Cours Unix 2

Michel Mauny

INRIA ROCQUENCOURT

ETGL

Plan du cours 2

1. Recherches de et dans des fichiers
2. Comparer des fichiers: `cmp`, `diff`, `patch`
3. Disques, partitions, fichiers, ...
4. (Parenthèse: installer Linux sur un PC)
5. Processus

Recherches de et dans des fichiers

On va examiner ici plusieurs commandes effectuant des recherches de fichiers ou dans des fichiers.

Recherche de chaîne dans un fichier: grep

UNIX offre la commande `grep` pour afficher les lignes de fichiers donnés en arguments et qui contiennent un motif donné. La syntaxe à utiliser est:

```
grep [option] motif [nom_fichier]
```

Ex:

```
$ grep son fic?
fic1:Ceci est le fichier fic1 du repertoire personnel.
fic3:Ceci est le fichier fic3 du repertoire personnel.
```

Recherche de chaîne dans un fichier: grep

Options:

- `-v`: affichent les lignes qui ne contiennent pas le motif;
- `-i`: ignore la distinction minuscule/majuscule dans les comparaisons.

Recherche de chaîne dans un fichier: grep

Ex:

```
$ grep -i est fic?
fic1:Ceci est le fichier fic1 du repertoire personnel.
fic2:echo Il est: `date | awk '{printf "%s\n", $4}'`
fic3:Ceci est le fichier fic3 du repertoire personnel.
fic4:CECI EST LE FICHIER fic4.
$ grep -v est fic*
fic2:#!/bin/sh
fic4:CECI EST LE FICHIER fic4.
fic4:
```


Recherche d'un fichier: find

La commande `find` descend récursivement dans des sous-arborescences de répertoires, en imprimant le nom ou en cherchant à appliquer une commande donnée à des fichiers précisés par un ou plusieurs critères de sélection (nom, type, date de modification, etc.).

Recherche d'un fichier: find

La commande `find` s'utilise de la façon suivante:

`find liste_de_répertoires expression`

Où

- `liste_de_répertoires` est la liste des racines des arborescences à parcourir ;
- `expression` est une suite d'options exprimant les critères de sélection des fichiers et les actions à leur appliquer. Lorsque que le critère est vrai, l'action est exécutée. Dans la suite, on appelle fichier courant, le fichier examiné par la commande find à un moment donné.

Recherche d'un fichier: find

Voici quelques unes des options de sélection:

- `-name motif`: vrai si le motif s'applique sur le nom du fichier courant;
- `-user nom_utilisateur`: vrai si le fichier courant appartient à l'utilisateur `nom_utilisateur`;
- `-mtime n`: vrai si le fichier a été modifié dans les `n` derniers jours (`+n` pour exprimer `n` *et plus* et `-n` pour exprimer `n` *et moins*).

Recherche d'un fichier: find

Les éléments de l'expression peuvent être connectés par les opérateurs logiques suivants:

- négation: `!`
- et: simple juxtaposition des éléments
- ou: `-o`

  Dans ce cas, il est nécessaire d'entourer l'expression complète avec des parenthèses (qu'il faut précéder d'une contre-barre pour qu'elles ne soient pas interprétées par le shell).

Recherche d'un fichier: find

Les actions à effectuer sur les fichiers sélectionnés sont:

- `-print`: affiche le nom du fichier courant;
- `-exec com`: exécute la commande sur le fichier courant; `com` est terminée par le marqueur `\;` et le paramètre spécial `{}` désigne le fichier courant.

Recherche d'un fichier: find

Ex:

```
$ find . -name 'fic?' -print
./rep1/fic1
./rep1/fic4
./fic1
./fic2
./rep3/fic1
$ find rep1 rep2 \( -name 'fic?' -perm 755 \) -print -exec rm {} \;
rep1/fic1
rep2/fic5
```

Recherche d'un fichier: find

Note: on peut utiliser la commande grep comme filtre pour éliminer les informations inintéressantes produites par une commande précédente comme dans l'exemple suivant:

```
find . -name 'fic*' -exec cat {} \; | grep est
```

Affiche les lignes des fichiers dont les noms commencent par la chaîne `fic`, qui appartiennent à l'arborescence de racine définie par le répertoire courant et qui contiennent le motif `est`.

Comparer des fichiers

La commande `cmp`

```
cmp fichier1 fichier2
```

`cmp` compare deux fichiers de type quelconque et écrit le résultat sur la sortie standard.

Par défaut, `cmp` est silencieux si les fichiers sont identiques. S'ils diffèrent, les numéros d'octet et de ligne de la première différence détectée sont rapportés.

```
$ cd /boot
$ cmp System.map-2.4.24 System.map-2.4.24.old
System.map-2.4.24 System.map-2.4.24.old differ: \
                                byte 145321, line 5064
```


Différences entre fichiers

Deux commandes importantes:

- `diff`: imprime les différences entre fichiers texte;
- `patch`: met à jour un fichier à partir de ses différences avec un autre.

Si `diff` effectue une sorte de soustraction entre deux fichiers, `patch` effectue une sorte d'addition.

Usage de diff

`diff [options] from-file to-file`

```
$ cat f1
une seule ligne
$ cat f2
une seule ligne
et ici une deuxieme.
$ diff f1 f2
1a2
> et ici une deuxieme.
```


Context diff

```
$ diff -c f1 f2
*** f1  2003-10-23 22:06:16.000000000 +0200
--- f2  2003-10-23 22:06:38.000000000 +0200
***************
*** 1 ****
--- 1,2 ----
  une seule ligne
+ et ici une deuxieme.
$ diff -c f1 f2 > delta
$ patch <delta
patching file f1
$ diff f1 f2
```


Options de diff

Options principales:

- `-b` ignore les changements concernant seulement les espaces
- `-i` ignore les changements concernant seulement la casse
- `-c` donne des lignes de contexte
- `-e` produit un résultat utilisable par l'éditeur `ed`

Diff admet aussi des répertoires en argument:

- alors le contenu des répertoires est comparé
- l'option `-r` va traiter récursivement ces répertoires.

Usage de patch

En général:

`patch [-p n] < patchfile`

car les noms de fichiers à traiter sont dans `patchfile`.

Options (linux):

- `-p`n efface des chemins d'accès aux fichiers le plus petit segment initial contenant n caractères "/"
- `-R` "renverse" le patch.

Fonctionnement de `patch`

Plus précisément, comment ça marche:

- si le patch est (partiellement) applicable au fichier, la version originale est sauvegardée dans `fichier.orig`
- les patches rejetés sont sauvés dans `fichier.rej`
- si on se trompe de sens, `patch` propose de «renverser» le patch
- `patch` a un peu d'intelligence: la commande essaie d'appliquer le patch à quelques lignes près.

Disques, partitions, fichiers, . . .

Types de fichiers

- **Fichiers «normaux»:** suite d'octets d'une certaine longueur (taille du fichier).
- **Liens symboliques** (voir ci-après).
- **Répertoire:** table associant des inodes (adresses) à des noms.
- **Fichiers spéciaux:** des pilotes (drivers) vus comme des fichiers. Ils répondent aux appels `open`, `close`, `read`, `write` et `ioctl` (contrôle).

Fichiers spéciaux

Deux types de fichiers spéciaux:

- *block devices*: associés à des disques, lecteurs de bandes, *etc.* Entrées-sorties «bufferisées».
- *character devices*: associés à des terminaux, souris et autres périphériques (reéls ou virtuels).

Ces *devices* sont visibles depuis le système de fichiers, et sont généralement situés dans `/dev`.

Partitions

Un disque est partagé en partitions (disques logiques, volumes) contenant chacune un système de fichiers (*filesystem*), sauf pour les partitions de *swap* destinées à fournir la mémoire virtuelle.

Une partition peut être

- *locale*, cad être physiquement située sur un support mémoire (disque) connecté à l'ordinateur)
- *distante*, accédée au travers du réseau (elle peut alors ne pas correspondre exactement à une vraie partition de son support physique). Habituellement *via* NFS (*Network File System*).

Partitions

Les partitions locales sont vues comme des *devices* `/dev/...`

≪Monter≫ une partition, c'est ≪accrocher≫ son système de fichiers dans l'arborescence courante.

```
$ mount
/dev/hda6 on / type ext2 (rw)
/dev/hda8 on /home type ext2 (rw)
/dev/hda5 on /mnt/windowsXP type vfat (rw)
```

Les répertoires `/`, `/home`, ..., sont appelés ≪points de montage≫.

Les ≪volumes≫ montés peuvent contenir des systèmes de fichiers différents. Ils présentent tous une interface commune (ouvrir, lire, écrire).

Informations relatives aux fichiers

Informations produites par `ls -l`:

```
-rw-rw-r-- 1 tom users 5945 Dec 1 17:59 unix2.txt
```

type de fichier	`d, b, c, -`
droits	`rwx, s, t`
nombre de liens physiques	ici, 1
propriétaire	`tom`
groupe	`users`
taille	nombre d'octets, 5945
date de dernière modification	`Dec 1 17:59`
nom	`unix2.txt`

Informations relatives aux fichiers

Commandes modifiant le statut des fichiers:

- `touch`: met à jour la date de dernière modification. Crée le fichier s'il n'existe pas.
- `chown`: change le propriétaire
- `chgrp`: change le groupe

Adresse des fichiers

`ls -i`: affiche le numéro d'inode des fichiers (l'inode est le numéro associé à un fichier: son adresse dans la partition, en quelque sorte).

```
$ ls -il unix2.txt
96109 -rw-rw-r-- 1 tom users 5945 Dec 1 17:59 unix2.txt
```


Modifications de l'ordre de listage des fichiers (alphabétique sur le nom par défaut):

- `-t`: par date de dernière modification
- `-c`: par date de dernière modification du statut (`touch`, `chown`, ...)
- `-S`: par taille
- `-u`: par date de dernier accès
- `-r`: dans l'ordre inverse de l'ordre spécifié

Lien physique: deux noms différents pour un même fichier (inode)

```
$ ls -il unix2.txt
96109 -rw-rw-r-- 1 tom users 5945 Dec 1 17:59 unix2.txt
$ ln unix2.txt autre_nom
$ ls -l unix2.txt autre_nom
-rw-rw-r-- 2 tom users 5945 Dec 1 17:59 autre_nom
-rw-rw-r-- 2 tom users 5945 Dec 1 17:59 unix2.txt

$ ls -il unix2.txt autre_nom
96109 -rw-rw-r-- 2 tom users 5945 Dec 1 17:59 autre_nom
96109 -rw-rw-r-- 2 tom users 5945 Dec 1 17:59 unix2.txt
```


Partage d'inode ⇒

- on ne peut établir de lien physique entre différentes partitions;
- on ne peut établir de lien physique entre répertoires (un système de fichiers est un graphe acyclique dont seules les ≪feuilles≫ peuvent être partagées).

NB: les ≪pointeurs arrière≫ "." et ".." ne sont pas comptés ici comme créant des cycles: ils servent essentiellement à remonter dans l'arborescence.

Liens symboliques

Liens symboliques: fichier spécial contenant le chemin d'accès à un autre fichier (correspond aux «raccourcis» de Windows).

```
$ ln -s unix2.txt lien_symb
$ ls -il unix2.txt liens_symb
96110 lrwxrwxrwx 1 tom users    9 \
                   Dec 1 18:57 lien_symb -> unix2.txt
96109 -rw-rw-r-- 1 tom users 5945 Dec 1 17:59 unix2.txt
```


(Parenthèse: installation de Linux)

Une recette

Ingrédients: préparer

- un PC avec ou sans MS-Windows installé
- un outil de manipulation de partitions (si MS-W préinstallé)
- un CD ou DVD Linux

Pour installer Linux,

1. si nécessaire, dégager de l'espace libre sur le disque du PC
2. insérer le CD/DVD
3. démarrer la machine et suivre les instructions.

Retirer du feu, et servir. À consommer sans modération.

Processus

Processus

Activation d'un programme. (Plusieurs activations ⇒ plusieurs processus.)

```
$ ps
  PID TTY          TIME CMD
11071 pts/1    00:00:00 bash
11807 pts/1    00:00:00 man
11810 pts/1    00:00:00 sh
11811 pts/1    00:00:00 sh
11816 pts/1    00:00:00 less
11969 pts/1    00:00:00 ps
```


Processus: une grande famille

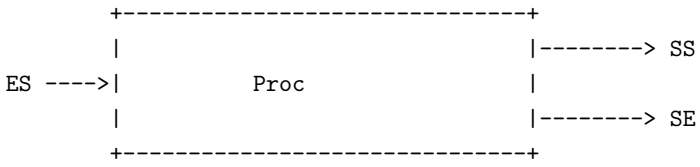
Processus:

- identifié par un numéro (PID)
- est lancé par un autre processus: *fils* d'un *processus parent*
- renvoie un *code de retour* lorsqu'il se termine

```
$ pstree
init-+-apmd
     |-atd
     |-crond
    ...
     |-gnome-terminal-+-bash-+-man---sh---sh---less
     |                |      `-pstree
     |                `-gnome-pty-helpe
    ...
```


Redirections

Un processus en général:



Une instance particulière

Redirections

- `<input`: redirige l'entrée standard
- `1>output` ou bien `>output`: redirige la sortie standard
- `1>>output` ou bien `>>output`: redirige la sortie standard
- `2>output`: redirige la sortie d'erreurs
- `2>>output`: redirige la sortie d'erreurs
- `2>&1` indique que la sortie d'erreurs sera ouverte en écriture vers la sortie standard.
- `>&-`, `1>&-`, , `2>&-`, ferme la sortie
- `<<etiquette`: redirige l'entree standard vers le texte qui suit la commande jusqu'à la prochaine occurrence de `etiquette` seule sur une ligne.

Redirections

Les redirections sont traitées **dans l'ordre** par le shell (*cf.* 3ième et 4ième lignes ci-dessous).

Avec redirections et pipes, on peut ≪brancher≫ et ≪assembler≫ des processus à loisir:

```
ls >t
ls 2>t
ls 1>&2 2>t
ls 2>t 1>&2
ls >>t
head -2
head -2 <t
ls | sort
```


Redirections

```
cp f g
cat f >g
cat <f >g
tee g <f >/dev/null
sh f
sh <f
cp f g </dev/null >/dev/null 2>/dev/null

ed -s /etc/passwd<<FIN
/tom/p
q
FIN
```


La commande `kill`*, les signaux*

Tuer le processus `pid`: `kill pid`

Envoyer un signal particulier à `pid`: `kill -signal pid`

Lister les signaux disponibles: `kill -l`

Les signaux servent à plus de choses qu'à stopper les processus: ils servent aussi à . . . signaler!

La commande `kill`*, les signaux*

```
$ kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL
 5) SIGTRAP      6) SIGABRT      7) SIGBUS       8) SIGFPE
 9) SIGKILL     10) SIGUSR1     11) SIGSEGV     12) SIGUSR2
13) SIGPIPE     14) SIGALRM     15) SIGTERM     17) SIGCHLD
18) SIGCONT     19) SIGSTOP     20) SIGTSTP     21) SIGTTIN
22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO
30) SIGPWR      31) SIGSYS
```

Shell: les caractères génériques

Les caractères génériques

- sont des caractères interprétés spécialement par le shell, afin de produire des noms de fichiers **existants**;
- peuvent représenter un ou plusieurs caractères (`?`, `*`).

Shell: les caractères génériques

Rappels:

`*`: l'étoile représente (filtre) toute chaîne de caractères (ne contenant pas `/`).

L'étoile seule filtre tout nom de fichier, sauf ceux commençant par ≪`.`≫.

`?`: représente tout caractère (sauf le ≪`.`≫ initial).

Shell: les caractères génériques

`[ab123]`: représente tout caractère parmi `ab123`.

`[a-zA-Z]`: représente tout caractère alphabétique.

`[Ab-em-z]`: représente `A`, tout caractère compris entre `b` et `e` et tout caractère de `m` à `z`.

`[!a-zA-Z]` désigne tout caractère non alphabétique.

Dans un mot, le caractère ≪\≫ inhibe l'effet du caractère suivant (y compris de lui-même).

Commandes, ksh

Les commandes sont (pour la plupart) des noms de fichiers.

Les commandes sont recherchées dans une liste de répertoires: le `PATH`.

`PATH` est une *variable d'environnement*, qui est transmise d'un processus à tous ses fils.

```
$ echo $PATH
/bin:/usr/bin:/usr/X11R6/bin:/usr/local/bin:\
      /home/tom/bin:/sbin:/usr/sbin
```

Commandes, ksh

Autres variables intéressantes:

- `HOME`
- `USER`
- `PS1`

Les variables de ksh seront décrites plus en détail durant les cours ultérieurs.

Les fichiers d'initialisation de `ksh`

Au login, `ksh` lit le fichier `$HOME/.profile`.

Chaque nouveau processus `ksh` lit le fichier `$HOME/.kshrc` (valeur de la variable `ENV`).

Attention: `$HOME/.kshrc` peut être lu par des instances de ksh qui n'ont pas de terminal associé: il faut donc éviter d'y appeler des commandes qui sont d'une façon ou d'une autre associées au terminal (imprimer un message, régler différents aspects du terminal).

Les modifications de `PATH` vont généralement dans `$HOME/.profile`