Introduction au Système d'Exploitation Unix/Linux

Vincent Vigneron

Université d'Evry Val d'Essonne
UFR ST

Plan

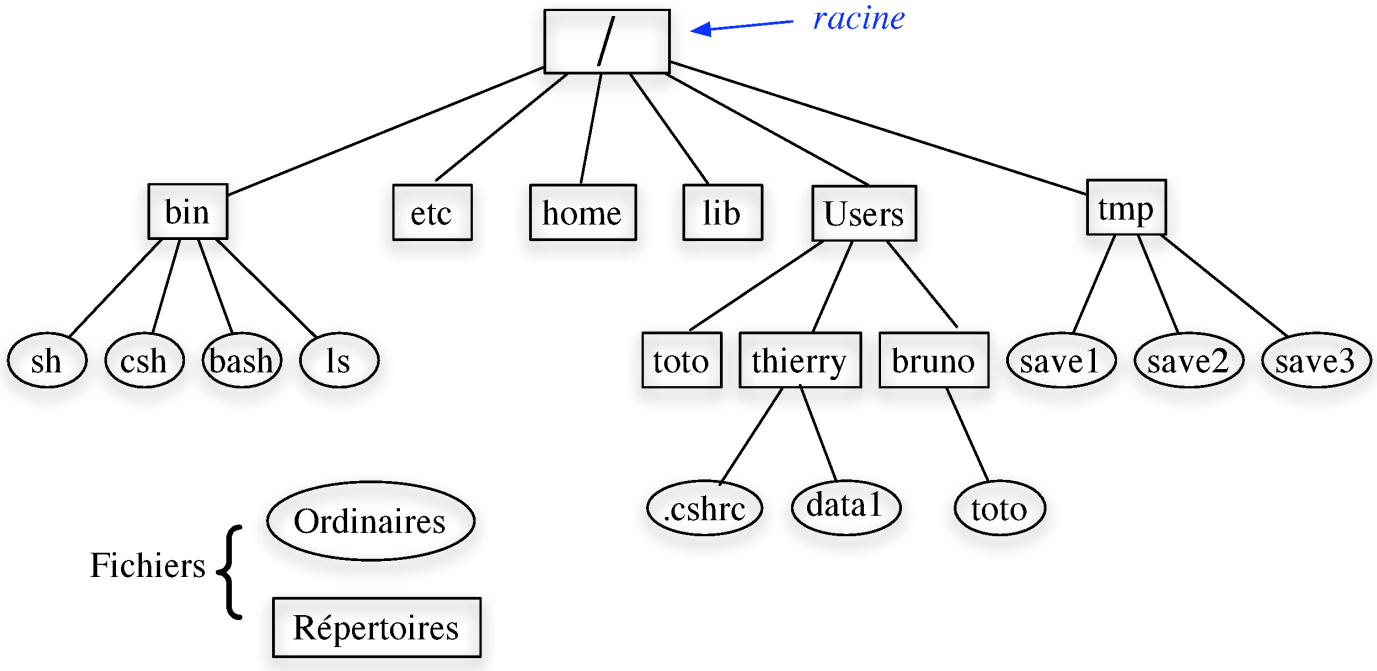
Plan

Gestion des fichiers

- Modèle hiérarchique en "arbre inversé"
  - "arborescence Unix"
  - 1 seule racine notée "/"
    $\neq$ Windows plusieurs racines (`C : D : A : ...`)
- 2 principaux types de fichiers
  - les fichiers ordinaires : programmes, données, textes, images
  - Les fichiers répertoires (ou *directory*) $\approx$ noeud de l'arborescence pouvant contenir
    - d'autres noeuds (répertoires)
    - des fichiers ordinaires

Arborescence

Exemple d'arborescence Unix :



Noms de répertoires

Vocabulaire relatif aux répertoires :

- répertoire d'accueil ou *home* :
  $\longrightarrow$ répertoire sur lequel on est positionné à la connexion
- répertoire courant ou *working directory* :
  $\longrightarrow$ répertoire sur lequel on se trouve à tout moment
- répertoire père :
  $\longrightarrow$ répertoire/noeud au dessus du répertoire courant dans l'arborescence Unix

Accès à ces répertoires

- chemin absolu : à partir de la racine (‘‘/’’)
- chemin relatif au répertoire d'accueil (‘‘∼’’)
- chemin relatif au répertoire courant (‘‘.’’)
- chemin relatif au répertoire père (‘‘..’’)

Nom de fichiers

- Suite de caractères ascii sauf le /
- Longueur limitée (14, ou 255 caractères. . .)
- Pas de contraintes, mais des suffixes conventionnels
- Les langages
  - .c, .h C (et include)
  - .java, .class Java
  - .f Fortran
  - .o Binaire objet
  - .a Librairie de binaires objets
  - pdf Portable Document Format
  - tex TeX ou LATEX

Droits d'accès aux fichiers

Il faudrait définir autant de droits d'accès qu'il y a de

- façons d'utiliser un fichier (lecture, écriture, modificiations. . .)
- d'utilisateurs (souvent des centaines)

$\Rightarrow$ impossible, nombre de combinaisons trop grand
Définition "arbitraire"de

- 3 façons d'utilisation (appelés droits)
  - droit de lecture (read)
  - droit d'ecriture (write)
  - droit d'exécution (ou de traverser un répertoire)
- 3 classes d'utilisateurs :
  - le propriétaire du fichier
  - le groupe auquel $\in$ le propriétaire
  - tous les autres

Affichage des droits

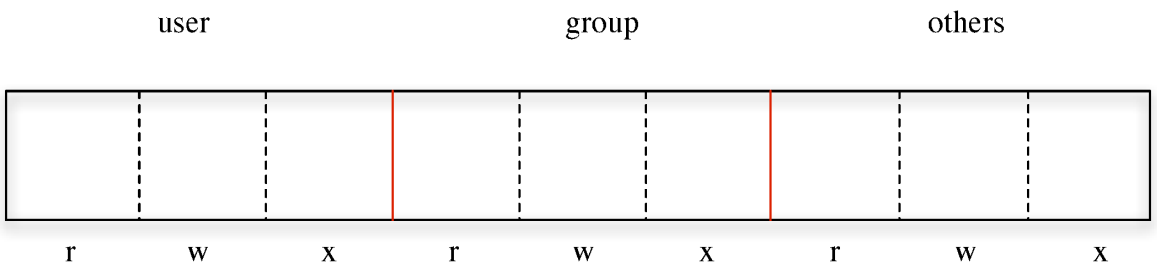
Avec la commande `ls -l`

Affichés sur 10 bits : `-rwxwxrwx`

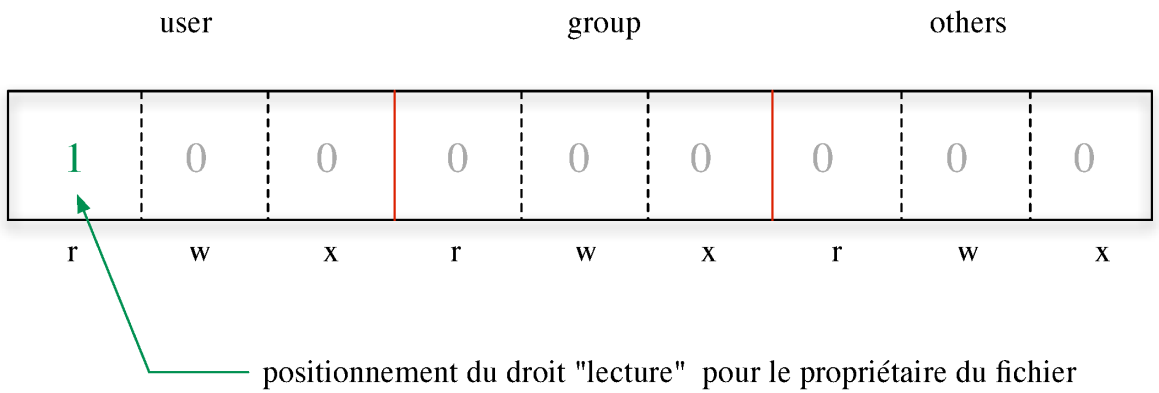
- 1 : Type du fichier
  - `-` : ordinaire
  - `d` : répertoire (directory)
- 2 à 10 : Droits d'accès / d'utilisation
  - 3 droits
    - `r` (Read)
    - `w` (Write)
    - `x` (eXecution)
  - pour les 3 classes d'utilisateurs :
    - `u` (User)
    - `g` (Group)
    - `o` (Others)

$\Rightarrow 3 \times 3$ combinaisons possibles

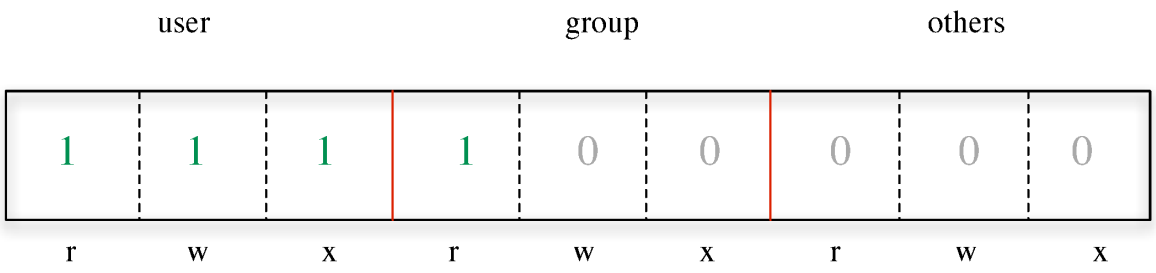
Exemples droits de fichiers



Exemples droits de fichiers



Exemples droits de fichiers



Explications : Fichier que

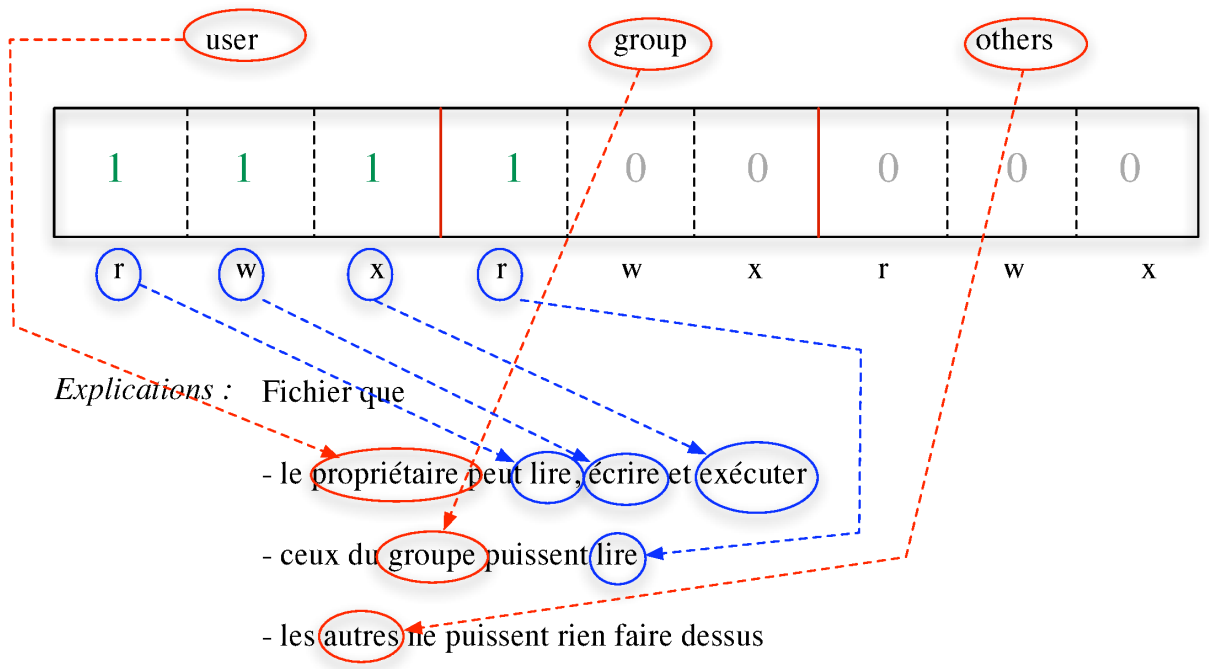
- le propriétaire peut lire, écrire et exécuter

- ceux du groupe puissent lire

- les autres ne puissent rien faire dessus

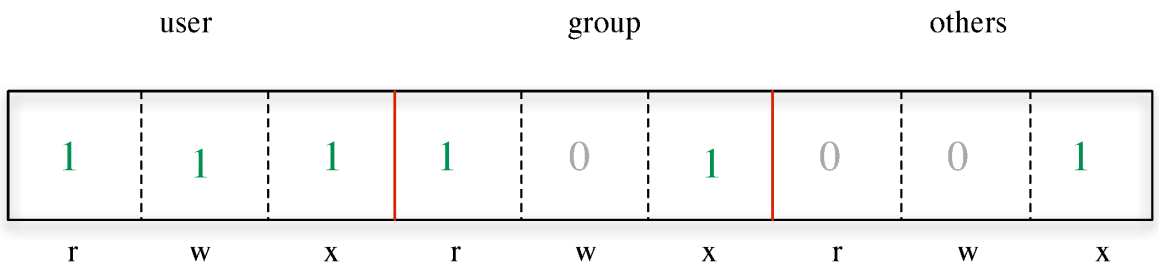
Affichage : `-rwxr-----`

Exemples droits de fichiers



Affichage : `-rwxr-----`

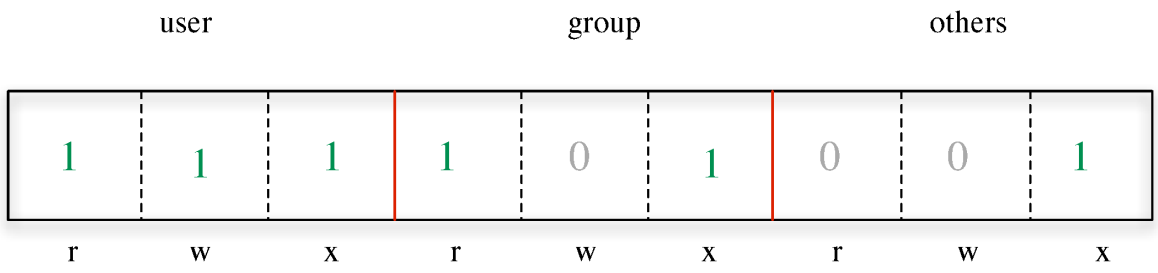
Exemples droits de fichiers



Affichage :

Explications :

Exemples droits de fichiers



Affichage : `-rwxr-x--x`

Explications :

Changement mode d'un fichier

```
chmod [-R] [ugoa] [+-=] [rwx st ugo] filename...
chmod [-R] octalmode filename...
```

Classe

- a appliqué à tous (défaut)
- u appliqué au propriétaire (user)
- g appliqué au groupe
- o appliqué aux autres (others)

Opérations

- + ajout de droits
- - retrait de droits
- = positionnement de droits

Droits

- `r` droit de lecture (read)
- `w` droit d'ecriture (write)
- `x` droit d'exécution (ou de traverser un répertoire)
- `s` "set-uid" bit (associé à u)
  "set-gid" bit (associé à g)
- `t` "sticky" bit

- Ou en octal, le mode est spécifié par combinaison des nombres octaux
  - 400 lisible par le propriétaire
  - 200 modifiable par le propriétaire
  - 100 exécutable par le propriétaire
  - 40 lisible par les membres du groupe propriétaire
  - 20 modifiable le groupe propriétaire
  - 10 exécutable le groupe
  - 4 lisible par les autres utilisateurs
  - 2 modifiable par les autres utilisateurs
  - 1 exécutable par les autres utilisateurs
- Le " sticky " bit limite les swap entre différentes exécutions

Exemple

```
$ ls -l file
-rw--w---- 1 jacob enseign 433 sep 20 15:28 file

$ chmod a+x,g-w+r file
$ ls -l file
-rwxr-x--x 1 jacob enseign 433 sep 20 15:30 file
```

- Fichier exécutable par tous
- Avec les droits du groupe enseign
- Lisible par le groupe
- Modifiable par le propriétaire

Exercice

Exercice sur les droits. . .

Commandes des répertoires

- `pwd` (path working directory)
- `cd` (change directory)
- `mkdir` (make directory)
- `rmdir` (remove directory)
- `ls` (list directory) Donne le contenu d'un répertoire

```
ls [-algiARF...] [name]...
```

Options

- `-a` : all (même commençant par un .)
- `-l` : format long
- `-c` ou `-t` : tri par dernière date de modification
- `-R` : récursif

Exemples

```
homel% ls /bin
X11     grep    roffbib   ...

homel% ls -l
drwxr-xr-x   2 jacob ens    512 mar 26  2003 TclTk
drwxr-xr-x  21 jacob ens   9216 sep 18 17:29 Temporaire
drwxr-xr-x   3 jacob ens    512 jan 10  2002 tst
-rw-r--r--   1 jacob ens    148 jui  7 16:41 uhb.fr

homel% ls -a
.                bin              kadb
..               cdrom            lib
.cshrc           dev              mnt
.login           etc              net
```

Expressions régulières simples

- Elles sont utilisées par les shells pour faire des *modèles* (ou patterns) de noms de fichier.
  Supportées par les commandes traitant plusieurs fichiers (`ls`,...)
- Elle utilisent des métacaractères
  - `?` : "joker" pour un car
  - `*` : n car. ($n \geqslant 0$)
  - `[c,h]` : 'c' ou 'h'
  - `{c,pl}` : 'c' ou 'pl'
  - `[a-e]` : un car. $\in$ [a,b,c,d,e]
  - `[^d-f]` : un car. tous sauf d,e,f
  - `[a-zA-Z]` : une lettre minuscule ou majuscule
  - `[a-z][0-9]` : lettre minuscule suivie d'un chiffre

Exemples

```
prompt% ls
File1    File2    File3
fichier  prog1.c  prog2.c

prompt% ls File?
File1 File2 File3

prompt% ls ?i*
File1 File2 File3 fichier

prompt% ls *.c
prog1.c prog2.c

prompt% ls [^Ff]*
prog1.c prog2.c
```

Commandes de fichiers

- `basename` – le préfixe
- `dirname` – le suffixe
- `file` →type du fichier
- `cat` → affiche le contenu sans pause
- `head` → premières lignes
- `tail` → dernières lignes
- `touch`
  - si le fichier $\exists$: maj date dernière modif
  - si le fichier $\nexists$, créer un fichier vide
- `more, less` → affiche le contenu avec pause

Copie de fichiers

`cp` Copy, 3 utilisations

- Recopie de fichiers
  `cp filename1 filename2` : Recopie du fichier filename1 dans le fichier filename2
- Copie de répertoire
  `cp -rR [-ip] dirname1 dirname2` : Copie récursive de dirname1 dans dirname2
- Copie de fichiers dans un répertoire
  `cp [-iprR] filename... dirname` : Copie des fichiers filename dans dirname
  (dirname doit déjà exister)

Destructions de fichiers

`rm` Remove

```
rm [-] [-fir] filename...
```

Options :

- -r Destruction récursive
- -i Mode interactif
- -f Force

Déplacement de fichiers

`mv` Déplace ou renomme un fichier/un répertoire

- `mv filename1 filename2` : Renomme filename1 en filename2
- `mv dirname1 dirname2` Si dirname2 n'existe pas, renomme dirname1 en dirname2
- `mv filename... dirname` : Place les fichiers filename dans le répertoire dirname

```
home% mv /home/deust/isr1/  /home/deust/isr2
```

Commandes de fichiers et/ou répertoires

du [-s] [-a] [filename]...
Taille disque utilisée par un fichier ou par un répertoire (récursif) en kilo-octets

```
%ls
Cours_1         Cours_2         Cours_Annexes
%du
1334    ./Cours_2
640     ./Cours_1
3970    ./Cours_Annexes
5946    .
```

Archives

- commande tar
- "mise à plat" d'une arborescence de fichiers dans un seul fichier
- convention : suffixe .tar

Compression

- commande zip ou gzip
- compression d'un fichier
- convention : suffixe .zip ou .gz
- souvent utilisé avec une archive $\longrightarrow$ suffixe .tar.gzip

Exercice

- Exercice sur les commandes de fichiers...

Commandes grep et find

- `find` : recherche de fichiers dans une arborescence de répertoires
- `grep` : recherche de lignes dans un ou plusieurs fichiers

Ces commandes utilisent des expressions régulières simples pour faire des modèles (patterns) de noms de fichiers

- ceux à rechercher (find)
- ceux dans lesquels on recherche des lignes (grep)

⊕ `grep` utilise des expressions régulières étendues pour faire des modèles de lignes à rechercher

Expression régulières étendues

Métacaractères des "*regexp*" étendues :

- `.`(dot) : un caractère quelconque
- `*` : opérateur de répétition
- [*xyz*] : x ou y ou z
- [*A* – *G*] : intervale
- [^ `xyz`] : sauf x, y ou z
- ^: début de ligne
- `$` fin de ligne
- `\{m,n\}` : répétitions entre m et n fois
- `\` : échappement (enlève l'interprétation d'un car. spécial)

Exemples

- ^$: représente une ligne vide
- ^[A-Z] : une majuscule en début de ligne
- [^A-Z] : tout sauf une majuscule
- [a-z][a-z]$: 2 minuscules en fin de ligne
- [ABCD]\{2,10\}$: entre 2 et 10 car. A,B,C ou D en fin de ligne

Recherche de fichier

`find dir expr command`

- Recherche dans une arborescence
- Des fichiers satisfaisant expr
- Application de `command`
- `dir` racine de l'arborescence
- `expr`
  - ( ) -o -a opérateurs logiques
  - -name reg-expr-shell
  - -user name
  - -size n / -size +/-n (taille n $\times$ 512 octets)
  - -atime n
  - Autre possibilitées : `man find`

- `command`
  - -print affiche le nom du fichier
  - -exec unix-cmd (avec {}=nom du fichier)

Exemples de find

- les fichiers .c de plus de 40 Ko

```
% find . -name '*.c' -size +80 -print
./pvm3/src/lpvm.c
./pvm3/src/pvmd.c
```

- mes fichiers

```
% find / -user $USER -print
```

- nettoyage : suppression des fichiers a.out ou .o datant de plus de 7 jours

```
% find $HOME \( -name a.out -o -name '*.o' \) \
  -atime +7 -exec rm {} \;
```

Recherche de motifs dans un fichier grep

`grep [-vin] regexpr [file]...`

- Cherche et affiche les lignes contenant regexpr
- Options
  - -v : complémentaire
  - -i : maj. et min. indifférentes
  - -n : affiche les numéros de lignes

Exemple de grep

```
$more texte
il fait beau
il fait chaud
beau temps n'est ce pas ?

$ grep "^il" texte
il fait beau
il fait chaud
```

Exercice

- Exercices sur grep et find

Plan

Filtres Unix

- Filtres unix $\Leftrightarrow$ Filtre en traitement du signal
  - filtre signal : prend un signal en entrée, le modifie, l'envoie sur sa sortie pour être éventuellement repris par un autre filtre
  - filtre unix : lit des données sur son entrée standard, les modifie, les écrit sur sa sortie standard. Ils peuvent être enchaînés avec des tubes
- possibilité d'écrire des filtres en C

Filtres connus sous Unix

- tr
- cut
- sort
- paste
- uniq
- sed
- awk

Tous ces filtres

- lisent donc leurs données sur l'entrée standard et
- écrivent leurs résultats sur la sortie standard

Filtre `tr`

Syntaxe = `tr [options] chaine1 chaine2`

- tr = translate characters
- substitution ou suppression de caractères sélectionnés
- Un caractère $\in$ *chaine1* est remplacé par le caractère de même position dans *chaine2*
- Options principales :
  - `-d` : suppression des caractères sélectionnés
  - `-s` : "aaaaa" dans *chaine1* $\rightarrow$ "a" dans *chaine2*
- Abréviations :
  - `[a-z]` = segment de 26 car. allant de 'a' à 'z'
  - `[a*n]` = a...a (n fois)
  - `\xyz` = désigne le car. de code octal `xyz`

Exemple

```
$echo "coucou" | tr [a-z] [A-Z]
COUCOU

$ echo "aaabbbaaa" | tr -s [a-z] [A-Z]
ABA

$tr -d''150''' < fich_MS_DOS.c > fich_Unix.c
# le CR est elimine : cas transfert fichier MS DOS vers Unix
```

Filtre cut

- cut = sélectionne des caractères selon leur position dans la ligne
- Position par rapport :
  - au n°d'octet (`cut -b` *`no octet`*)
  - au rang du car. (`cut -c` *`rang`*)
  - au n°de champ (`cut -f` *`champs`* `-d` *`delimiteur`*)
- Option
  - `-s` (avec `-f`) : supprime les lignes vides

Exemple

```
#!/usr/bin/sh
# Selection sur le no d'octet
cut -b 5,7-10  fich_cut.txt

# Selection sur le rang du caractere
cut -c 1-5,10  fich_cut.txt

# Selection sur les champs delimites par un ":"
cut -d: -f1,3  fich_cut.txt
```

Filtre `sort`

- sort = trie, fusionne un ou plusieurs fichier(s)
- tri lexicographique
- Options :
  - `-o` *`fichier`* le résultat est mis dans *fichier*
  - `-k` *`n1,n2`* : clé = champ *n1* à *n2*
  - `-t` *`délimiteur`* : car. de séparation des champs (avec `-k`)
  - `-u` : (unique) efface toutes les lignes sauf une qui ont la même clé

Exemple

```
#!/usr/bin/sh

# Exemple de tri
sort fich_sort1.txt

# Exemple de fusion
sort fich_sort1.txt fich_sort2.txt

# Exemple avec delimiteur
sort -t: -k2 fich_sort1.txt fich_sort2.txt

# Exemple avec cle unique
sort -t: -k2  -u fich_sort1.txt fich_sort2.txt
```

Filtre paste

- usage : `paste` *fichier1 fichier2...*
- concatène les lignes de même n°dans *fichier1* et *fichier2*
- Exemple

```
  #!/usr/bin/sh
paste fich_paste1.txt fich_paste2.txt
```

Filtre `uniq`

- `uniq` = donne un seul exemplaire des lignes
- Options :
  - `-u` seules les lignes en 1 exemplaire sont affichées
  - `-c` donne le nombre d'exemplaires de chaque ligne

Exemple

```
#!/usr/bin/sh

# Affichage des lignes en 1 seul exemplaire
uniq fich_uniq.txt

# Affichage des lignes qui sont en 1 seul exemplaire
uniq -u fich_uniq.txt

# Affichage des lignes avec leur nombre d'occurences
uniq -c fich_uniq.txt
```

Conclusion sur les commandes/filtres

- Beaucoup de commandes/filtres...
  (faire `ls /usr/bin`)
- Savoir qu'elles existent
- Savoir ce que l'on peut en attendre
- man ...

Plan

Processus/Programmes

- Programme exécutable = commande
  = lignes de codes
  = statique
- Processus
  = exécution d'un programme
  = dynamique

E/S standards

Par défaut, un processus

- à une entrée standard (n°0) : le clavier
- à une sortie (n°1) : l'écran
- à une sortie des erreurs (n°2) : l'écran

Les redirections

On peut rediriger les E/S d'un processus avec :

- > pour la sortie standard
- < pour l'entrée standard
- 2> pour la sortie erreur (sous bash et batch)

Redirections écran

- `echo ‘‘coucou’’ > fich`

```
%cat fich
coucou

% ls -l
total 6
-rw-r--r--   1 jacob parole 7 sep 22 16:00 fich
-rw-r--r--   1 jacob parole 7 sep 22 16:00 fich2
-rw-r--r--   1 jacob parole 7 sep 22 16:00 fich3
```

- `ls -l > listfich`

```
% more listfich
total 6
-rw-r--r--   1 jacob parole 7 sep 22 16:00 fich
-rw-r--r--   1 jacob parole 7 sep 22 16:00 fich2
-rw-r--r--   1 jacob parole 7 sep 22 16:00 fich3
-rw-r--r--   1 jacob parole 0 sep 22 16:03 listfich
```

Redirection clavier 1/2

Exemple avec la commande `mail`

```
% mail jacob@lium.univ-lemans.fr
test 1 pour redirection clavier
.

% mail
From Bruno.Jacob@lium.univ-lemans.fr Wed Sep 22 17:16:30 2004
Date: Wed, 22 Sep 2004 17:16:14 +0200 (MEST)
From: Bruno Jacob <Bruno.Jacob@lium.univ-lemans.fr>
Message-Id: <200409221516.i8MFGEg24569@lucke.univ-lemans.fr>
Content-Length: 33

test 1 pour redirection clavier

? q
%
```

Redirection clavier 2/2

```
% echo "test 2 pour redirection clavier" > clavier
```

```
$ mail jacob@lium.univ-lemans.fr < clavier
```

```
% mail
From Bruno.Jacob@lium.univ-lemans.fr Wed Sep 22 17:20:56 2004
Date: Wed, 22 Sep 2004 17:20:41 +0200 (MEST)
From: Bruno Jacob <Bruno.Jacob@lium.univ-lemans.fr>
Message-Id: <200409221520.i8MFKfo24579@lucke.univ-lemans.fr>
Content-Length: 33

test 2 pour redirection clavier

? q
%
```

Redirection sortie erreur

```
% ls
clavier    fich       fich2      fich3      listfich
```

sous sh, bash, batch

```
% ls toto 2> erreurs
```

```
% more erreurs
toto: Ce fichier ou ce répertoire n'existe pas
%
```

Concaténation des redirections

- >> pour la sortie standard
- 2>> pour la sortie des erreurs

Concaténation sortie standard

```
%  more fich
coucou
```

```
% echo salut >> fich
```

```
%  more fich
coucou
salut
```

Concaténation sortie erreur

```
% more erreurs
toto: Ce fichier ou ce répertoire n'existe pas
```

```
% cat titi 2>> erreurs
```

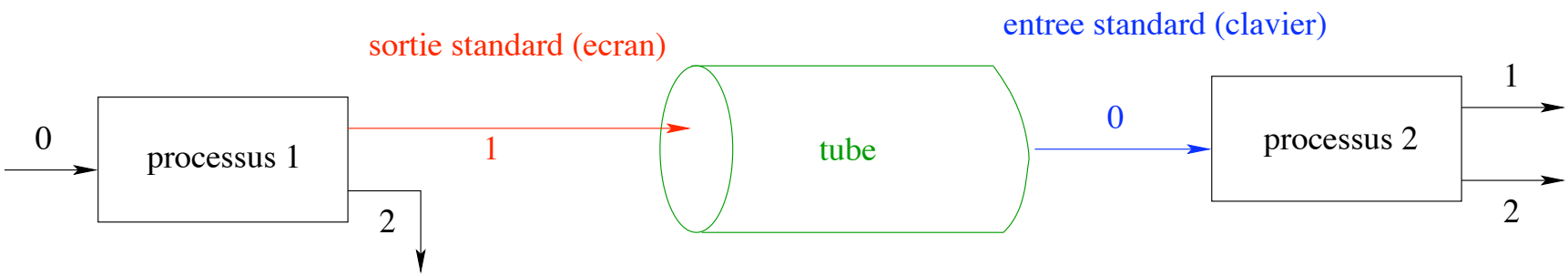
```
% more erreurs
toto: Ce fichier ou ce répertoire n'existe pas
cat : impossible d'ouvrir titi
%
```

Tubes

Un tube est une redirection entre 2 processus/commandes

```
% processus1 | processus2
```

- la sortie standard (n°1) du processus1 est redirigée vers l'entrée standard (n°0) du processus2
- le résultat du processus1 est la donnée du processus2



Tubes

Exemple avec les commandes

- ps : affiche à l'ecran les processus en cours d'execution
- wc : sans argument wc attend que l'on tape les lignes au clavier pour les compter

```
%  ps
   PID TTY      TIME CMD
  1357 pts/5    0:01 tcsh
 18745 pts/5    0:45 emacs
 22793 pts/5    0:04 xfig

%  wc -l
lig1
lig2
       2
```

```
% ps | wc -l
       3
%
```

Exercice

exercice sur

- les redirections de fichiers
- les tubes (pipes)

Plan

Enchainement des commandes

- ; séquencement (synchronisme)
  les commandes s'excutent séquentiellement
- & lancement en *background* (asynchronisme) les commandes s'exécutent en parallèle

Exemples

- `xterm ; netscape ; xemacs`
  → `xterm` puis `netscape` puis `xemacs`
- `xterm & netscape & xemacs &`
  → `xterm` en même temps que `netscape` et que `xemacs`

Plan

Consultation des processus

Liste des processus qui "tournent" sur votre machine

- `ps`
- Options :
  - par défaut : liste de vos processus
  - `-a` : tous les processus de tous les utilisateurs
- donne les n^o de processus (PID)

Exemples ps

```
ps
```

```
  PID TTY           TIME CMD
 1616 ttys000    0:00.06 -bash
```

```
ps -a
```

```
  PID TTY           TIME CMD
 1615 ttys000    0:00.03 login -pf jacob
 1616 ttys000    0:00.06 -bash
 3880 ttys000    0:00.00 ps -a
```

Arrêt d'un processus

"Tuer" un processus (qui boucle par exemple)

- `kill` PID
- envoie un signal au processus n^o PID
- Signaux :
  - par défaut envoie le signal 3 (`kill -3`)
  - -9 signal "d'arrêt d'urgence"

Sous Linux :

- `killall` *nom d'un processus*
- "tue" tous les processus avec ce nom

Filtre `uniq`

- `uniq` = donne un seul exemplaire des lignes
- Options :
  - `-u` seules les lignes en 1 exemplaire sont affichées
  - `-c` donne le nombre d'exemplaires de chaque ligne

Exercice

Exercices sur

- le séquencement de commandes
- les processus en background
- l'arrêt de processus qui "bouclent"

Plan

Programmation shell

- ensemble de commandes dans un fichier ayant éventuellement des paramètres
- nom de la commande = nom du fichier = `script shell`
- paramètres repérés par leur position quand on appelle la commande
- les commandes sont regroupées par une syntaxe de *langage de commandes*
- plusieurs langages → plusieurs syntaxes regroupées en 2 familles
  - les Bourne Shells (`sh,ksh,,bash`)
  - les C-Shells (`csh,tcsh`))

Ordre des opérations

1. Ouvrir un fichier texte et mettre en 1^{ere} ligne le shell choisi
   `# !/usr/bin/sh`
2. Ecrire la commande
3. La rendre exécutable :
   `chmod a+x` *votre commande*

Exemple

1. Ecriture de la commande dans un fichier `cmd1.sh`

```
#!/usr/bin/sh
date
echo "Catalogue accueil    =" $HOME
echo "Catalogue de travail = \c" ; pwd
echo "Nombre de fichiers   = " ; ls | wc -l
```

2. La rendre exécutable

```
chmod a+x cmd1.sh
```

3. L'exécuter

```
cmd1.sh
```

Variables utilisateurs

- On peut créer des variables `V` par affectation et les utiliser avec `$V`
- Manipulation des expressions arithmétiques avec `expr`

```
#!/usr/bin/sh
C="abc"
N=12
echo "variable C = " ${C}
echo "variable N = " ${N}
echo "variable N+1 = \c" ;  expr ${N} + 1
N=‘expr ${N} + 10‘
echo "variable N apres N=N+10= " ${N}
```

Variables prédéfinies

- $* : liste des paramètres
- $# : nb de paramètres
  (sans le nom de la commande)
- $$: n^o processus en cours
- $! : n^o du dernier processus en background
- $? : code retour de la dernière commande exécutée

Exemples

```
$cat cmd2.sh

#!/usr/bin/sh
echo "Nombre de parametres = " $#
echo "Parametre numero 0 = " $0
echo "Parametre numero 1 (nom du catalogue) = " $1
echo
echo "No du processus en cours ($0) = $$"
pwd &
echo "No du dernier processus en background (pwd)= $!"
echo
date
echo "Nombre de fichiers dans le catalogue $1 = " ; ls $1 | wc -l
```

Exemples

```
$cat cmd3.sh

#!/usr/bin/sh

ls
echo "exemple code de retour OK (ls)= $? "

ls toto
echo "exemple code de retour KO = $? "
```

La commande `test`

Syntaxe `test` *exp* ou `[` *exp* `]` ;
Elle permet :

- de réaliser des tests
- de restituer le résultat sous la forme d'un code retour (=0 si OK, ≠0 sinon)

Quelques exemples de *exp* :

- `-r` *fichier* : vrai si on peut lire dans fichier
- `-x` *fichier* : vrai si on peut l'exécuter
- `-d` *fichier* : vrai si répertoire
- `-n` *c1* : vrai chaine non nulle
- *c1* `=` *c2* : vrai si c1 = c2
- *c1* `!=` *c2* : vrai si c1 ≠ c2

faire un `man` ...

Entrée standard

Par la commande

`read` *variable*

Exemples :

```
#!/usr/bin/sh

read v
echo $v
```

Sous shell

Forcer l'interprétation de *string* → ‘*string*‘
(appel d'un sous shell)

Exemple :
`$echo ls` → `ls`
`$echo ‘ls‘` → liste des fichiers

Sélection

if ...then ...else ...fi

Syntaxe :

```
if <liste de commandes 1> then
     <liste de commandes 2>
else
     <liste de commandes 3>
fi
```

Exemples

```
$cat cmd6.sh

#!/usr/bin/sh

# if ( test $# = 0 ) then  ou
if [ $# = 0 ]; then
    echo "Il n'y a pas de parametres"
else
    echo "il y a $# parametres"
fi
```

Itération bornée

`for in ...do ...done`

Syntaxe :

```
for variable in "liste de valeurs" ; do
    action1 ; action2 ; ...
    action3
    action4
done
```

Exemples

Droits de tous les fichiers du répertoire courant :

```
#!/usr/bin/sh
for file in * ; do ls -l $file ; done
```

Affichage de chaînes de caractères :

```
for v in a bc cdef ; do echo $v; done
```

Affichage du nombre de lettres (à un près) :

```
for v in a bcd cdefrt ; do
    echo $v `echo $v | wc -c` "lettres" ;
done
```

Variables globales

Par la commande `export`, on peut rendre des variables name accessibles aux sous-shells

Exemple

- Dans le prg appelant :

```
#!/usr/bin/sh
var1=coucou
echo " "
echo "Dans cmd15_1.sh"
echo $var1
export var1
cmd15_2.sh
```

- Dans le prg appelé

```
#!/usr/bin/sh
echo "Dans cmd15_2.sh"
echo $var1
```

Tableaux en `bash`

Dans le shell `bash` on peut utiliser des variables

- de type tableau
- à 1 dimension
- de chaînes de caractères
- 1^{er} élément commence à 0
- l'indice * donne tous les éléments

Exemple

```
#!/usr/bin/bash
declare -a Tab  #facultatif

Tab=(il fait beau)

echo $Tab[1]
echo ${Tab[1]}
echo ${Tab[*]}
echo ${#Tab[*]}
```

Plan

Plan

FTP

FTP =

- File Transfert Protocole (Protocole de Transfert de Fichiers)
- protocole client-serveur qui permet à un utilisateur de transférer des fichier vers et depuis un serveur distant

Nécessite

- une connexion au serveur distant
- parfois un identifiant et un mot de passe
- d'invoquer les commandes pour effectuer les transferts entre votre machine et le serveur

Connexion

Pour se connecter au serveur server.fort.lointain.fr

```
$ ftp server.fort.lointain.fr
```

ou

```
$ ftp
ftp> open server.fort.lointain.fr
```

Identification

1. Saisie de l'identifiant

   `Name (server.fort.lointain.fr):`

   On peut saisir
   - un "vrai" mot de passe pour les sites privés
   - `anonymous` ou `ftp` pour les sites publics

2. Saisie du mot de passe

   `331...login ok,....`

   `Password:`

   Ce peut être :
   - un "vrai" mot de passe
   - une adresse e-mail ...

Si la connexion est réussie :

```
Remote system type is UNIX. Using binary mode to transfer files.
ftp>
```

On peut alors taper les commandes de transfert de fichiers

Type de Transfert des fichiers

2 modes :

- ascii : pour transférer des fichiers textes
- binaire : pour tous les autres fichiers (bianires, images. . .)

on peut changer de mode de transfert durant la connexion
Exemples de mode de transfert :

- fichier noyau du système → binaire
- images (jpeg, pdf. . .) → binaire
- fichiers compressés (zip, gzip) → binaire
- en cas de doute → binaire

Explorer et voir la liste des fichiers

- `ls` → affichage des fichiers sur le serveur
- `ls` est exécutée sur le serveur distant

Exemple :

```
ftp> ls 200 PORT command successful.
150 Opening ASCII mode data connection for /bin/ls.
total 33590
-r--r--r--   1   root   other 34348506 Dec 03 03:53 IAFA-LIST
lrwxrwxrwx   1   root   other        7 Jul 15 1997  README ->
-rw-r--r--   1   root   other      890 Nov 15 13:11 WELCOME
dr-xr-xr-x   2   root   other      512 Jul 15 1997  bin
dr-xr-xr-x   2   root   other      512 Jul 15 1997  dev
dr-xr-xr-x   2   root   other      512 Jul 18 1997  etc
drwxrwxrwx  11   ftp    20        4608 Nov 28 16:00 incoming
lrwxrwxrwx   1   root   other       13 Jun 04 1998  ls-lR -> l
dr-xr-xr-x  17   root   root       512 Jun 08 11:43 pub
```

Télécharger des fichiers

Pour télécharger un fichier du serveur vers votre machine :

- commande `get` → un fichier
- commande `mget` → plusieurs fichiers

Exemples :

1. téléchargement de linux-2.2.13.tar.gz.

```
ftp> get linux-2.2.13.tar.gz
local: linux-2.2.13.tar.gz remote: linux-2.2.13.tar.gz
200 PORT command successful.
150 Opening BINARY mode data connection for linux-2.2.1
```

→ `ftp` commence à sauvegarder le fichier distant vers votre ordinateur.

2. téléchargement de tous les fichiers commençant par "linux"

```
ftp> mget linux*
```

Transfert/Upload

Pour transferer des fichiers depuis le répertoitre courant de votre machine vers le serveur :

- commande `put` → un fichier
- commande `mput` → plusieurs fichiers

Exemple :

1. Transfert du fichier `Fiona.tar.gz` dans le répertoire `Incoming` du serveur `serveur.fort.lointain.fr`

```
ftp> cd /incoming
ftp> put fiona.tar.gz
local: fiona.tar.gz remote: fiona.tar.gz
200 PORT command successful.
150 Opening BINARY mode data connection for fiona.tar.g
226 Transfer complete.
10257 bytes sent in 0.00316 secs (3.2e+03 Kbytes/sec)
```

Transfert/Upload

Si le fichier à transférer n'est pas dans le répertoire courant
→commande `lcd`

- Local Change Directory
- change le répertoire courant de votre machine

Exemple : `fiona.tar.gz` se trouve dans le répertoire `donjon` de votre machine :

```
ftp> lcd /donjon
Local directory now donjon
```

Utiliser des commandes shell

- Le client ftp permet l'utilisation du point d'exclamation (!) pour effectuer des commandes locales. Par exemple, pour afficher la liste des fichiers dans le répertoire local en cours, entrez ceci :

  ```
  ftp> !ls
  ```

- Cette commande appelle les fonctions du shell, et utilise la commande indiquée après le " !". Vous pouvez utiliser n'importe quelle commande que votre shell supporte en appelant le " !". Notez que !cd ne changera pas de répertoire local, c'est pourquoi la commande lcd existe.

Progression des transferts

Progression de vos transferts pendant l'utilisation de FTP → commande `hash` :

```
ftp> hash
Hash mark printing on (1024 bytes/hash mark).
```

affiche une indication tous les 1024 bytes pendant le téléchargement.

Il y a aussi l'option tick :

```
ftp> tick
Tick counter printing on (10240 bytes/tick increment).
Bytes transferred: 11680
```

Autres commandes FTP

Il existe beaucoup d'autres commandes ftp
Pour plus d'informations :

- dans ftp : `help` *nom de la commande* : pour une commande spécifique de ftp
- dans un terminal : `man ftp` pour le manuel d'aide sur ftp

Plan

ssh

Secure Shell

Plan

Éditeur sed

`sed [-n] [-e` *`script`*`] [`*`fichier`*`]`

ou

`sed [-n] [-f` *`fichier_script`*`] [`*`fichier`*`]`

- `sed` : *streameditor*
- Prend ses données (des lignes)
  - dans l'entrée standard
  - dans des fichiers si `-f` *fich* (1 fichier par `-f`)
- Affiche ses résultats sur
  - la sortie standard
  - ne les affiche pas si `-n`

`sed` modifie les lignes à partir de `script`

- `script` = commandes d'édition
  - en ligne avec `-e` *`script`* (1 commande par `-e`)
  - contenu dans un fichier avec `-f` *`fichier_script`*

  Si il y a seulement une option `-e` et pas de `-f` alors on peut omettre le `-e`

Commandes d'édition

`[addresse [,adresse]] fonction [arguments]`

- sélectionne les lignes selon les `adresses`
- leur applique une `fonction` de sed avec ses `arguments`

Adresses de sed

- vide $\longrightarrow$ toutes les lignes sont sélectionnées
- *n* $\longrightarrow$ la ligne de numéro *n* dans chaque fichier
- \$ $\longrightarrow$ seulement la dernière ligne de chaque fichier
- *n1*,*n2* $\longrightarrow$ n^{o}de lignes entre *n1* et *n2*
- `/expression reguliere/` $\longrightarrow$ définit un contexte d'adresse

Contexte d'Adresses

- décrit le contexte dans lequel doivent être les lignes sélectionnées.
- définit par une `/expression reguliere/`
  - `sed` supporte les expressions régulières étendues (voir cours 2)
  - + `\n` (NEWLINE)

Fonctions de `sed`

Il existe beaucoup de fonctions (Commandes d'édition) (`man sed`)
Parmis les plus utilisées :

- `a` (append) ajoute du texte
- `c` (change) remplace la ligne
- `d` (delete) efface la ligne
- `w` *fichier* (write) écrit la ligne dans *fichier*

Exemples

```
$ cat script.sed
1a\
zzzzzz

$ cat test_sed.txt
aaaaaa
bbbbbb
cccccc

$ sed -f prog.sed test_sed.txt
aaaaaa
zzzzzz
bbbbbb
cccccc
```

Fonction s de sed

C'est **la** plus utilisée

`s/reg-exp/remplacement/flags`

- `s` (substitute)
- `reg-exp` expression régulière de sed
  - Stockage `\( reg-exp \)`
  - Rappel `\1 \2 \3 ...`
- `flags` :
  - `g` (global) (faire toutes les substitutions de la ligne)
  - `n` avec $n \in [1 - 512]$ remplace seulement la n^{ieme} occurence
  - `p` (print) si ok sort la ligne sur la sortie standard
  - `w` *fichier* (write) si ok écrit la ligne dans *fichier*

Exemples

```
$ cat f
aaa bbb aaa bbb aaa
aaa ccc ddd

$ sed -e 's/aaa/AAA/' f
AAA bbb aaa bbb aaa
AAA ccc ddd

$ sed -e 's/aaa/AAA/g' f
AAA bbb AAA bbb AAA
AAA ccc ddd

$ sed -e 's/aaa/AAA/3' f
aaa bbb aaa bbb AAA
aaa ccc ddd
```

Exemples

- Utilisation du stockage et du rappel des champs
- Inversion des 2 premiers champs

```
$ cat f
aaaaa:bbbbbb:ccccc
ddddd:ee:ff
ggg:hhhhh:iiii

$sed -e 's/\^\([^:]*\):\([^[:]*\)/\2:\1/' f
bbbbbb:aaaaa:ccccc
ee:ddddd:ff
hhhhh:ggg:iiii
```