

UPMC

Master SPE

Table des matières

Méthodes mathématiques et modélisation de l'environnement

Introduction à l'environnement Unix

2020–2021

Jacques.Lefrere@upmc.fr

1 Introduction au système UNIX	6
1.1 Système d'exploitation	6
1.2 Historique et développement d'unix et linux	7
1.3 Principales caractéristiques du système UNIX	8
1.4 Compte utilisateur	9
1.5 Sessions unix	10
1.6 Exemples de commandes élémentaires d'affichage	10
2 Le shell : introduction	12
2.1 Syntaxe de la ligne de commandes	12
2.2 Aides à l'interactivité du shell	13

UPMC M1 SPE

1

2020-2021

TABLE DES MATIÈRES

Unix

TABLE DES MATIÈRES

TABLE DES MATIÈRES

Unix

TABLE DES MATIÈRES

2.3 Générateurs de noms de fichiers	14
2.4 Documentation en ligne	16
3 Hiérarchie des fichiers unix	17
3.1 Arborescence	17
3.2 Chemins d'accès (<i>path</i>) d'un fichier	20
3.2.1 Affichage du répertoire courant avec <code>pwd</code>	21
3.2.2 Changement de répertoire courant avec <code>cd</code>	21
3.3 Raccourcis pour les répertoires d'accueil	26
4 Commandes de base	27
4.1 Commandes de gestion de fichiers	27
4.1.1 Affichage de liste de noms de fichiers avec <code>ls</code>	27
4.1.2 Copie de fichiers avec <code>cp</code>	29

4.1.3 Déplacement et renommage de fichiers avec <code>mv</code>	30
4.1.4 Suppression de fichiers avec <code>rm</code>	31
4.1.5 Compression de fichiers avec <code>gzip</code> ou <code>bzip2</code>	32
4.2 Commandes de gestion de répertoires	33
4.2.1 Création de répertoire avec <code>mkdir</code>	33
4.2.2 Suppression de répertoire (vide) avec <code>rmdir</code>	33

5 Commandes pour fichiers textes	34
5.1 Fichiers binaires et fichiers texte, codage	34
5.2 Codage des fichiers textes	34
5.2.1 Transcodage de fichiers textes avec <code>recode</code> ou <code>iconv</code>	35
5.3 Accès au contenu des fichiers	37
5.3.1 Identification des fichiers avec <code>file</code>	37

- 5.3.2 Comptage des mots d'un fichier texte avec `wc` 38
- 5.3.3 Affichage du contenu de fichiers texte avec `cat` 39
- 5.3.4 Affichage paginé du contenu d'un fichier texte avec `more/less` 40
- 5.3.5 Début et fin d'un fichier texte avec `head` et `tail` 41
- 6 Environnement réseau, transfert et archivage 42**
- 6.1 Connexion à distance via `slogin` 42
- 6.2 Transfert de fichiers à distance : `scp`, `sftp` 43
- 6.3 Explorateurs et téléchargement 45
- 6.4 Archivage d'arborescence avec `tar` 46
- 7 Redirections et tubes 51**
- 7.1 Flux standard 51
- 7.2 Redirections 52

- 7.2.1 Redirection de sortie vers un fichier (> et >>) 53
- 7.2.2 Redirection de l'entrée depuis un fichier (<) 55
- 7.3 Tubes ou *pipes* (|) 56

8 Processus en arrière plan 60

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 S.E. (en anglais *operating system* O.S.)
- exemples : MS-DOS, Windows (XP, 7, ...,10), famille **Unix** (**linux**, Mac-OS, Android sur les mobiles, ...)

Linux aujourd'hui dominant dans le calcul intensif : plus de 97% des calculateurs du TOP 500

N.-B. : **machine virtuelle** = application qui émule un système d'exploitation
 ⇒ ex. : linux émulé sous **virtualbox** ou **VMware** dans une fenêtre windows

1.2 Historique et développement d'unix et linux

- depuis les années 1970, plusieurs branches de développement
 ⇒ quelques différences dans les commandes surtout au niveau administration
- système ouvert : implémentations du téléphone portable (noyau d'Android) au super-calculateur
 - propriétaires (aix, hp-ux, solaris, os-X, ...)
 - libres (**linux** depuis 1991, net-bsd, free-bsd, ...) : linux est (presque) un unix !
- **nombreuses distributions** linux, principales branches :
 - **debian** ↪ ubuntu ↪ mint
 - **slackware** ↪ Suse ↪ **OpenSuse** (LUTES)
 - **Red-Hat** ↪ Mandrake ↪ mandriva ↪ mageia,
 - ↪ **CentOS** (serveur appli1)
 - ↪ scientific-linux,
 - ↪ Fedora, ...

1.3 Principales caractéristiques du système UNIX

- multi-tâches (concurrentes et indépendantes)
- multi-utilisateurs (dont l'administrateur ou *root*)
 - ⇒ système d'**identification** et **droits** d'accès aux fichiers
- **chaînage des processus** par les **tubes** (pipes)
 - ⇒ assemblage d'outils élémentaires pour accomplir des tâches complexes
- Le **shell** est l'interface utilisateur du système d'exploitation.

bash : *Bourne again shell* (sh : shell historique de Bourne)

l' **interpréteur de commandes** (**shell**) intègre un **langage de programmation** avec variables, structures de contrôle, fonctions ...

⇒ programmes interprétés en shell = fichiers de commandes = **shell-scripts**

⇒ création de commandes par l'utilisateur

1.4 Compte utilisateur

- un **identifiant** (ou *login*) (unique)
- un **mot de passe** (ou *password*) confidentiel
- un **groupe** parmi ceux définis sur la machine
- un **répertoire d'accueil** personnel (ou *home directory*) où stocker ses fichiers
- un **« interpréteur de commandes »** (ou *shell*) : **bash**

Ces informations sont stockées dans un fichier système (souvent */etc/passwd*)

Le mot de passe est crypté

⇒ l'administrateur ne peut pas retrouver un mot de passe oublié

Ressources limitées, par exemple par quota sur le disque

⇒ problème de connexion en mode graphique si quota atteint.

1.5 Sessions unix

- point commun : une session commence par

- **identification** (*login*)
- **authentification** (*password*)

la même invite apparaît après la fin de session

- deux types de **sessions** de travail :

- mode **texte** (console, accès distant (*slogin*), ...): **ligne de commande**
avantage : conservation de l'historique des commandes
- mode **graphique** (multi-fenêtres) : icônes et menus pour lancer les applications (dont les consoles **konsole**, *xfce4-terminal* et *xterm* par ex.)
environnements de bureau : **kde**, *gnome*, **xfce**, *mate*, **lxde**...
gestionnaires de fenêtres : *fvwm*, *icewm*...

1.6 Exemples de commandes élémentaires d'affichage

la commande	affiche
date	la date
whoami	le login
hostname	le nom de la machine
who	la liste des utilisateurs connectés
echo "chaîne de caractères"	la chaîne saisie
id	le numéro d'utilisateur
uname	le nom du système d'exploitation

2 Le shell : introduction

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

2.1 Syntaxe de la ligne de commandes

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

- (1) premier mot = **la commande** l'action
- (2) mots suivants = **les paramètres** ou arguments les objets
(rôle déterminé par leur position dans la ligne de commande)
- (3) paramètres **optionnels** introduits par « - » les modalités

cp	-p	fich1	fich2
(1) commande	(3) option	(2) paramètre 1	(2) paramètre 2
copie	en gardant la date	source	cible

- Le shell distingue les **majuscules** (rares) des **minuscules**
- il interprète certains **caractères** dits **spéciaux** : blancs, " , ' , \ , * , ? , () , ...

avant de d'exécuter la commande

⚠ ⇒ éviter les blancs dans les noms de fichiers

2.2 Aides à l'interactivité du shell

- ↑ et ↓ permettent de parcourir l'historique des commandes
- ← et → déplacements pour éditer la ligne de commande
- Ctrl E** = **^E** ou **^A** déplacement en fin (*End*) ou début de ligne (**^B** est pris par *Back*)
- TAB** demande au système de **compléter le nom** de commande ou de fichier
⇒ évite les fautes de saisie et **valide les chemins**
- TAB TAB** affiche les différentes possibilités de complétion
plus beaucoup d'autres (voir chapitres suivants)

2.3 Générateurs de noms de fichiers

Caractères *jokers* interprétés par le shell pour désigner des fichiers selon des **motifs génériques**

- *** une chaîne de caractères quelconque dans le nom d'un fichier
(y compris la chaîne vide)
- ?** un caractère quelconque et un seul dans un nom de fichier
- [...]** un caractère quelconque pris dans la liste exhaustive entre crochets
- [c₁-c₂]** un caractère quelconque entre c₁ et c₂ dans l'**ordre lexicographique**

Exemples de motifs de noms de fichiers

- *** tous les fichiers du répertoire courant (sauf ceux commençant par .)
 - *.py** tous les fichiers dont le nom finit par **.py** (scripts *python*)
 - *.*** tous les fichiers dont le nom comporte un point (au moins)
 - 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
NB. : **f.[25-70]** (maladroit, mais) les fichiers **f.0**, **f.2**, **f.5**, **f.6** et **f.7**
 - *.[ch]** les fichiers source en C (***.c**) et les fichiers d'entête (*header* : ***.h**)
- Tester ces motifs avec par exemple la commande **echo** : `echo *.[ch]`

2.4 Documentation en ligne

— **man cmd** : affichage du manuel de la commande `cmd`

page par page grâce au filtre `more` ou `less`

- se déplacer dans le manuel : ↑ ↓, page suivante/précédente
- rechercher un motif : `/motif`, l'occurrence suivante **n** (*next*)
- **sortir** du manuel : touche q *quit*

Préciser parfois la section du manuel (1 = commandes, 3 = bibliothèques)

man 3 printf (⇒ la fonction C)

au lieu de `man printf` (⇒ section 1 commande)

— **cmd --help** : affiche un bref aide-mémoire de la commande

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

3 Hiérarchie des fichiers unix

3.1 Arborescence

L'ensemble des fichiers est structuré hiérarchiquement en un **arbre unique** constitué de

— nœuds : **répertoires** (*directories*, dossiers (*folders*) sous windows), les répertoires contiennent d'autres fichiers

— feuilles : **fichiers** (*files*) ordinaires en général.

★ le séparateur de niveaux est la barre oblique `/` (*slash*)

★ le répertoire `/` est la **racine** (*root*), qui contient tous les autres fichiers.

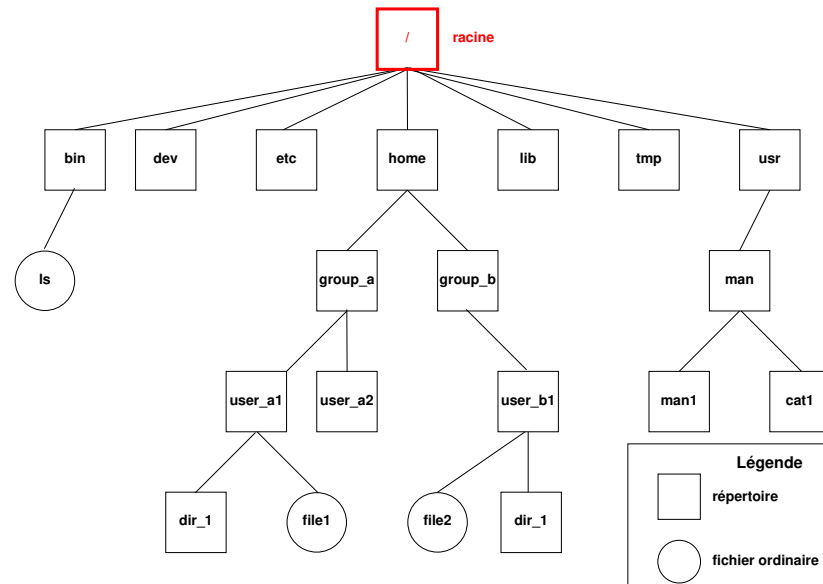


FIGURE 1 – Arborescence des fichiers UNIX

Montage de branches

L'arbre unique d'unix est **purement logique**.

Plusieurs périphériques peuvent être « montés » sur le système de fichiers → greffe temporaire d'une branche sur un « point de montage » de cet arbre.

Par exemple, un disque amovible ou une clef USB peuvent être « montés » par exemple dans le répertoire `/media/removable`.

Après utilisation, il faut « démonter » ces branches (pour achever les transferts de données) avant de déconnecter physiquement ces périphériques.

Partage via le réseau : divers protocoles (`ssh` via `sshfs`, `nfs`) réalisent des montages pour partager des répertoires hébergés par d'autres ordinateurs (serveurs) via le réseau.

Sous windows séparateur = contre-oblique `\` (*antislash*)

périphériques désignés par une lettre suivie de « : » ex. `C:\` ou `D:\`

⚠ pas de distinction minuscule/majuscule ⇒ problème si montage linux

3.2 Chemins d'accès (*path*) d'un fichier

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

Exemples : /usr/man/man1/ls.1, /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 ou de travail (*working directory*)
- .. répertoire père (*parent directory*)

⚠ Des fichiers de même nom ne peuvent exister que dans des répertoires différents

NB : `tree rep` permet de représenter la branche qui part du répertoire `rep`

3.2.1 Affichage du répertoire courant avec `pwd`

`pwd` (*print working directory*) affiche le chemin absolu du répertoire courant commande interne (*builtin*) du shell

3.2.2 Changement de répertoire courant avec `cd`

`cd` [*répertoire*] (*change directory*)

commande interne (*builtin*) du shell

- `cd` (sans paramètre) retour au répertoire d'accueil ~/.
- `cd -` retour au précédent répertoire (dans le temps)
- `cd ..` retour au répertoire père (dans la hiérarchie)

Exemples en supposant que `pwd` affiche `/home/group_a/user_a1`

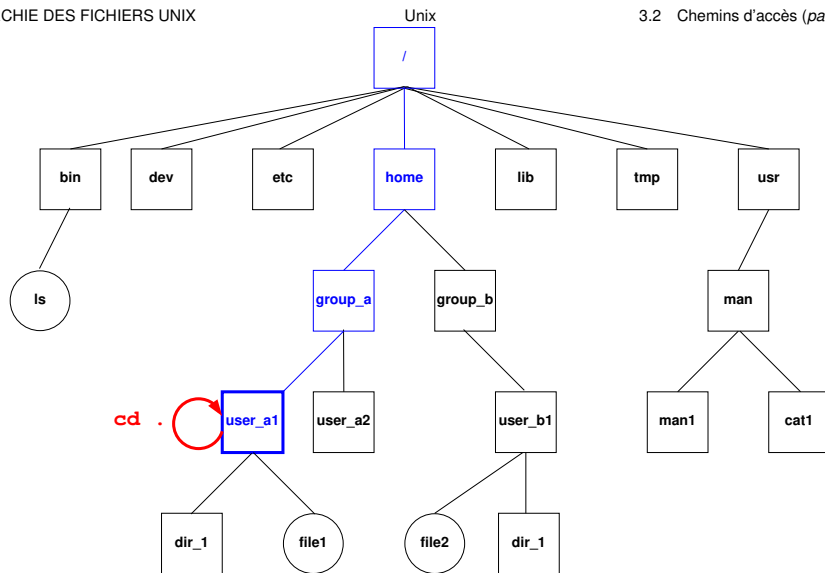


FIGURE 2 – La commande `cd .` laisse dans le répertoire courant /home/group_a/user_a1.

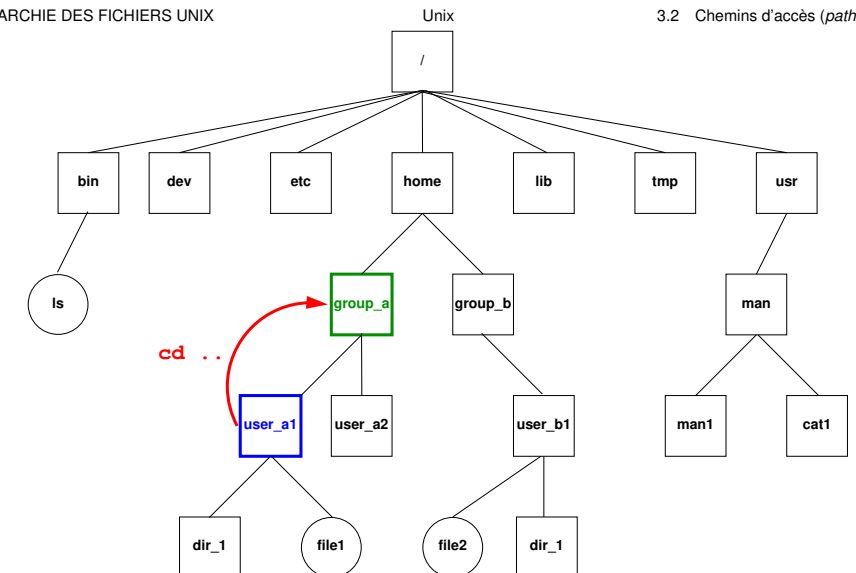
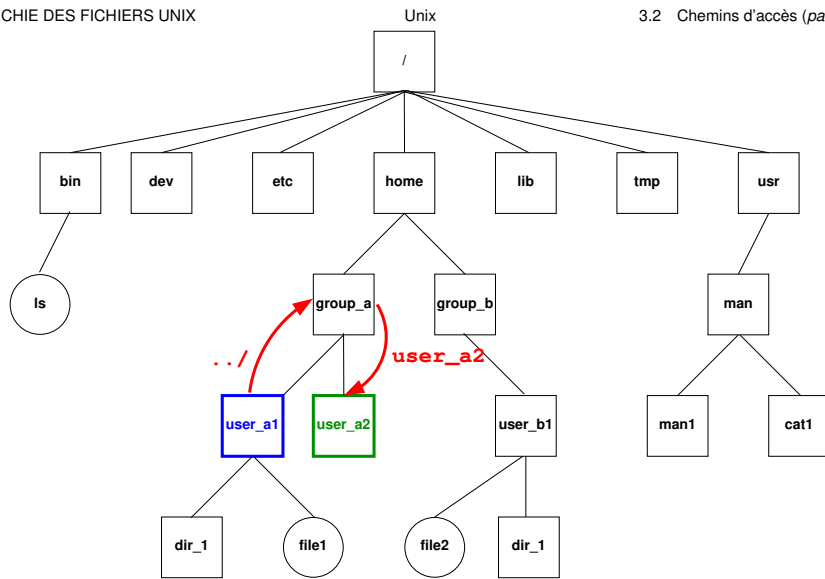
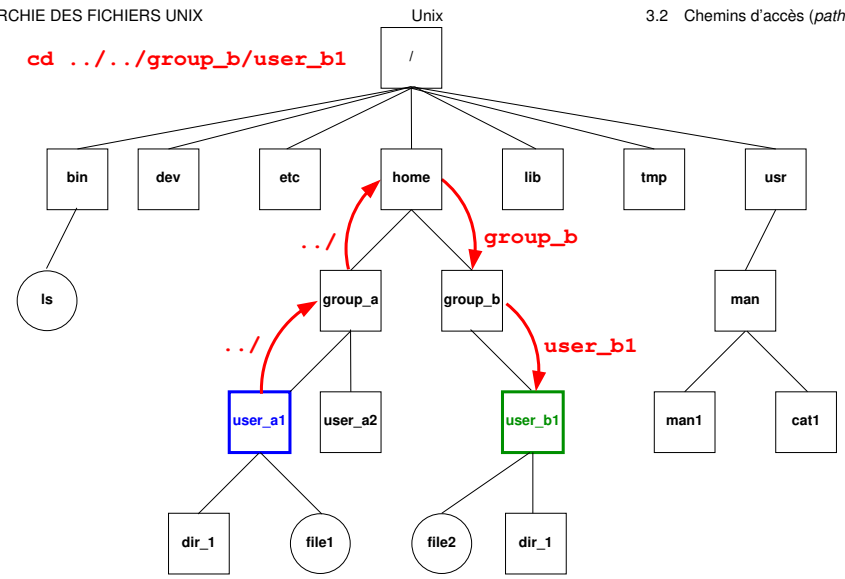


FIGURE 3 – La commande `cd ..` déplace dans le répertoire père group_a.

FIGURE 4 – `cd ../user_a2` déplace dans le répertoire `user_a2`FIGURE 5 – `cd ../../group_b/user_b1` déplace dans le répertoire `user_b1`.

3.3 Raccourcis pour les répertoires d'accueil

Chemins en fait absolus :

`~user` répertoire d'accueil de l'utilisateur nommé `user`

`~` son propre répertoire d'accueil

Exemples :

`~/bash_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`.

⚠ Ne pas confondre `~lefrere` et `~/lefrere`

4 Commandes de base

⚠ Les arguments fichier des commandes pourront comporter un chemin d'accès.

4.1 Commandes de gestion de fichiers

4.1.1 Affichage de liste de noms de fichiers avec `ls` (*list*)

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

Quelques options :

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

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

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

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

Exemples

ls (sans argument) liste (courte) des fichiers du répertoire courant ⇔ **ls .**
ls -l liste des fichiers du répertoire courant avec attributs
ls -al idem avec aussi les fichiers cachés (commençant par `.`)

⚠ cas des répertoires : par défaut le **contenu**

ls -l rep liste détaillée des fichiers du répertoire `rep` (**contenu**)
ls -dl rep nom et attributs du **répertoire** `rep` (**contenant**)
ls -l fic1 fic2 rep liste détaillée des fichiers ordinaires `fic1`, `fic2`
 et du **contenu du répertoire** `rep`
ls -R /tmp liste des fichiers de toute la branche `/tmp`

4.1.2 Copie de fichiers avec **cp** (*copy*)

— copie avec changement de nom et de chemin éventuels

deux arguments seulement

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

— copie d'un ou plusieurs fichiers sans changement de nom
 vers **un même répertoire**

cp [*-options*] **liste_de_fichiers repertoire_cible**

`cp *.c bck/` copie les fichiers source C dans le répertoire `bck`

Principales options :

-r (*recursive*) copie d'une branche (si le premier argument est un répertoire)

-p (*permissions*) sans changer les droits ni la date

-i (*interactive*) demande de confirmation si la cible existe déjà

Confirmation en cas d'écrasement : répondre **y** (o si francisé)

4.1.3 Déplacement et renommage de fichiers avec **mv** (*move*)

Trois syntaxes possibles

1. **mv fichier_origine fichier_cible**

2 arguments seulement renommage sauf si chemins d'accès différents

`mv fic1 fic2` renomme `fic1` en `fic2`

`mv /tmp/fic1 fic2` idem mais prend `fic1` dans `/tmp`

2. **mv liste_de_fichiers repertoire_cible**

déplacement des fichiers de la liste vers le répertoire cible (qui doit exister)

`mv fic1 fic2 /tmp/` déplace `fic1` et `fic2` dans `/tmp`

3. **mv repertoire_source repertoire_cible**

renommage ou déplacement de branche

Principale option :

-i (*interactive*) demande de confirmation interactive si écrasement de fichier

4.1.4 Suppression de fichiers avec **rm**

en anglais *remove*

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

Principales options :

-i (*interactive*) demande de confirmation interactive

-r ou **-R** (*recursive*) destruction d'une branche (puissant mais... dangereux)

-f (*force*) sans demande de confirmation ni erreur si fichier inexistant

⚠ Attention : pas toujours de confirmation (sauf alias en **rm -i**)
 destruction immédiate irréversible (pas de corbeille)

4.1.5 Compression de fichiers avec `gzip` ou `bzip2`

Compression et décompression sans perte d'information

— Compression → fichier de suffixe `.gz`

```
gzip [-options] liste_de_fichiers
```

— Décompression d'un fichier de suffixe `.gz`

```
gunzip [-options] liste_de_fichiers
```

⚠ Ne pas confondre avec `zip` de windows : `gzip` n'archive pas... ⇒ voir `tar`.

Autre outil, plus efficace : `bzip2` / `bunzip2` (suffixe `.bz2`)

4.2 Commandes de gestion de répertoires

4.2.1 Création de répertoire avec `mkdir`

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

4.2.2 Suppression de répertoire (vide) avec `rmdir`

```
rmdir répertoire (remove directory)
```

refus de suppression si le répertoire contient des fichiers

⇒ utiliser `rm -R répertoire`, mais dangereux !

5 Commandes traitant le contenu des fichiers texte

5.1 Fichiers binaires et fichiers texte, codage

Un **fichier** (ordinaire) = lot d'informations, conservé dans une mémoire permanente (disque, CD, clef USB, ...) et auquel on donne un nom.

Deux aspects du fichier :

bas niveau : suite de **bits** groupés en octets

haut niveau : représentation de texte, d'image, de code machine,...

selon un certain **codage** qui permet d'interpréter la suite de bits.

Préférer des suffixes rappelant le type de codage utilisé :

— fichiers **texte**

`.c` source C, `.py` source python, `.txt` texte, `.html` hypertexte, ...

— fichiers **binaires**

`.pdf` pour du PDF, `.jpg` pour une image JPEG

`.o` objet binaire, `.a` bibliothèque `.pyc` python précompilé, ...

5.2 Codage des fichiers textes

Plusieurs codages pour les caractères :

— **ASCII** sur 7 bits ($2^7 = 128$ caractères) => non accentués

— codages sur 1 octet = 8 bits ($2^8 = 256$ caractères) avec caractères accentués :

– propriétaires : CP852, CP1252 sous windows, MacRoman sous MacOS

– **ISO-8859** avec les variantes locales

ISO-8859-1 ou **latin1** pour le français par exemple

— évolution en cours vers standard **unicode** pour représenter toutes les langues :

nécessiterait jusqu'à 4 octets par caractère : UTF-32 !

implémentation **UTF-8** : **taille variable des caractères : de 1 à 4 octets**

– sur-ensemble de l'ASCII (donc sur 1 octet pour les non-accentués)

– les caractères non-ascii de latin1 sur 2 octets

⚠ les codes binaires (sur 1 octet) des caractères accentués de latin1 sont **invalides** en UTF-8 !

5.2.1 Transcodage de fichiers textes avec `recode` ou `iconv`

— `iconv -f code_initial -t code_final fichier`

⚠ la conversion s'arrête à la première combinaison invalide

— `recode code_initial..code_final fichier`

⚠ par défaut `recode` travaille « en place » (modifie le fichier initial).

Exemples de transcodage de latin 1 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
```

De nombreux éditeurs (`vim`, `emacs`...) peuvent faire de la conversion au vol pour la phase d'édition, puis sauvegarder dans le codage initial.

⚠ Ne pas mélanger deux codages dans un fichier (via par ex. copier/coller)

5.3 Accès au contenu des fichiers

5.3.1 Identification des fichiers avec `file`

`file liste_de_fichiers`

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

⇒ l'utiliser pour savoir avec quelles commandes manipuler un fichier

```
a.out: ELF 64-bit LSB executable, x86-64
carre.py: ASCII text
carre+invite.c: symbolic link to `carre+invite-utf.c'
carre+invite-iso.c: ISO-8859 C program text
carre+invite-utf.c: UTF-8 Unicode C program text
ligne.txt: ISO-8859 text
ligne.utf: UTF-8 Unicode text
poly-unix.tex: LaTeX 2e document text
tel-arbre.pdf: PDF document, version 1.5
```

5.3.2 Comptage des mots d'un fichier texte avec `wc`

```
wc [-cmwl] [liste_de_fichiers] (wordscout)
```

`wc` (*word count*) affiche par défaut le nombre de lignes, de mots et d'octets, sauf si options cumulables pour sélectionner :

— `-l` compte les lignes (*lines*)

— `-w` compte les mots (*words*)

— `-m` compte les caractères (*multibytes*, `c` est pris !) : utile en UTF-8 seulement

— `-c` compte les octets (*characters* au sens historique!) comme `ls -l`

⚠ ordre d'affichage fixe : `l`, `w`, `m`, `c`, du plus gros ensemble au plus petit

5.3.3 Affichage du contenu de fichiers texte avec `cat`

`cat [liste_de_fichiers]`

affiche (**concatène**) le contenu des fichiers de la liste

⚠ N.B. : pas de contrôle du défilement (voir `more` ou `less`)

ex: `cat fic1 fic2 fic3` concatène et affiche le contenu des trois fichiers

`cat` = filtre identité : recopie l'entrée standard (clavier) sur la sortie standard (écran)

`cat -n` affiche les lignes avec leur numéro en tête, suivi d'une tabulation

⚠ Ne pas confondre `cat fichier` avec `echo chaine`

5.3.4 Affichage paginé du contenu d'un fichier texte avec `more/less`

`more` *liste_de_fichiers*

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

`less` *liste_de_fichiers*

préférable sous linux (défilement arrière possible)

Requêtes sous le pagineur

Entrée avance d'une ligne

Espace avance d'un écran

`/motif` recherche la prochaine occurrence de *motif* en avançant

`?motif` recherche la prochaine occurrence de *motif* en reculant

`q` quitte l'affichage

⚠ (nécessaire avec `less` car on peut remonter)

Rappel : `less` = pagineur utilisé par la commande `man`

5.3.5 Début et fin d'un fichier texte avec `head` et `tail`

`head/tail` [*options*] [*liste_de_fichiers*]

`head -n nb fichier` affiche les nb premières lignes de fichier

`tail -n nb fichier` affiche les nb dernières lignes de fichier

`tail -n +11 fichier` affiche à partir de la ligne 11

6 Environnement réseau, transfert et archivage

6.1 Connexion à distance via `slogin`

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

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

`slogin user@dist_host.domain`

`slogin etu1@sappli1.datacenter.dsi.upmc.fr`

⚠ ne pas oublier le login, sauf si identique sur la machine locale

Option `-X` pour autoriser les applications graphiques (fenêtres X11) via `ssh`

Lancement de commandes sur la machine distante :

`ssh user@dist_host.domain dist_cmd`

`ssh etu1@sappli1.datacenter.dsi.upmc.fr ls ~lefrere/M1/Doc`

6.2 Transfert de fichiers à distance via `scp` et `sftp`

La commande `scp`

Copie de fichiers personnels entre deux machines, sans ouvrir de session sur la

machine distante, via `scp` (mot de passe à chaque commande ou clef `ssh`)

Syntaxe de `cp`, mais préfixer le chemin d'accès des fichiers distants par

`user@dist_host.domain`:

`scp [user1@]host1:file1 file2 distant vers local`

`scp 1234567@sappli1.datacenter.dsi.upmc.fr:unix/tel.txt tel-l.txt`

copie le fichier `tel.txt` du répertoire `unix` côté serveur dans le répertoire local courant sous le nom `tel-l.txt`

`scp file1 [user2@]host2:file2 local vers distant`

`scp m3e/*.py 1234567@sappli1.datacenter.dsi.upmc.fr:m3e/python`

copie tous les fichiers source python du répertoire local `m3e` vers le répertoire

`m3e/python/` du serveur

L'utilitaire sftp

Session **sftp** (*secure file tranfert protocol*) pour plusieurs transferts

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*
- navigation distante : **cd** *dist_dir*
- navigation locale : **lcd** *loc_dir* (*local change directory*)
- autres requêtes : **ls**, **lls**, **pwd**, **lpwd**, **mkdir**, **lmkdir**, ...

exit ou **quit** pour terminer la session **sftp**.

6.3 Explorateurs et téléchargement

Navigateurs Web (*firefox*, *opera*, *konqueror*, *chrome*...)

Protocoles : **ftp** (*File Transfer Protocol*), **http** (*Hypertext Transport Protocol*), ou **https** (sécurisé par cryptage).

Ressources localisées grâce à une **URL** (*Universal Resource Locator*).

Exemples d'**URL** :

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

http : //wwens.aero.jussieu.fr/lefrere/master/SPE/ page de l'UE

En ligne de commande : **wget** ou **curl** pour télécharger des fichiers

par exemple :

wget "http://www.scipy-lectures.org/_downloads/ScipyLectures.pdf"

6.4 Archivage d'arborescence avec tar

tar *options* **-f** *archive* [*répertoire*]

Une option et une seule spécifiant **l'action** parmi :

- c** (*create*) création de l'archive à partir de l'arborescence (⇒ argument *répertoire*)
- t** (*list*) liste des fichiers archivés (tels qu'ils seront extraits)
- x** (*extract*) extraction des fichiers pour restaurer l'arborescence

Une option obligatoire à argument :

-f *archive* (*file*) précise le nom du fichier d'archive (toujours nécessaire)

Autres options combinables :

- v** (*verbose*) affiche des informations complémentaires
- z** ou **-j** avec dé/compression (*gzip* ou *bzip2*) du fichier *.tar*

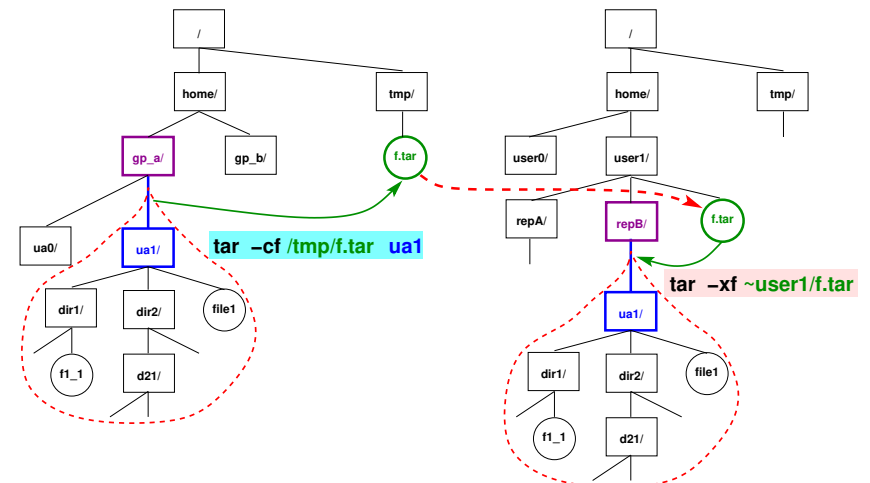


FIGURE 6 – Transfert de branche via **tar** : création de l'archive *f.tar*, transfert de l'archive entre les machines, puis extraction sous *repB*

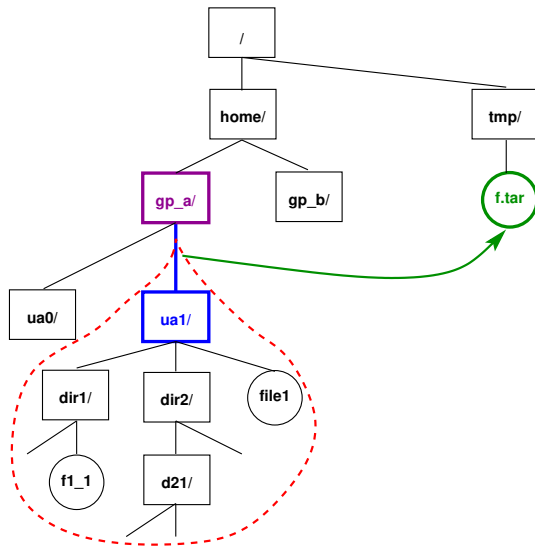


FIGURE 7 – Création

(sous /tmp) de l'archive **f.tar** de la branche de l'utilisateur **ua1** :

1) se placer au dessus de ua1

`cd ~/..`

2) archiver ua1

`tar -cf /tmp/f.tar ua1`

3) vérifier le contenu de l'archive

`tar -tf /tmp/f.tar`

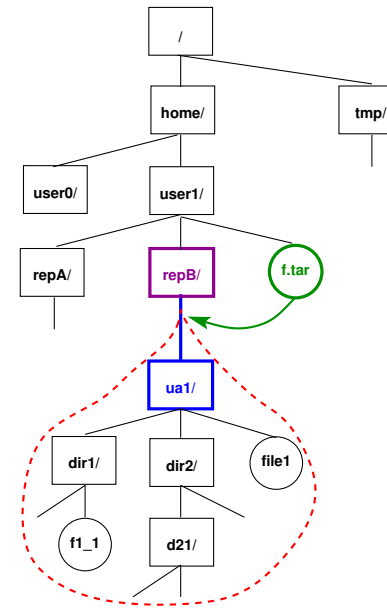


FIGURE 8 – Restauration de

branche à partir de l'archive :

1) `cd ~/repB/`

2) restaurer la branche ua1

juste en dessous de repB

`tar -xf ~/f.tar`

Exemples

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

archive toute l'arborescence de l'utilisateur **user** dans **archive.tar**

(se placer un niveau au-dessus de la branche à archiver)

`tar -tf /tmp/archive.tar`

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

`tar -xvf /tmp/archive.tar`

restaure toujours l'arborescence dans le répertoire **courant** (à partir de l'archive)

(se placer au niveau où « greffer » la branche à restaurer)

NB : l'option **-f** avec argument `-f fichier_archive` est obligatoire.

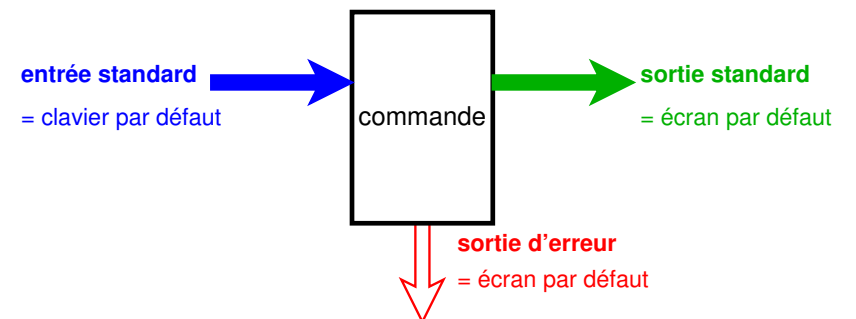


Ne pas désigner la branche à archiver par son chemin absolu, sinon les fichiers seront obligatoirement restaurés au même endroit.

7 Redirections et tubes

7.1 Flux standard

Commande UNIX \Rightarrow trois flux standard de données :



7.2 Redirections

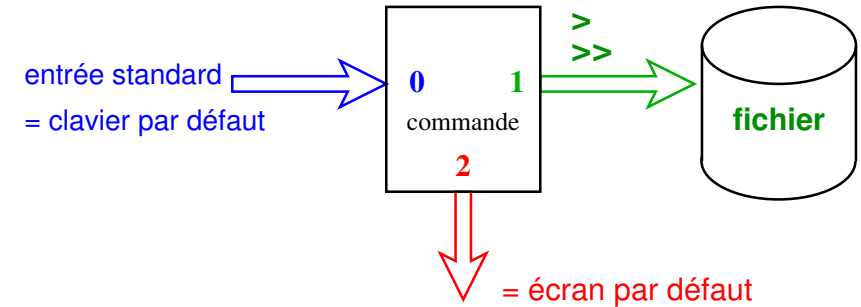
- Au lieu d'une saisie au clavier et d'un affichage à l'écran, stocker de façon permanente l'information d'entrée ou de sortie
 - ⇒ rediriger les flux standards à partir ou vers des **fichiers**
 - Combiner des commandes de base pour effectuer des traitements complexes
 - ⇒ rediriger les flux standards vers les entrées/sorties d'**autres commandes**. (mécanisme des tubes ou pipe-lines)
- ⇒ grande souplesse et puissance du système UNIX

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



Le fichier résultat est créé.

pour ajouter le résultat à la fin du fichier



Exemples

liste détaillée dans un fichier

```
ls -l > liste.txt
```

10 premières puis 10 dernières lignes

```
head fic.txt > deb+fin
```

```
tail fic.txt >> deb+fin
```

les noms des fichiers sources fortran,
puis ceux des fichiers en C

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

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

Attention : le shell interprète très tôt les redirections

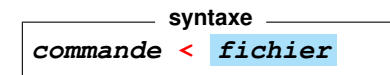
⇒ ne pas rediriger la sortie vers le fichier d'entrée

```
cat -n fic1 > fic1 efface le contenu du fichier fic1
```

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



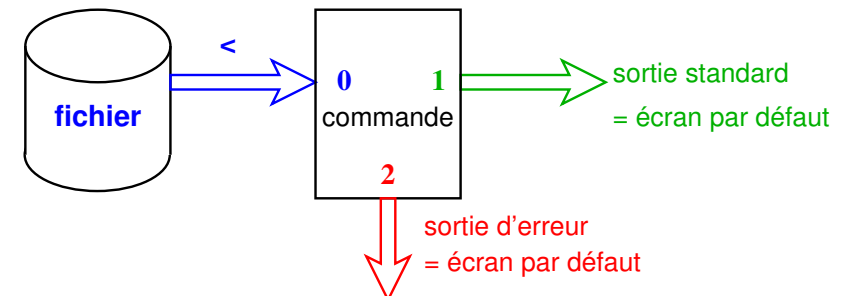
7.2.2 Redirection de l'entrée depuis un fichier (<)



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

le fichier doit exister au préalable.

```
python carre.py < entrees
```



7.3 Tubes ou pipes (|)

Appliquer deux traitements successifs à un flux de données :

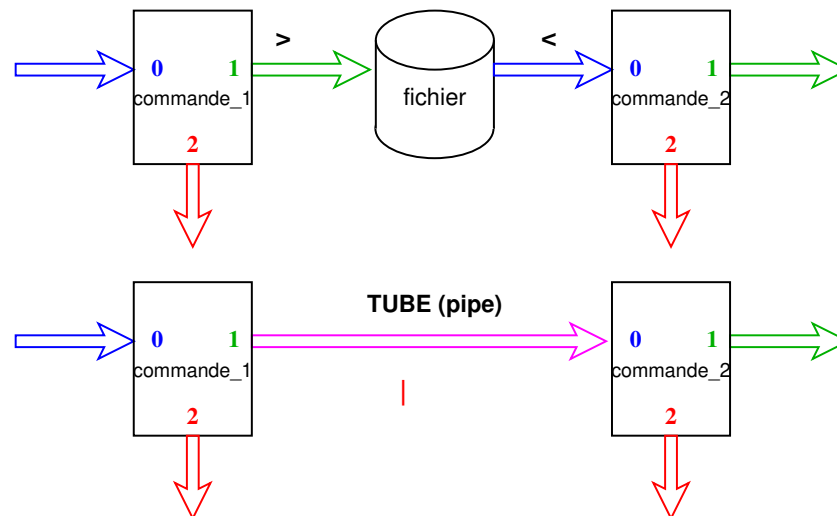
- Méthode **séquentielle** avec fichier intermédiaire :

```
commande_1 > fichier           => attente éventuelle
commande_2 < fichier
rm fichier
```

- **Traitement à la chaîne** en connectant les deux processus par un **tube** ou **pipe** = zone mémoire \Rightarrow communication synchronisée entre les 2 processus

syntaxe
`commande_1 | commande_2`

plus **rapide** que le traitement séquentiel



Exemple 1 : affichage paginé de la liste des fichiers du répertoire courant

Méthode séquentielle (à éviter)

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

Chaînage avec tube (à préférer)

```
ls -l | more
```

Exemple 2 : affichage de la 12^e ligne du fichier toto

Méthode séquentielle (à éviter)

```
head -n 12 toto > tmp1
tail -n 1 tmp1
rm tmp1
```

Chaînage avec tube (à préférer)

```
head -n 12 toto | tail -n 1
```

Cas de plusieurs redirections

L'ordre des redirections sur la ligne est indifférent (avec une seule commande)

commande < entree > sortie

commande > sortie < entree

- ⚠ Avec un tube, **ne pas détourner le flux** : 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 Processus en arrière plan

Système UNIX multi-tâche :

- commandes longues non-interactives en **arrière-plan** (*background*)
- « garder la main » pour d'autres commandes pendant cette tâche de fond (asynchrone)

syntaxe <i>commande</i> &

Gestion des processus en arrière-plan :

- **fg** (*foreground*) passe le job courant en premier plan
 - **bg** (*background*) 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.

Exemples

- **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 **fg** puis **^C**
- si on oublie le **&**, **^Z** pour suspendre le processus,
 puis **bg** pour le passer en arrière-plan