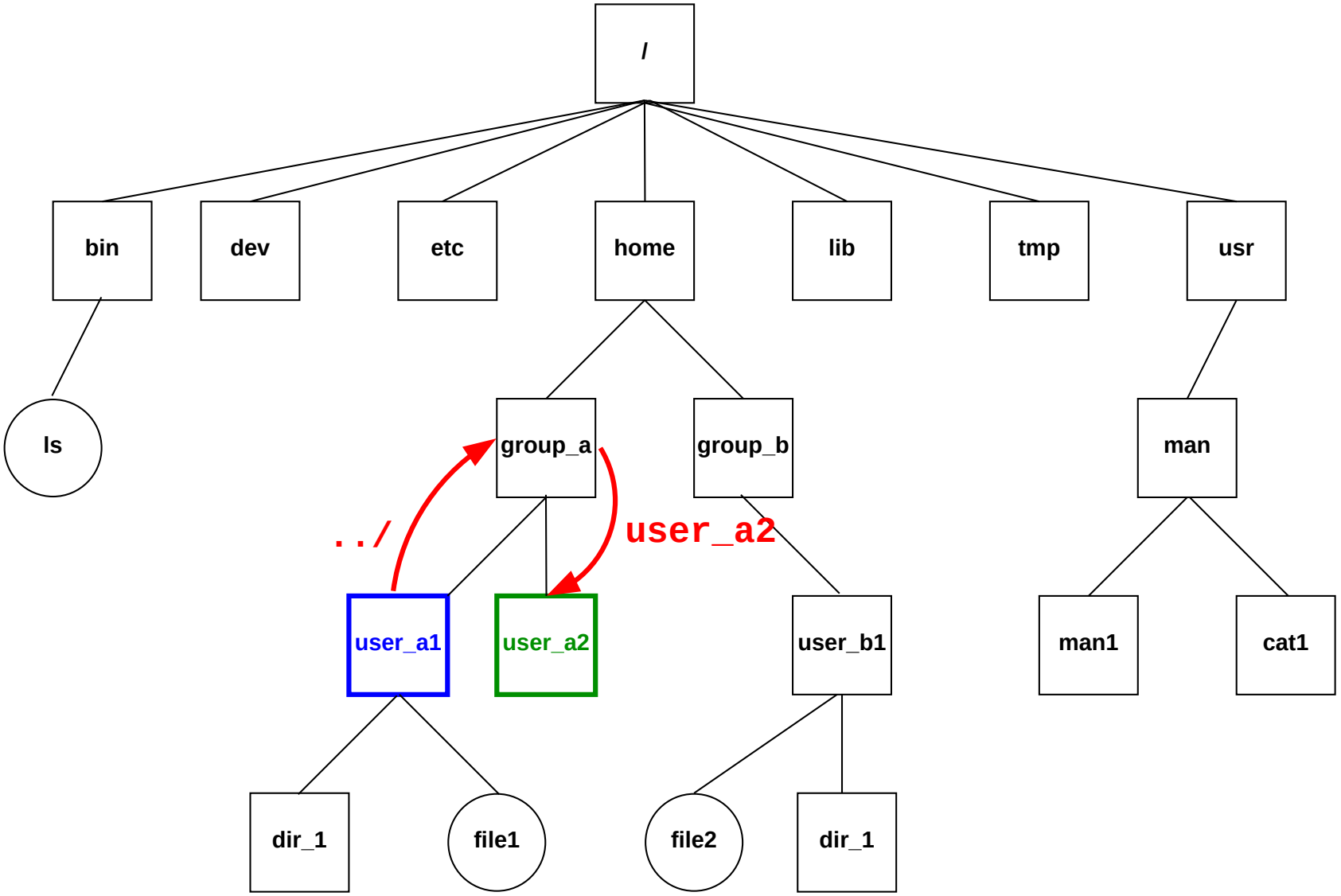

FIG. 4 – La commande `cd ../user_a2` déplace dans le répertoire `user_a2`

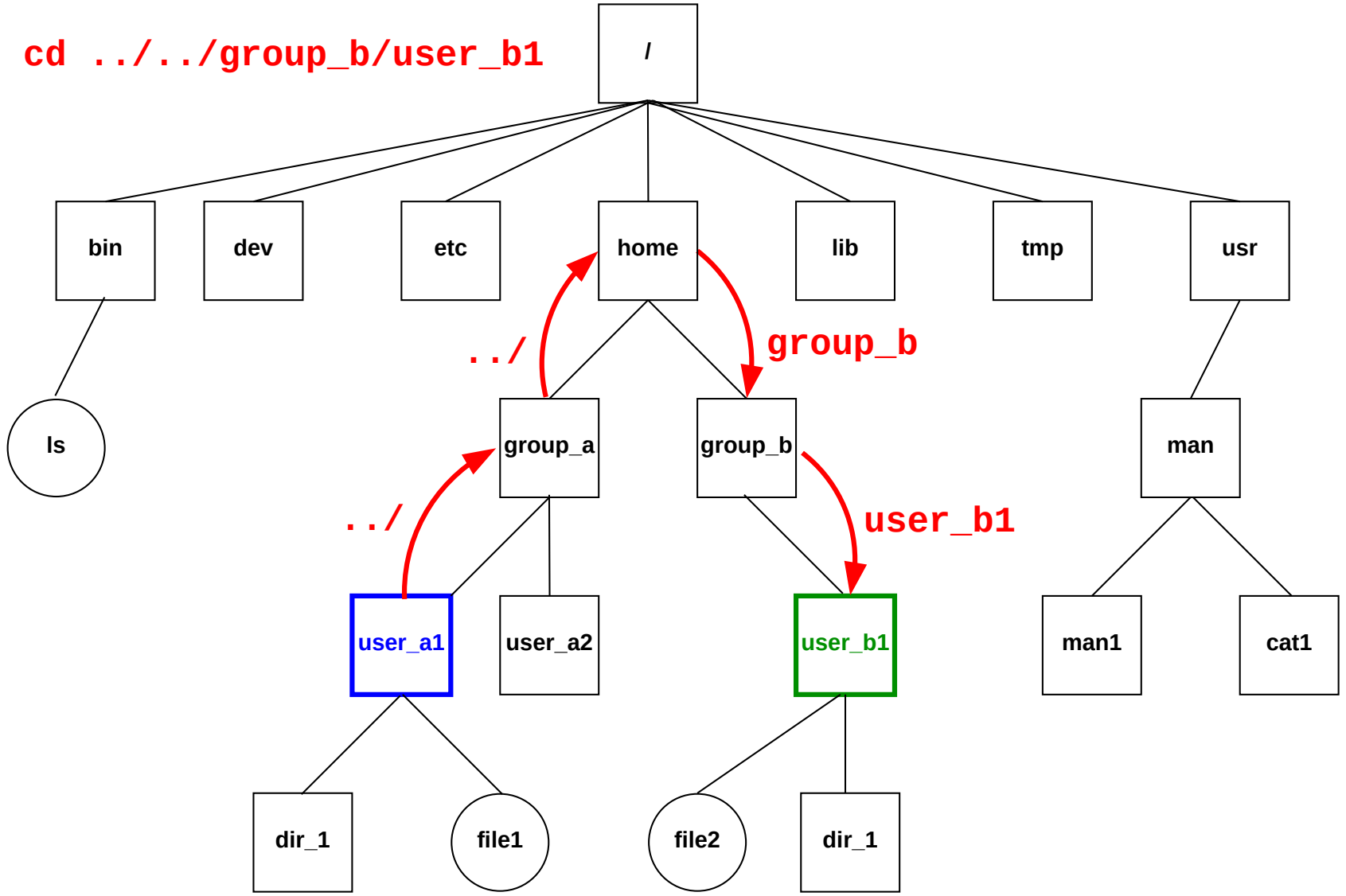

FIG. 5 – La commande `cd ../../group_b/user_b1` déplace dans le répertoire `user_b1`.

3.3 Raccourcis pour les répertoires d'accueil (~ et ~user)

Chemins en fait absolus :

- **~user** répertoire d'accueil d'un utilisateur quelconque
- **~** son propre répertoire d'accueil

Exemples :

`~/.profile`

est le chemin absolu de votre fichier d'initialisation personnel.

`~lefrere/M1/Doc/unix/poly-unix/`

est le chemin absolu du répertoire du polycopié UNIX,

situé sous le compte de l'utilisateur `lefrere`.

4 Commandes de base

4.1 Commandes de gestion de fichiers

4.1.1 Liste de fichiers (et répertoires) (`ls`)

`ls` [*-options*] [*liste_de_fichiers*]

`-a` (***a**ll*) liste aussi les fichiers cachés (de nom commençant par `.`)

`-l` (***l**ong*) affiche les attributs (droits, taille, date, ...) des fichiers

`-R` (***R**ecursive*) affiche la liste des fichiers contenus dans tous les sous répertoires éventuels

`-F` (***F**lag*) marque les fichiers répertoires (`/`), exécutables (`*`) ou les liens (`@`)

`-t` (***t**ime*) classe la liste par ordre anti-chronologique

`-d` (***d**irectory*) affiche le nom des répertoires mais pas leur contenu

`-h` (***h**uman readable*) affiche la taille en indiquant l'unité k (kilo-octet), M (méga), G (Giga)

`-r` (***r**everse*) inverse l'odre de classement (décroissant)

4.1.2 Copie de fichiers (`cp`)

en anglais ***copy***

Copie d'un (ensemble de) fichier(s)

`cp` [*-options*] *fichier_origine repertoire_cible*

`cp` [*-options*] ***liste***_*de_fichiers repertoire_cible*

Copie d'un fichier avec modification du nom

`cp` [*-options*] *fichier_origine fichier_cible*

Principales options :

`-i` (***i****nteractive*) demande de confirmation si *fichier_cible* existe déjà

`-r` (***r****ecursive*) copie d'une branche

4.1.3 Déplacement et renommage de fichiers (`mv`)

en anglais ***move***

Déplacement d'un fichier ou d'un répertoire

Le `répertoire_cible` *doit exister* au préalable

```
mv [-options] fichier_origine repertoire_cible
mv [-options] repertoire_origine repertoire_cible
mv [-options] liste_de_fich_ou_rep repertoire_cible
```

Déplacement d'un fichier ou d'un répertoire avec renommage

Le `répertoire_cible` *ne doit pas exister* au préalable

```
mv [-options] fichier_origine fichier_cible
mv [-options] repertoire_origine repertoire_cible
```

Principale option :

`-i` (***i**nteractive*) demande de confirmation interactive

4.1.4 Suppression de fichiers (`rm`)

en anglais ***rem**ove*

`rm` `[-`*`options`*`]` *`liste_de_fichiers`*

Principales options :

`-i` (***i**nteractive*) demande de confirmation interactive

`-r` (***r**ecursive*) destruction d'une branche de l'arborescence

4.2 Commandes de gestion de répertoires

4.2.1 Affichage du répertoire courant (`pwd`)

`pwd` (***p**rint **w**orking **d**irectory*) affiche le chemin absolu du répertoire courant

commande interne (*builtin*) du shell

4.2.2 Changement de répertoire courant (`cd`)

`cd` `[répertoire]` (***c**hange **d**irectory*)

commande interne (*builtin*) du shell

`cd` (sans paramètre) retour au répertoire d'accueil `~/.`

`cd -` revient au précédent répertoire

`cd ..` revient au répertoire père

4.2.3 Création de répertoire (`mkdir`)

`mkdir` *`répertoire`* (***make directory***)

option `-p` (***p**arent*) : crée les répertoires parents si nécessaire

exemple : `mkdir -p dir/subdir`

4.2.4 Suppression de répertoire (vide) (`rmdir`)

`rmdir` *`répertoire`* (***rem**ove **dir**ectory*)

refus de suppression si le répertoire contient des fichiers

$\Rightarrow$ utiliser `rm -R` *`répertoire`*, mais dangereux !

4.3 Accès au contenu des fichiers

4.3.1 Identification des fichiers (`file`)

`file` *`liste_de fichiers`*

affiche une indication sur la nature du fichier (texte, binaire, ...)

l'utiliser pour savoir avec quelles commandes consulter un fichier

4.3.2 Affichage du contenu de fichiers texte (`cat`)

`cat` [*`liste_de fichiers`*]

affiche (con**cat**ène) le contenu des fichiers de la liste (sans contrôle du défilement)

`cat` est le filtre identité : il recopie l'entrée sur la sortie

4.3.3 Affichage paginé du contenu d'un fichier texte (`more` et `less`)

`more` *`liste_de fichiers`*

affiche le contenu des fichiers de la liste (avec contrôle du défilement)

`less` *`liste_de fichiers`*

préférable sous linux (plus rapide que `more` pour les gros fichiers)

Requêtes sous le pagineur :

Entrée	avance d'une ligne
Espace	avance d'un écran
`b`	recule d'un écran (***b****ack*)
`/`*`motif`*	recherche la prochaine occurence de *`motif`*
`q`	quitte l'affichage

5 Supplément sur les commandes

5.1 Afficher une ligne de texte (`echo`)

`echo` [*-options*] *ligne_de_texte*

Commande très utilisée pour envoyer du texte sur l'entrée standard d'une autre commande via un **|** :

```
echo elmfjelf | tr 'e' 'a'
```

5.2 Différence entre deux fichiers (`diff`)

Trouver des différences entre 2 fichiers

`diff` [*-options*] *fich_1 fich_2*

option `-b` ignore les changements portant sur les blancs

option `-y` affiche en deux colonnes

5.3 Compression de fichiers (gzip)

Compression et décompression sans perte d'information

gzip [*-options*] *liste_de_fichiers*

Compression ⟶ fichier de suffixe `.gz`

Décompression d'un fichier de suffixe `.gz`

gunzip [*-options*] *liste_de_fichiers*
gzip -d [*-options*] *liste_de_fichiers*

Autre outil, plus efficace (mais plus lent), presque partout disponible :
bzip2/**bunzip2** ou **bzip2 -d**.

5.4 Archivage d'arborescence (`tar`)

`tar` *`options [archive] répertoire`*

Principales actions possibles (une et une seule) :

- **`-c`** (***c**reate*) création de l'archive à partir de l'arborescence
- **`-t`** (*lis**t***) liste des fichiers tels qu'ils seront extraits
- **`-x`** (*e**x**tract*) extraction des fichiers pour restaurer l'arborescence
- **`-z`** (*g**z**ip*) compression (ou décompression) de l'archive à la volée

Autres options combinables :

- **`-v`** (***v**erbose*) affiche des informations complémentaires
- **`-f`** *`archive`* (***f**ile*) précise le nom du fichier d'archive utilisé

Exemples de `tar`

`cd ~/.. ; tar -cvf /tmp/archive.tar user`

archive toute l'arborescence d'un utilisateur dans `archive.tar`

`tar -tvf /tmp/archive.tar`

affiche la liste des fichiers archivés dans `archive.tar`

`tar -xvf /tmp/archive.tar`

restaure l'archive dans le répertoire courant

`cd ~/.. ; tar -cvzf /tmp/archive.tar.gz user`

archive et compresse toute l'arborescence d'un utilisateur dans `archive.tar.gz`

Remarque : éviter les chemins absolus dans les sauvegardes, sinon les fichiers seront obligatoirement restaurés au même endroit.

5.5 Recherche de fichiers dans une arborescence (find)

find *répertoire_de_départ critère(s) action*

Recherche dans **toute** la hiérarchie en dessous du répertoire de départ.

Commande très puissante : **critères** de sélection nombreux

-name *motif*	nom selon un motif
-iname *motif*	nom selon un motif sans respect de la casse
-type *T*	de type donné (f=fichier ordinaire, d=répertoire)
-size *entier*	taille

L'**action** la plus usitée est :

-print	affiche la liste des fichiers (un par ligne)

Exemples de recherches avec find :

```
find . -name core -print
```

affiche la liste des fichiers nommés core sous le répertoire courant

```
find /tmp -size +1000c -size -2000c -print
```

affiche la liste des fichiers de taille entre 1000 et 2000 octets sous /tmp

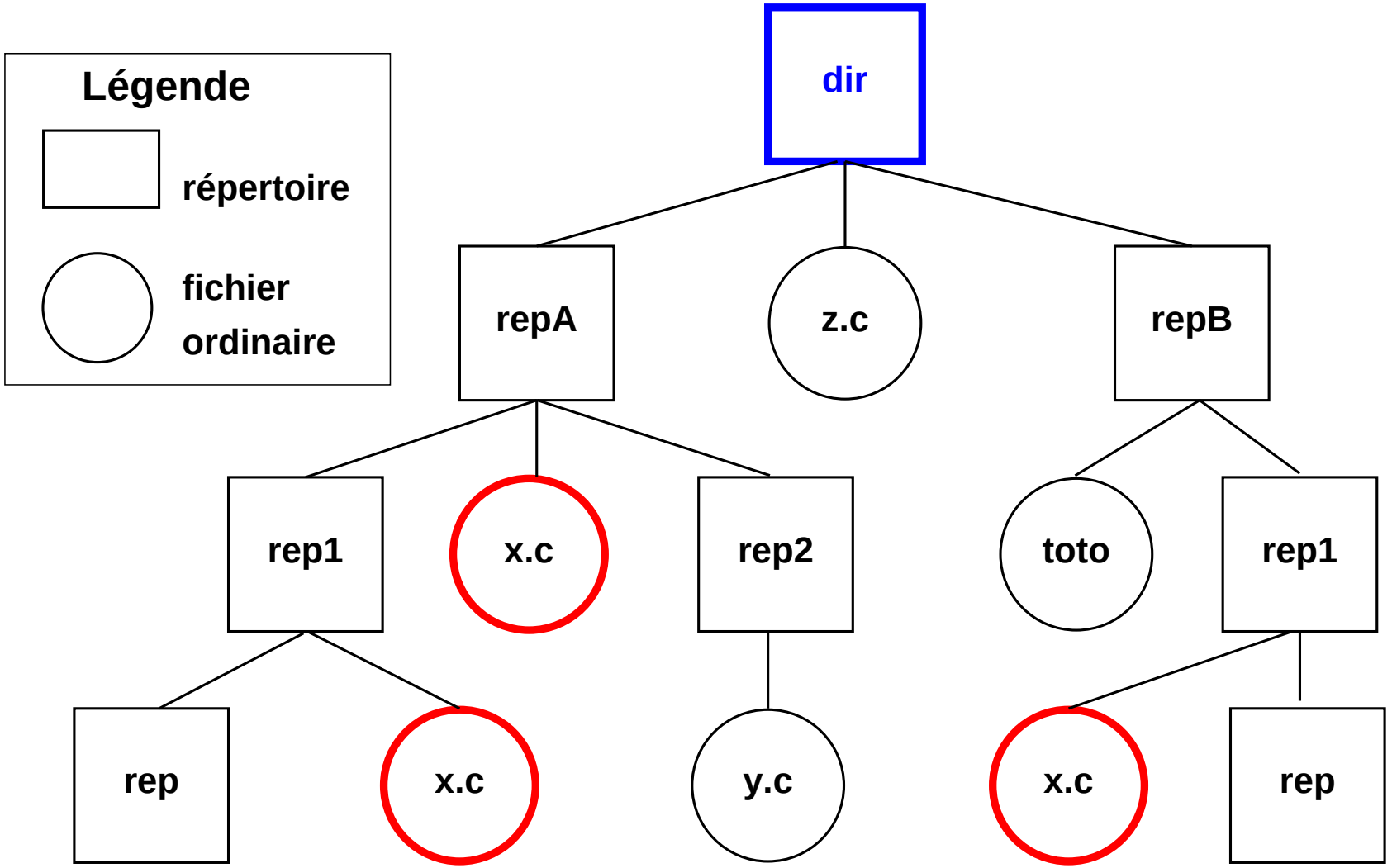

FIG. 6 – **`find . -name x.c -print`** si **`dir`** est le répertoire de travail

$\Rightarrow$ trois fichiers

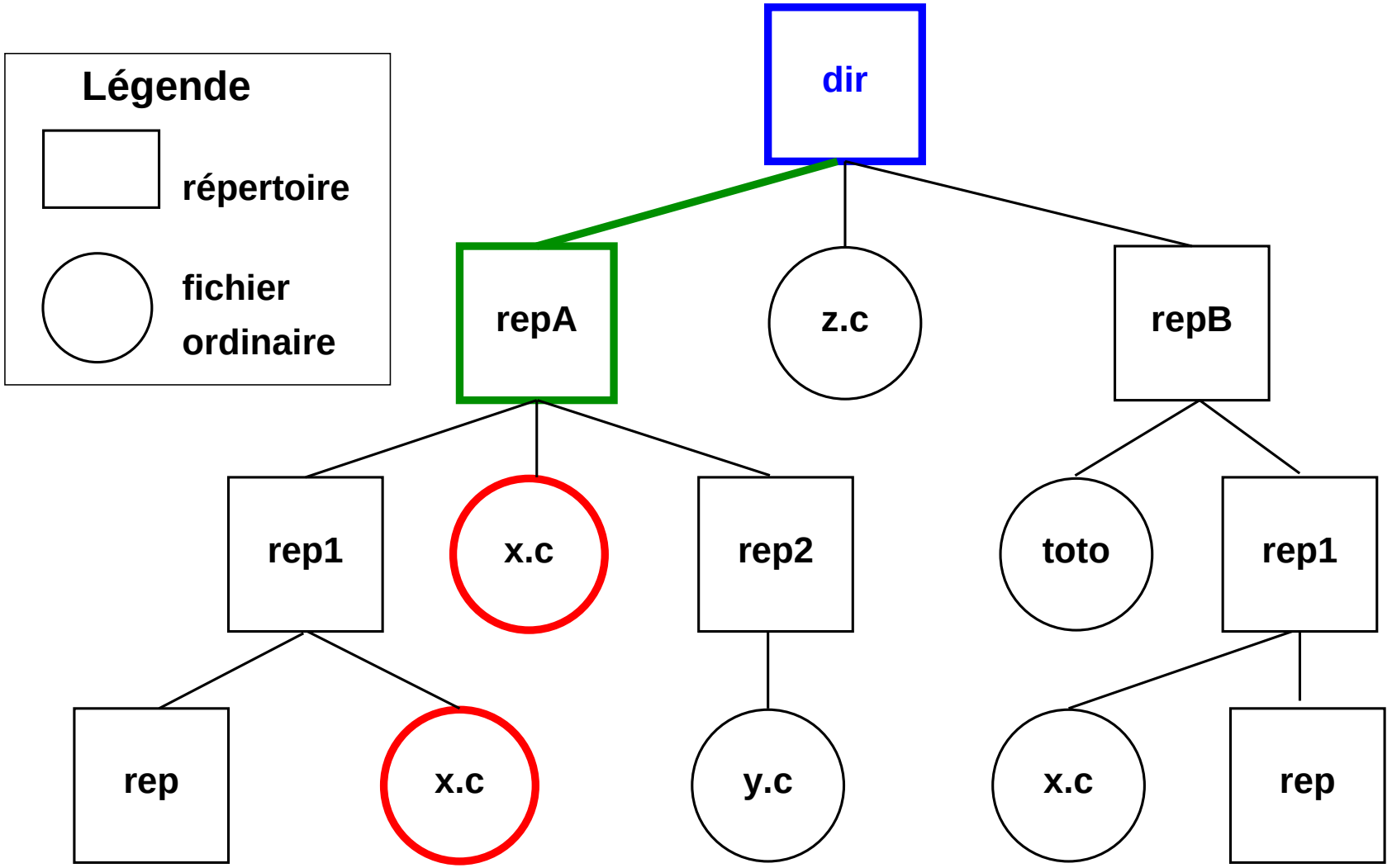

FIG. 7 – **find repA -name x.c -print** à partir de **repA**

⇒ deux fichiers

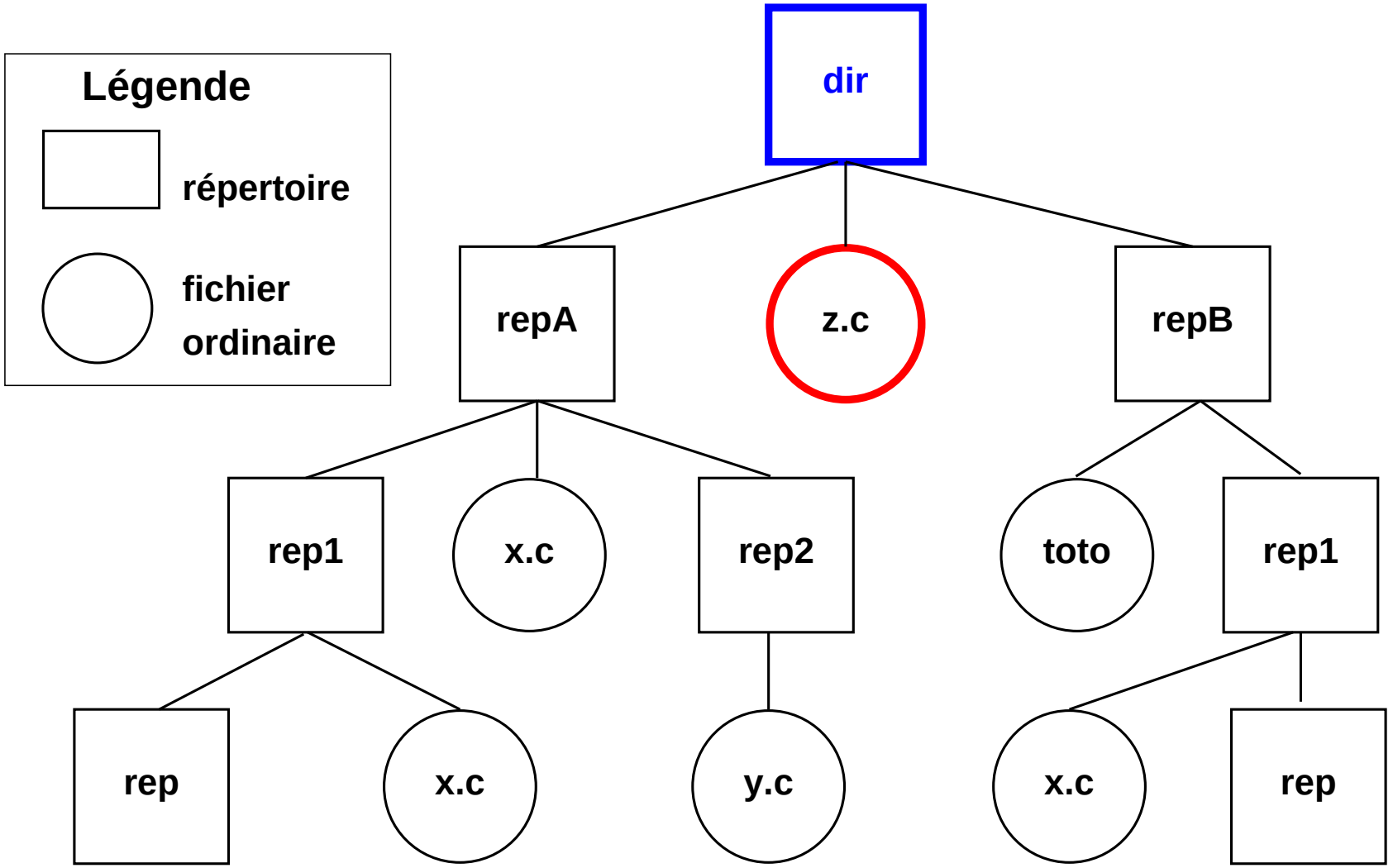

FIG. 8 – **find . -name *.c -print** ⇒ **z.c**

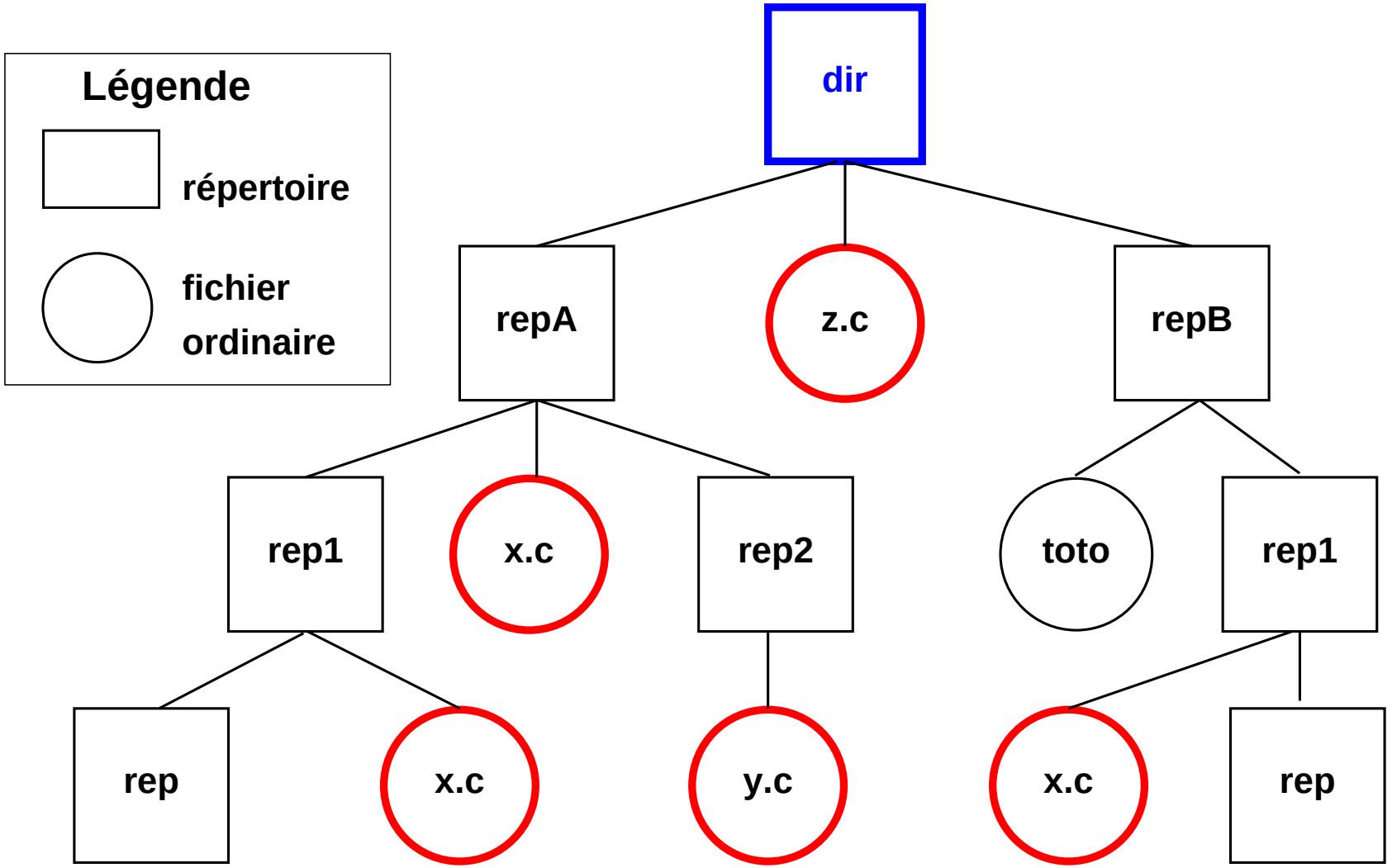

FIG. 9 – **`find . -name '*.c' -print`** ⇒ cinq fichiers

6 Droit d'accès aux fichiers

-/d/l	propriétaire	groupe	autres
	user	group	others
-	r w x	r w x	r w x

r	*read*	lecture
w	*write*	écriture
x	*execute*	exécution
-		aucun droit

6.1 Affichage des droits d'accès (`ls -l`)

Cas général : `ls -l` *`liste_de_fichiers`*

Exemple : `ls -l ~lefrere/M1/Config/`

```
drwxr-xr-x 2 lefrere personnel 1024 sep 17  2009 lisp
-rwxr-xr-x 1 lefrere personnel 1076 oct  7  2009 MNI.bash_profile
-rwxr-xr-x 1 lefrere personnel 3101 oct 22  2009 MNI.bashrc
lrwxrwxrwx 1 lefrere personnel   15 sep 15 17:40 motd -> motd.16sept2010
-rw-r--r-- 1 lefrere personnel  434 sep 15 21:18 motd.16sept2010
```

Première colonne : **d** si répertoire **l** si lien (*link*) symbolique (raccourci vers **->**)

6.2 Changement des droits d'accès (chmod)

chmod *mode liste_de_fichiers*

où *mode* représente la portée, **u**, **g**, **o**, ou **a**.

suivie de **=** (définit un droit), **+** (ajoute un droit), ou **−** (enlève un droit),

suivi de la permission **r**, **w**, ou **x**.

L'utilisateur doit être propriétaire du fichier pour en modifier les droits.

Exemples :

- `chmod go-r fichier`
  supprime les droits de lecture au groupe et aux autres
- `chmod u+w,go-w fichier`
  donne le droit en écriture au propriétaire et le supprime au groupe et aux autres.

Signification des droits sur les répertoires

- **`r`** nécessaire pour afficher la liste des fichiers du répertoire
- **`w`** permet d'ajouter, de renommer, de supprimer des fichiers dans le répertoire (pas nécessaire pour modifier le contenu d'un fichier)
- **`x`** permet d'agir sur les fichiers du répertoire, à condition de connaître leurs noms (même si on ne peut pas afficher leur liste) : par exemple traverser le répertoire

Exemple :

```
drwx--x--x  42 lefrere personnel 4096 sep 20 18:17 lefrere
```

peut être traversé par tout le monde pour accéder à `lefrere/M1/`

mais seul son propriétaire peut lister son contenu

7 Environnement réseau

7.1 Courrier électronique

Commandes de gestion du courrier :

– en mode texte : `mail`, `elm`, `pine`, `mutt`
  autres outils gérant le courrier : l'éditeur `emacs`
– en mode graphique : les navigateurs (`mozilla-thunderbird`, ...).
– à distance : accès à sa boîte aux lettres personnelle via un navigateur (après authentification) grâce à un service de `webmail`

Exemple d'adresse électronique :

***Prenom.Nom*@etu.upmc.fr**,

7.2 Connexion à distance (slogin et ssh)

Connexion sur une machine distante grâce à la commande sécurisée **slogin**.

Authentification sur la machine distante par mot de passe ou échange de clefs.

slogin *user@dist_host.domain*

Lancement de commandes sur la machine distante :

ssh *user@dist_host.domain dist_cmd*

7.3 Transfert de fichiers à distance (scp et sftp)

Échange de fichiers personnels entre deux machines, sans ouvrir de session sur la machine distante, via **scp**

Syntaxe de cp mais préfixer le chemin d'accès des fichiers distants par ***user@dist_host.domain:***

`scp [[`*`user1`*`@]`*`host1`*`:]`*`file1`* **`[[`*`user2`*`@]`*`host2`*`:]`*`file2`

Session **`sftp`** (***s**ecure **f**ile **t**ranfert **p**rotocol*)

`sftp` ***`user@dist_host.domain`***

Après authentification sur le serveur distant,
importation de fichiers distants (`get` *`dist_file`*),
exportation de fichiers vers la machine distante (`put` *`local_file`*)
`exit` ou `quit` pour terminer la session `sftp`.

7.4 Navigateurs

Explorateurs Web (`lynx`, `mozilla-firefox`, `opera`, `konqueror`, `amaya`, ...)
Protocoles : `ftp` (***F**ile **T**ransfer **P**rotocol*), `http` (***H**ypertext **T**ransport **P**rotocol*), ou `https` (sécurisé par cryptage).

Ressources localisées grâce à une *URL* (***U**niversal **R**esource **L**ocator*).

Exemples d'*URL* :

file:`/home/lefrere/M1/Doc/unix/` sur la machine locale

http:`//www.formation.jussieu.fr/ars/2006-2007/UNIX/cours/`

http:`//www.w3.org/TR/xhtml1`

wget pour importer des documents depuis le web.

8 Redirections et tubes

8.1 Flux standard

Commande UNIX $\Rightarrow$ trois flux standard de données :

descripteur	flux	défaut
0	entrée standard	le clavier
1	sortie standard	l'écran
2	sortie d'erreur standard	l'écran

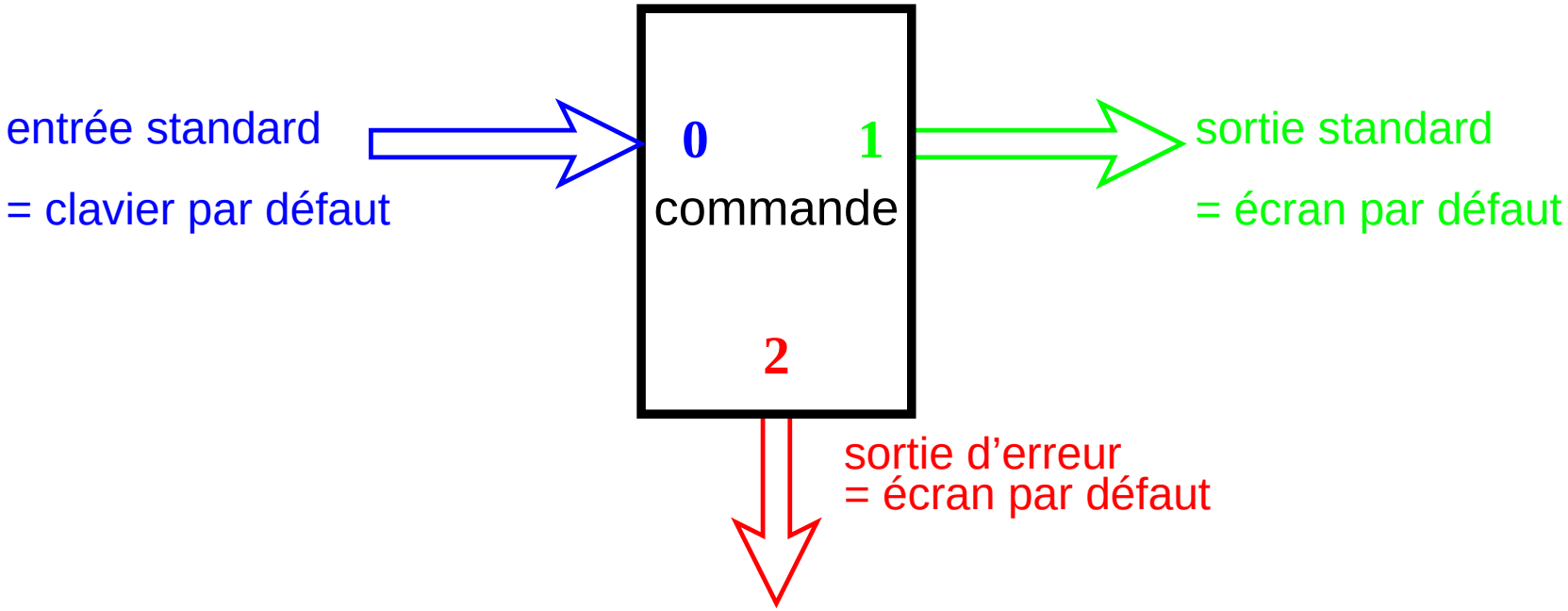

FIG. 10 – Flux standard de données associés à une commande

8.2 Redirections

Redirection des flux standards vers :

– des **fichiers**

(stockage/extraction d'information)

– les entrées-sorties d'**autres commandes**

(utilisation de tubes ou *pipes* permettant de combiner des commandes de base pour effectuer des traitements complexes)

$\Rightarrow$ souplesse du système UNIX

8.2.1 Redirection de sortie vers un fichier (> et >>)

syntaxe

```
commande > fichier
```

Le fichier résultat est créé.

syntaxe

```
commande >> fichier
```

pour ajouter le résultat à la fin du fichier

Exemples :

```
ls > liste.txt
date > resultats
echo fin >> resultats
```

```
ls *.f90 > liste_f+c
ls *.c >> liste_f+c
```

Attention : le shell interprète très tôt les redirections

`cat -n fic1 > fic1` (efface le contenu de `fic1`)

Solution : `cat -n fic1 > tmp ; mv tmp fic1`

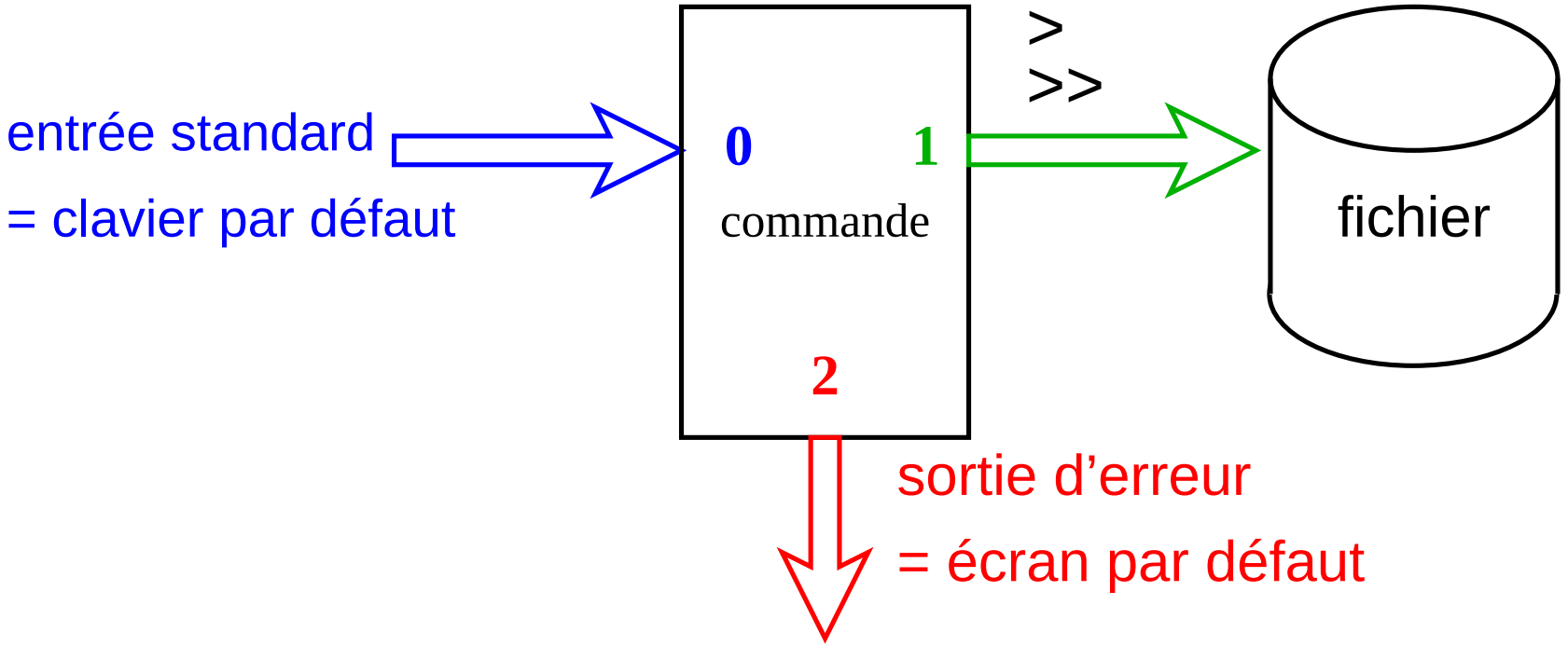

FIG. 11 – Redirection de la sortie

8.2.2 Redirection de l'entrée depuis un fichier (<)

syntaxe

commande < fichier

le fichier doit exister au préalable

Exemple : lecture des données d'entrée d'un exécutable sur un fichier au lieu de la saisie au clavier

```
a.out < entrees
```

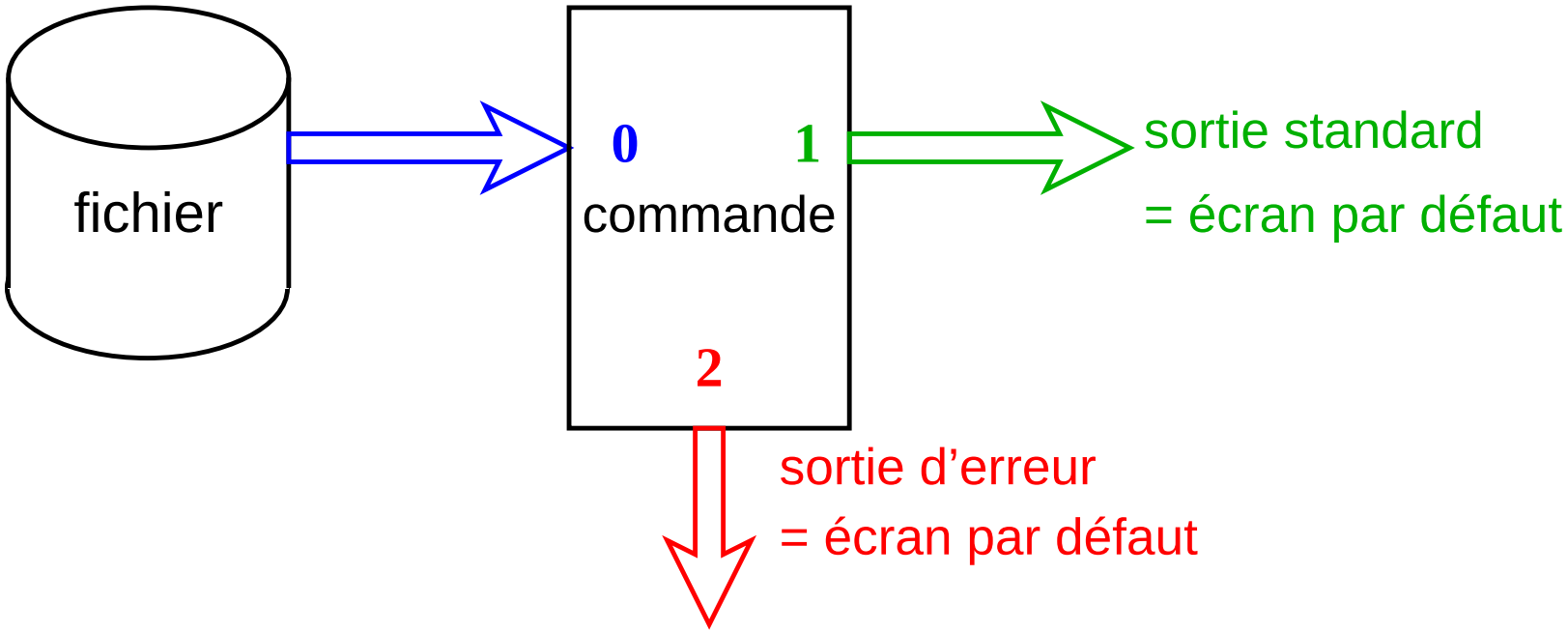

FIG. 12 – Redirection de l'entrée

8.3 Tubes ou *pipes* (|)

Appliquer deux traitements successifs à un flux de données :

– **Traitement séquentiel** avec création d'un fichier intermédiaire

```
commande_1 > fichier
commande_2 < fichier
rm fichier
```

– **Traitement à la chaîne** en connectant les deux processus par un **tube** (= zone mémoire) ⇒ sortie de la commande 1 synchronisée à l'entrée de la commande 2

syntaxe

```
commande_1 | commande_2
```

plus **rapide** que le traitement séquentiel

Exemple : affichage paginé des fichiers du répertoire courant

Méthode séquentielle (**à éviter**)

```
ls -l > liste
more liste
rm liste
```

Chaînage avec un tube (**à préférer**)

```
ls -l | more
```

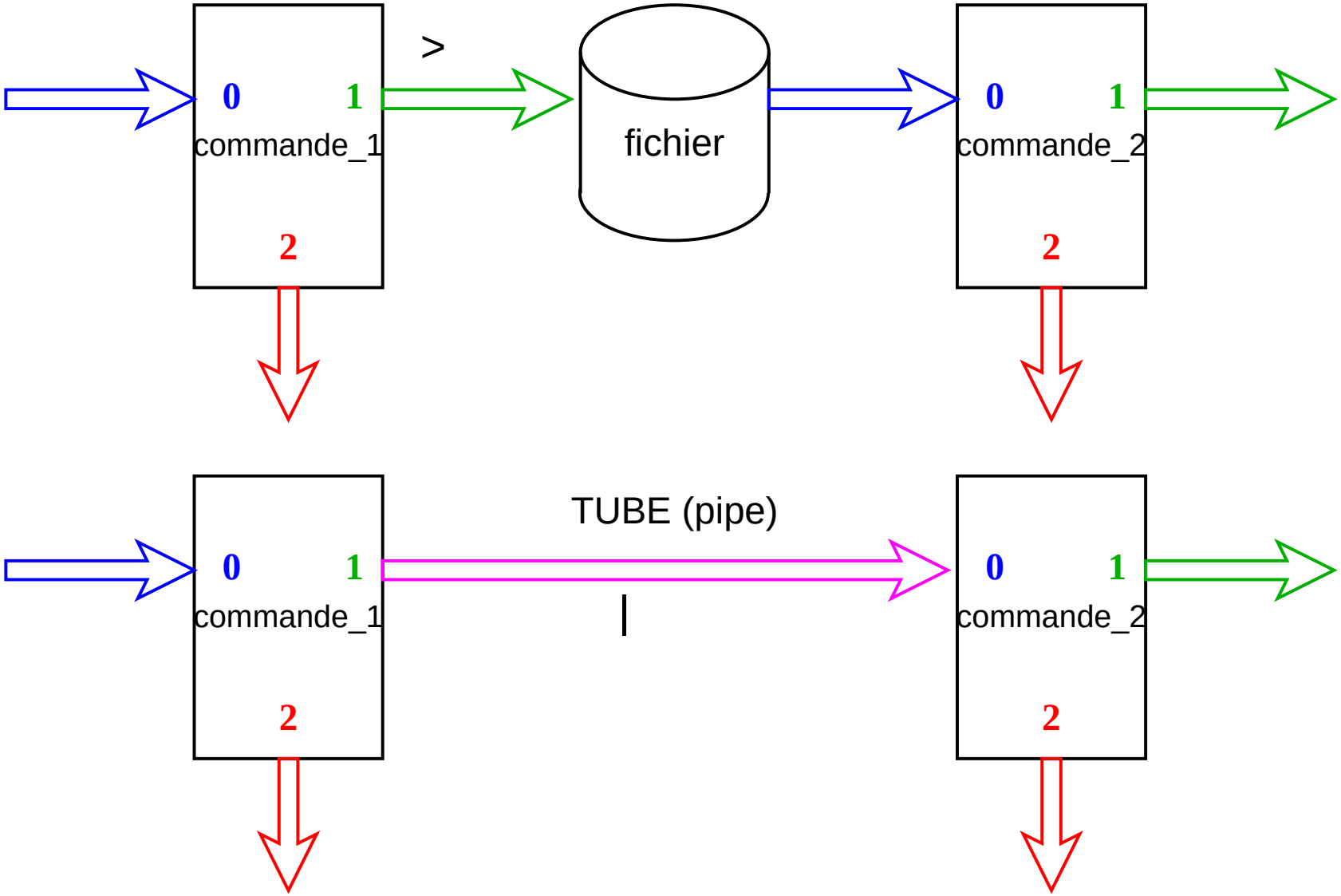

FIG. 13 – Tube ou pipe

8.4 Compléments

8.4.1 Redirection de la sortie d'erreurs vers un fichier (2> et 2>>)

syntaxe

```
commande 2> fichier
```

Attention : pas d'espace entre 2 et >

syntaxe

```
commande 2>> fichier
```

pour ajouter les erreurs à la fin du fichier.

Exemple : stockage des diagnostics d'une compilation dans un fichier pour éviter le défilement à l'écran (afin de localiser la première erreur)

```
gcc essai.c 2> erreurs
```

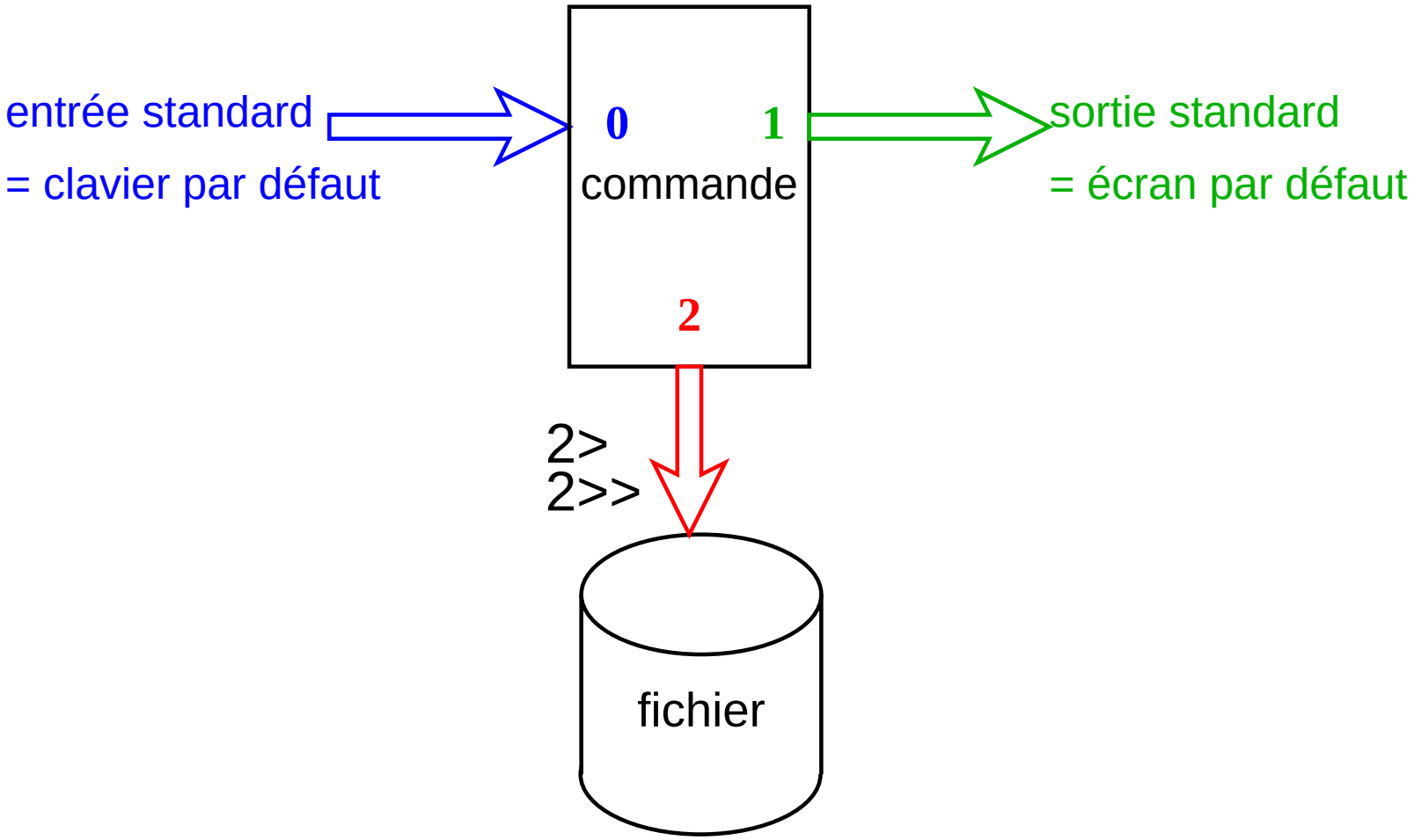

FIG. 14 – Redirection de l'erreur

8.4.2 Redirection de l'erreur standard vers la sortie standard (2>&1)

Regroupement dans un même flux de la sortie standard et de la sortie d'erreur :

syntaxe

```
commande 2>&1
```

Exemple (on suppose que `/etc/motd` est accessible) :

```
cat /etc/motd  /fichier_inexistant
```

affiche le mot du jour et un message d'erreur

```
cat /etc/motd  /fichier_inexistant > resultat
```

affiche un message d'erreur

```
cat /etc/motd  /fichier_inexistant > resultat 2>&1
```

n'affiche plus rien

N.-B. : la redirection de la sortie standard dans la dernière commande doit *précéder* la redirection de l'erreur standard vers le flux de la sortie standard.

8.4.3 Associations de redirections et tubes

Redirection de l'entrée et de la sortie (ordre indifférent avec une seule commande) :

```
commande < fichier_in > fichier_out
commande > fichier_out < fichier_in
```

Redirection de l'entrée, de la sortie, et de la sortie d'erreur :

```
commande < fichier_in > fichier_out 2> fichier_err
```

Redirection de l'entrée, de la sortie et tubes : pas de redirection sur des fichiers en sortie de la première commande ni en entrée de la seconde

```
commande_1 < entree | commande_2 > sortie
```

~~`commande_1 > sortie | commande_2 < entree`~~

8.4.4 Les fichiers spéciaux : exemple `/dev/null`

Répertoire **`/dev`** : *fichiers spéciaux* gérant des flux de données entre le calculateur et les périphériques (*devices*) : terminaux, imprimantes, disques, ...

`tty` affiche le nom du fichier spécial particulier attribué à un terminal
le fichier spécial **`/dev/tty`** désigne de façon générique le terminal attaché à la connexion.

`/dev/null` = fichier spécial « poubelle » (vide)
$\Rightarrow$ utilisé pour se débarrasser de certaines sorties inutiles.

syntaxe

```
commande 2> /dev/null
```

empêche le flux d'erreur de s'afficher à l'écran.

Exemple : `find rep -name "nom" -print 2> /dev/null`

évite l'affichage des messages d'erreur quand on tente d'accéder à des fichiers non autorisés.

9 Filtres

9.1 Définition

filtre = commande qui lit l'entrée standard, effectue des transformations sur ces données et affiche le résultat sur la sortie standard.

Exemples de filtres : `cat`, `wc`, `tail`, `head`, `tr`, `sort`, `grep`, `sed`, `awk`...
mais `ls`, `date`, `vi`... ne sont pas des filtres.

9.2 Utilisation

– le filtre lit l'entrée standard (clavier) :

`filtre` (`^D` pour arrêter le flux d'entrée)

– le filtre lit un (ou des) fichier(s) par redirection :

`filtre` `< fichier`

– le filtre est synchronisé à la sortie d'une autre commande via un tube :

`cat [`*`liste_de fichiers`*`]` | **`filtre`**

– le filtre agit comme une commande ordinaire (***syntaxe conseillée si elle existe***) :

`filtre` `[`*`liste_de fichiers`*`]`

La dernière syntaxe est généralement implémentée (sauf pour `tr`).

10 Filtres élémentaires

10.1 Comptage des mots d'un fichier texte (`wc`)

`wc` `[-mcwl] [liste_de fichiers]` (***w*ords*c*ount**)

options (ordre d'affichage **fixe** **`lwmc`** du plus gros au plus petit) :

`-l` compte les lignes (***l**ines*)

`-w` compte les mots (***w**ords*)

`-m` compte les caractères

`-c` compte les octets (***c**haracters* ! ! !)

Attention : le nombre d'octets et le nombre de caractères sont égaux si l'encodage des caractères est conforme aux normes ISO 8859-* ou ASCII. Ils peuvent différer si l'encodage suit la norme Unicode (par ex, UTF-8).

10.2 Classement (`sort`)

`sort` trie, regroupe ou compare toutes les **lignes** des fichiers passés en paramètre

Par défaut : ordre lexicographique sur tous les champs de la ligne

Options :

- **`-r`** (***r**everse*) pour trier selon l'ordre inverse
- **`-f`** pour ignorer la casse (majuscule/minuscule)
- **`-b`** (***b**lank*) pour ignorer les blancs en tête de champ
- **`-n`** (***n**umeric*) pour trier selon l'ordre numérique
- **`-u`** (***u**nique*) pour fusionner les lignes ex-æquo
- **`-k`** *`début,fin`* classement selon les champs compris entre *`début`* et *`fin`*
- **`-t`** *`délim`* choisit le séparateur de champs *`délim`*

Exemples

`sort /etc/passwd`	classe les lignes du fichier `/etc/passwd` par ordre lexicographique
`ls -l | sort -k9,9`	classe les fichiers par ordre lexicographique
`ls -l | sort -k9,9r`	classe les fichiers par ordre lexicographique inverse
`ls -al | sort -k5,5n`	classe les fichiers par ordre de taille croissante
`ls -l | sort -k5,5n -u`	classe les fichiers par ordre de taille croissante et n'affiche qu'un exemplaire pour une taille donnée (fusion des ex-æquo)
`wc -l * | sort -k1,1n`	classe les fichiers par nombre de lignes croissant
`du | sort -k1,1nr | more`	affichage paginé de la taille des fichiers et répertoires par ordre de taille décroissante

10.3 Début d'un fichier texte (head)

head [*options*] [*liste_de fichiers*] affiche *par défaut* les 10 premières lignes de la *liste_de_fichiers*

Options :

- **-n** *N* affiche les *N* premières lignes
- **-n** *-N* affiche tout sauf les *N* dernières lignes

Exemple

```
head -n 5 fichier
head -n -2
```

10.4 Fin d'un fichier texte (`tail`)

`tail` [*options*] [*liste_de fichiers*] affiche *par défaut* les 10 dernières lignes de la *liste_de_fichiers*

Options :

- **`-n`** *N* affiche les *N* dernières lignes
- **`-n`** *+N* affiche les dernières lignes à partir de la ligne *N*

Exemples

```
tail -n 5 fichier
tail -n +5 fichier
head -n 5 fichier | tail -n 1
```

10.5 Transcription (`tr`)

`tr` [*options*] *chaine_1* [*chaine_2*]

substitue à chaque caractère de *chaine_1* son correspondant de *chaine_2*.

Attention : filtre **pur**. N'admet pas de nom de fichier en paramètre ⇒ redirections.

Attention aux différences selon les systèmes unix.

`tr '123' 'abc'` change les `1` en `a`, les `2` en `b` et les `3` en `c`.

Options :

- **`-s`** (squeeze-repeats) élimine les répétitions de caractères
- **`-d`** (delete) élimine les caractères de la *chaine_1*

Compléments : Possibilité d'utiliser des *intervalles* de caractères :

```
tr a-z A-Z
```

des *classes* de caractères (`[:digit:]`, `[:alpha:]`, `[:alnum:]`) :

```
tr '[:lower:]' '[:upper:]'
```

de travailler sur les caratères de contrôle :

`tr -s '\n'` : élimine les sauts de lignes successifs

`tr -d '\r'` : élimine les caractères *Carriage Return* (des fichiers windows)

11 Expressions régulières

Recherche de chaînes de caractères qui satisfont à un certain motif (*pattern*)

$\Rightarrow$ syntaxe particulière pour décrire des motifs **génériques** :

une *expression rationnelle*

Expressions rationnelles utilisées par les éditeurs **`ex`**, **`vi`** et **`sed`**, ainsi que les filtres **`grep`** et **`awk`**.

Deux versions exclusives de la syntaxe :

– expressions rationnelles de base ***BRE : Basic Regular Expressions***

– expressions rationnelles étendues ***ERE : Extended Regular Expressions***

11.1 Signification des caractères spéciaux

. (point)	représente un caractère quelconque et un seul
\ (contre-oblique : *backslash*)	sert à protéger le caractère qui le suit pour empêcher qu'il ne soit interprété
* (étoile)	représente un nombre d'occurrences quelconque (**zéro**, une ou plusieurs occurrences) du caractère ou de la sous-expression qui précède

Ne pas confondre ces caractères spéciaux des expressions rationnelles avec les caractères génériques (*wildcards*) pour les noms de fichiers, `*` et `?` qui sont, eux, interprétés par le shell.

Exemples

`a*`	un nombre quelconque de fois le caractère `a` (y compris une chaîne vide)
`aa*`	une ou plusieurs fois le caractère `a`
`.*`	un nombre quelconque de caractères quelconques (y compris une chaîne vide)
`..*`	au moins un caractère
`\.`	un point suivi d'un caractère quelconque
`\\*`	un nombre quelconque (y compris zéro) de contre-obliques

11.2 Ancres

Les ancres (*anchor*) ne représentent aucune chaîne, mais permettent de spécifier qu'un motif est situé en début ou en fin de ligne :

^ (accent circonflexe : *carret*)	spécial **en début** de motif, représente le début de ligne
$ (dollar)	spécial **en fin** de motif, représente la fin de ligne

^a	une ligne commençant par un `a`
^a.*b$	une ligne commençant par `a` et finissant par `b`
^$	une ligne vide
^.*$	une ligne quelconque, y compris vide
^..*$	une ligne non vide
^$.*$$	une ligne commençant et finissant par `$` (contenant au moins deux fois `$`) seul le dernier `$` est spécial
^^.*^$	une ligne commençant et finissant par `^` (contenant au moins deux fois `^`) seul le premier `^` est spécial

11.2.1 Ensembles de caractères

Les ensembles de caractères (parmi lesquels **un et un seul** caractère quelconque peut être choisi) sont spécifiés entre crochets :
`[`*ensemble_de_caractères*`]`.
À l'intérieur d'un tel ensemble, les caractères spéciaux sont :

- `-` utilisé pour définir des **intervalles** selon l'ordre lexicographique (dépend des variables de langue)
- `^` en tête pour spécifier le **complémentaire** de l'ensemble
- `]` qui délimite la **fin** de l'ensemble, sauf s'il est placé en première position

À l'intérieur des ces ensembles peuvent figurer des **classes de caractères**
`[:lower:]`, `[:upper:]`, `[:alpha:]`, `[:digit:]`, `[:alnum:]`
Exemple : `[[:digit:]]` au lieu de `[0-9]` mais aussi `[-+.[:digit:]]`

Exemples

[a0+]	un des trois caractères `a`, `0` ou `+`
[a-z]	une lettre minuscule
[a-z:;?!]	une lettre minuscule ou une ponctuation double
[0-9]	un chiffre
[^0-9]	n'importe quel caractère qui n'est pas un chiffre
[]-]	un crochet fermant `]` ou un signe moins `-`

12 Le filtre grep

grep (***g**lobal **r**egular **e**xpression **p**rint*)

affiche les lignes qui contiennent un motif passé en paramètre

grep *motif* [*liste_de_fichiers*]

où *motif* est une expression régulière décrivant un motif générique

Principales Options :

- **-i** **i**gnore la casse (majuscule/minuscule)
- **-v** in**v**erse la sélection (affiche les lignes sans le motif)
- **-l** affiche la **l**iste des fichiers contenant le motif
- **-n** affiche les lignes contenant le motif précédées de leur **n**uméro
- **-c** (***c**ount*) affiche les noms des fichiers et le nombre de lignes qui contiennent le motif

Exemples :

grep lefrere /etc/passwd	affiche la ligne de cet utilisateur dans le fichier de mots de passe
grep 'printf' test.c	affiche les lignes comportant le mot `printf` dans `test.c`
grep -v '//' test.c	affiche les lignes qui **ne comportent pas** de // dans `test.c`
grep'**^**//' test.c	affiche les lignes qui **commencent** par // dans `test.c`
grep'**^** *//' test.c	affiche les lignes dont le **premier caractère non blanc** est // dans `test.c`
grep ';**$**' test.c	affiche les lignes qui se **terminent** par **;** dans `test.c`

13 Le filtre sed

sed (***s**tream **ed**itor*) : éditeur de flux non interactif :

- analyse **ligne par ligne** ce qui est saisi sur l'entrée standard, ou dans un (ou des) fichier(s), ou ce qui lui est envoyé par un tuyau (|)
- transforme les lignes selon des **requêtes**
- affiche le résultat sur l'entrée standard ($\Rightarrow$ pensez aux redirections > et >> pour sauvegarder le résultat).

Deux syntaxes possibles suivant la complexité du traitement :

sed -e '*requête_sed*' [*liste_de_fichiers*]

NB : Les requêtes comportant des caractères spéciaux sont la plupart du temps protégées par des apostrophes de l'interprétation par le shell.

sed -f *fichier_de_requêtes.sed* [*liste_de_fichiers*]

où *fichier_de_requêtes* contient des lignes de requêtes d'édition.

La plupart des requêtes sont adressables : on peut indiquer sur quelles lignes elles vont porter.

Exemples :

sed -e 's/toto/tutu/g'	change **tous** les `toto` de chaque ligne en `tutu`
sed -e 's/0/1/'	change **le premier** `0` de chaque ligne en `1`
sed -e 's/0/1/g'	change **tous** les `0` en `1`
sed -e '3,$s/0/1/g'	change **tous** les `0` en `1` à partir de la ligne 3
sed -e 's/0/(&)/g'	insère des parenthèses autour de **tous** les `0`
sed -e 's/[0-9]/(&)/g'	insère des parenthèses autour de **tous** les chiffres
sed -e '/motif/s/0/1/g'	change **tous** les `0` en `1` dans les lignes contenant `motif`

14 Le filtre awk

awk : filtre programmable

fonctionnalités de calcul de type tableur, syntaxe proche du langage C

comme `grep` et `sed`, agit **ligne par ligne** sur son entrée standard ou sur des fichiers

Syntaxes :

awk *instructions_awk liste_de_fichiers*

awk -f *fichier_programme liste_de_fichiers*

Autres Options :

- **-F** *délim* spécifie le séparateur de champ (blancs et tabulations par défaut)

14.1 Structure des données pour awk

Sur chaque ligne (**enregistrement**), les données sont découpées en **champs** selon le séparateur **FS** (***f**ield **s**eparator*)

- **NR** (***n**umber of **r**ecords*), est le numéro de ligne (d'enregistrement)
- **NF** (***n**umber of **f**ields*) est le nombre de champs
- **$0** l'ensemble de la ligne courante
- **$1**, **$2**, ... **$NF** sont le premier, deuxième, dernier champ

14.2 Structure d'un programme awk

Suite de couples **sélecteur** **{action}**

Un **sélecteur** peut être :

- vide et il est vrai pour toutes les lignes
- un `motif` entre **/** et **/**

le sélecteur est vrai si le `motif` est présent dans la ligne

- une expression logique évaluée pour chaque ligne
- une combinaison logique (via **`&&`**, **`||`** ou **`!`**) de sélecteurs
- un intervalle de lignes sous la forme : sélecteur1, sélecteur2
- **`BEGIN`** ou **`END`** qui introduisent des actions exécutées

  avant ou après la lecture des données

Une **action** est une suite d'instructions (affectations de variables, calculs, opérations sur des chaînes de caractères, ...) exprimées dans une syntaxe analogue à celle du langage C.

Nombreuses fonctions, notamment numériques et chaînes de caractères disponibles.

Variables non déclarées et typées seulement lors de leur affectation

14.3 Exemples de programmes et commandes awk

– affichage des lignes ayant 2004 pour premier champ

programme : **$1 == 2004 {print $0}**

commande : **awk '$1 == 2004 {print $0}' *fichier***

– affichage des lignes avec leur numéro (équivalent de cat -n)

programme : **{print NR, $0}**

commande : **awk '{print NR, $0}' *fichier***

– affichage du nombre de lignes du fichier (équivalent de wc -l)

programme : **END{print NR}**

commande : **awk 'END{print NR}' *fichier***

– échange des champs 1 et 2 :

{a=$1 ; $1=$2; $2=a; print $0}

– Calcul de la moyenne du champ 1 :

```
BEGIN{ n=0; s=0}
```
(initialisation facultative)

```
{n=n+1 ; s=s+$1}
```
(cumul)

```
END{ print "moyenne = ", s/n}
```
(affichage)

– Calcul de la moyenne des valeurs supérieures à 10 du champ 1 :

```
BEGIN{ n=0; s=0}
```
(initialisation facultative)

```
$1 > 10 {n=n+1 ; s=s+$1}
```
(cumul)

```
END{ if (n  > 0 ) {
      print "moyenne = ", s/n
                        }
      else {
      print "pas de valeurs > 10"
           }
   }
```
(affichage)

15 Fichiers texte : codage et édition

15.1 Fichiers informatiques

Un fichier informatique ordinaire est un lot d'informations, portant un nom et conservé dans une mémoire permanente (i.e. disque dur, CD, ...).

Bas niveau : tout fichier informatique est une séquences de **bits** (***bi****nary digi****ts***).

Haut niveau : les séquences binaires du fichier peuvent représenter du texte, une image, un code objet...

Les fichiers se caractérisent par :

– un nom et une extension qui permet d'indiquer la nature du contenu du fichier
  `.txt` (fichier texte), `.c` (fichier source C), `.o` (fichier objet), ...
– un chemin d'accès
– une taille généralement en **octets** avec un préfixe : kilo, mega, giga, ...

15.2 Fichiers texte et codages

On distingue :

– **les fichiers textes :** constitués de séquences binaires représentant les caractères selon un certain **codage**.
Ce type de fichier est destiné à être lu par l'utilisateur ; par exemple, le code source d'un programme, ...

– **les fichiers binaires :** code objet ou fichier pdf par exemple.

$\Rightarrow$ commande `file` pour identifier la nature du fichier

Les codages de caractères les plus courants sont :

– **ASCII** : norme la plus connue avec 128 caractères, chacun codé sur 7 bits.
Contient tous les caractères nécessaires pour écrire en anglais.

– **iso-8859** : extension de l'ASCII avec 191 caractères, chacun codé sur 1 octet.
Permet d'inclure des caractères accentués.

– **unicode** : permet de représenter des millions de caractères.
UTF-8 : les caractères sont codés sur 1, 2, 3 ou 4 octets.

15.3 Transcodage de fichiers textes (`iconv` et `recode`)

Outils de transcodage :

- **`iconv`** `-f` *`code_initial`* `-t` *`code_final`* *`fichier`*
- **`recode`** *`code_initial..code_final`* *`fichier`*

Exemples de trancodage de l'iso vers utf-8 :

```
iconv -f ISO-8859-1 -t UTF-8 < fic-iso.txt > fic-utf8.txt
recode 'ISO-8859-1..UTF-8' < fic-iso.txt > fic-utf8.txt
```

attention : par défaut `recode` travaille « en place » (modifie le fichier initial)

$\Rightarrow$ préferer la forme filtre (avec redirections)

15.4 Éditeurs sous unix (**vi**, **emacs**, **kate**, **kwrite**, ...)

- pleine page : (nécessitent une connaissance du terminal utilisé)
  - **vi** très puissant, présent sur tous les unix, mais assez peu intuitif
    version **vim** sous linux, éditeur sensible au langage (C, fortran, latex, ...)
    avec mise en valeur de la syntaxe par des couleurs
  - **emacs** encore plus puissant, mais plus gourmand en ressources
- en environnement graphique multifenêtres, avec menus, gestion de la souris, ...
  xemacs, **nedit**, **kwrite**, **kate**, ...

On n'utilise un éditeur que si l'on souhaite modifier un fichier (sinon préférer `cat`, `more`, `less`)

16 Gestion des processus

16.1 Affichage de la liste des processus (`ps`)

Processus = tâche élémentaire identifiée par un numéro unique ou *pid* (***p**rocess **id**entifier*).

Afficher la liste des processus avec la commande **`ps`**.

Trois syntaxes pour sélectionner les processus et les informations affichées par `ps` : System V, BSD, et Posix en cours d'implémentation (contrôler avec `man`).

Principales options (syntaxe Posix) :

- **`-e`** affiche tous les processus de tous les utilisateurs
- **`-U`** *`liste_d_utilisateurs`* sélectionne les processus appartenant à cette liste d'utilisateurs
- **`-f`** (***f**ull*) affiche une liste complète d'informations sur chaque processus

Exemples

```
$ ps
    PID    TTY   TIME CMD
1212592  pts/2  0:00 ps
1294516  pts/2  0:01 bash
```

```
$ ps -U lefrere
   UID     PID    TTY   TIME CMD
40369  307400      -  0:02 sshd
40369 1212590  pts/2  0:00 ps
40369 1294516  pts/2  0:01 bash
```

N.-B. : la commande ps se voit agir.

Principaux champs affichés :

UID	PID	PPID	TTY	VSZ	CMD
n° d'utilisateur	n° du processus	n° du père	terminal	taille	commande

$ ps -f (***full***)

```
  UID      PID    PPID   C     STIME    TTY   TIME CMD
40369 1294516  307400   0 00:23:53  pts/2   0:01 bash
40369 2027692 1294516  45 00:59:00  pts/2   0:00 ps -f
```

Sous linux, **ps --forest** ⇒ affiche la hiérarchie des processus

```
  UID   PID PPID C STIME TTY          TIME CMD
40369  1935    1 0 03:45 ?        00:00:00 xterm -ls
40369  1936 1935 0 03:45 pts/6    00:00:00  \_ bash
40369  2114 1936 0 03:53 pts/6    00:00:00      \_ xterm -bg yellow
40369  2115 2114 0 03:53 pts/14   00:00:00      |   \_ bash
40369  2166 2115 0 03:55 pts/14   00:00:00      |       \_ xclock
40369  2167 1936 0 03:55 pts/6    00:00:00      \_ xclock
40369  2188 1936 0 03:57 pts/6    00:00:00      \_ ps --forest
```

Comparer le PPID et le PID du père...

16.2 Caractères de contrôle et signaux

Caractères de contrôle (notés **^X** pour `Cntrl X`) interprétés par le shell ⇒ gestion des processus attachés au terminal et des flux d'entrées/sorties.

^L	`clear`	efface l'écran
^S	`stop`	blocage de l'affichage à l'écran
^Q	`start`	déblocage de l'affichage à l'écran
^? ou **^H**	`erase`	effacement du dernier caractère
^D	`eof`	fermeture du flux d'entrée
^C	`int`	interruption du processus
^Z	`susp`	suspension du processus en cours
^	`quit`	interruption forcée du processus avec production d'une image mémoire (fichier `core`)

`stty` gère l'affectation des caractères de contrôle à certaines fonctions

`stty -a` indique leur affectation courante

Un caractère de contrôle ne peut agir que sur le processus en interaction avec le terminal.

16.3 Envoie d'un signal à un processus (`kill`)

Intervenir sur un processus (en cours dans le terminal) au moyen de son `pid`

$\Rightarrow$ envoyer un *signal* spécifique à un processus particulier

`kill` ***`pid`*** où *pid* est le numéro du processus

`kill` envoie par défaut un signal de terminaison

si le processus ne s'interrompt pas, `kill -s KILL` (ou `kill -9`)

`killall` : envoi à une liste de processus désignés par des noms de commandes

exemple : `killall -r '.*mozilla.*'`

16.4 Processus en arrière plan (&, jobs, fg, bg)

Système UNIX multi-tâche :

– commandes longues en *arrière-plan* (*background*)

– « garder la main » pour d'autres commandes pendant cette tâche de fond

syntaxe

```
commande &
```

Gestion des processus en arrière-plan :

– **jobs** affiche la liste des processus en arrière-plan.

– **fg** *%num_job* (***fore****ground*) passe le job de numéro *num_job* en premier plan

– **bg** (***back****ground*) passe le job courant en arrière-plan

Processus en arrière-plan ⇒ plus d'entrées au clavier ⇒ redirections de l'entrée et de la sortie vers des fichiers

mais arrêté par la fermeture du terminal.

Exemple

– `xterm` en premier-plan ⇒ on « perd la main » dans la fenêtre initiale.
  Dans la nouvelle fenêtre, terminer ce processus par `exit` ou `^D`
  ⇒ retrouver la main dans la fenêtre initiale.
– `xterm` **`&`** ⇒ conserve la main dans la fenêtre initiale.
  Depuis la fenêtre initiale, terminer ce processus `xterm`
  par `kill` *`pid`* ou par `fg` puis `^C`

16.5 Groupement de commandes (; et ( ))

Plusieurs commandes sur une même ligne ⇒ les séparer par un point-virgule **;**

cd ; pwd ; ls

Grouper plusieurs commandes grâce aux **parenthèses**

⇒ s'exécutent dans un sous-shell, fils du shell interactif.

commande	réponse	remarque
cd /tmp **(cd /bin; pwd)** **pwd**	/bin /tmp	le changement de répertoire est temporaire

Groupement et redirections :

date ; hostname > memo1 contient le nom du calculateur, la date est affichée

(date ; hostname) > memo2 contient la date et le nom du calculateur

16.6 Processus détaché (nohup)

Processus *détaché* = tâche sans interaction avec un terminal,

⇒ continue après la fin de session interactive.

⇒ rediriger tous les flux standards

nohup *commande* <entrees >sortie 2>erreurs &

Autres modes d'exécution des processus détachés et *différés* :

– soumission dans une file d'attente ou `batch` (via `qsub` par exemple)

  ⇒ le système gère les priorités d'exécution

– à un instant déterminé par exemple via le service `cron` :

  at *date commande*

– processus *périodiques* pour la gestion du système

  (voir la commande `crontab`).

17 Variables du shell

Variables de l'interpréteur de commandes :

– non typées (chaînes de caractères)

– non déclarées

17.1 Affectation et référence

– Syntaxe d'affectation (en shell de type BOURNE) :

syntaxe

variable*=*valeur (sans espaces autour du signe =)

– Référence à la valeur de la variable :

syntaxe

$*variable* ou, plus précisément

${*variable*}

La commande interne **set** (sans argument) affiche la liste des variables et leurs valeurs.

Exemples

```
alpha=toto ; b=35 ; c2=3b

echo alpha, b, c2 contiennent ${alpha}, ${b}, ${c2}
    ⇒ alpha, b, c2 contiennent toto, 35, 3b

set | grep alpha
    ⇒ alpha=toto
```

17.2 Portée des variables du shell

17.2.1 Portée des variables ordinaires

Pas d'héritage des variables ordinaires par les processus fils.

```
a=bonjour                                   affectation dans le processus père
echo a contient +${a}+
⇒ a contient +bonjour+
bash ____________________________________   lancement d'un shell fils
    echo a contient +${a}+
    ⇒ a contient ++                                  pas affecté
    a=salut                                 affectation dans le fils
    echo a contient +${a}+
    ⇒ a contient +salut+
    exit ____________________________________   retour au shell père
echo a contient +${a}+
⇒ a contient +bonjour+                      valeur avant appel du shell fils
```

17.2.2 Extension de la portée (`export`)

Exportation d'une variable vers les processus fils :

syntaxe

```
export variable
```

Mais pas d'héritage inverse, du processus fils vers le père.

17.2.3 Variable ordinaire et variable d'environnement

Une fois exportée, une variable ordinaire devient une variable d'environnement.

Les variables d'environnement sont systématiquement **héritées par les processus fils**.

```
a=bonjour                                   affectation dans le processus père
echo a contient +${a}+
⇒ a contient +bonjour+
export a                                                    exportation
bash ──────────────────────────────────────  lancement d'un shell fils
    echo a contient +${a}+
    ⇒ a contient +bonjour+                                  affecté
    a=salut                                 affectation dans le fils
    export a                                                exportation
    echo a contient +${a}+
    ⇒ a contient +salut+
    exit ──────────────────────────────────  retour au shell père
echo a contient +${a}+
⇒ a contient +bonjour+                      valeur avant appel du shell fils
```

17.3 Variables d'environnement standard

Il existe des variables d'environnement standard vitales au shell :

- **SHELL** : interpréteur de commandes utilisé (`bash`, `ksh`, `tcsh`, ...)
- **TERM** : type de terminal utilisé (`vt100`, `xterm`, ...)
- **HOME** : répertoire d'accueil
- **PATH** : liste des chemins de recherche des commandes séparés par des **:**
- **USER** : nom de l'utilisateur

En tant que variables d'environnement : elles sont systématiquement **héritées par les processus fils**.

Liste des variables d'environnement et de leur valeur : **env**

Valeur d'une des variables : `echo` *`$VARIABLE`*

17.3.1 La variable d'environnement `PATH`

Quand on lance une commande ou un exécutable :

– avec **/** dans le nom, on précise le chemin d'accès explicitement

*Exemple (**c'est la syntaxe conseillée**) :*

`./a.out` exécute `a.out` qui se trouve dans le répertoire courant

– sans **/** dans le nom, la recherche se fait dans tous les répertoires listés dans `PATH` en respectant l'ordre.

Exemple :

`echo $PATH` montre que le répertoire courant n'est pas scruté :

```
/usr/local/bin:/bin:/usr/bin:/usr/X11R6/bin:/home/lefrere/bin
```

Si on l'ajoute à la fin, il est scruté en dernier :

`PATH="${PATH}:." ; echo ${PATH}` donne

```
/usr/local/bin:/bin:/usr/bin:/usr/X11R6/bin:/home/lefrere/bin:.
```

Ne pas le placer au début pour des raisons de sécurité !

Remarques :

– `PATH=""` $\Rightarrow$ seules les commandes avec chemin sont trouvées

– importance des délimiteurs :

`"  "` : protection faible - le shell interprète le symbole `$`

`'  '` : protection forte - le shell n'interprète aucun symbole

17.4 Code de retour d'une commande ($?)

Toute commande UNIX retourne en fin d'exécution un code entier appelé valeur de retour (cf. celle de `main` en C) ou statut de fin (*return status*) **`$?`**

Code de sortie = `0` $\Longleftrightarrow$ la commande s'est bien déroulée.

Exemples :

`cd /bin`

`echo $?` affiche **`0`**

`cd /introuvable` affiche un message d'erreur

`echo $?` affiche **`1`**