Introduction à UNIX

Présentation globale d'UNIX (1)

UNIX :

- *Standard* pour les systèmes d'exploitation
- Disponibilité de nombreux utilitaires de manipulation de données textuelles
- Facilité de communication de données entre différents programmes

Des types d'UNIX : Linux, Solaris

Dans le cadre du cours, utilisation d'un UNIX : Linux

Présentation globale d'UNIX (2)

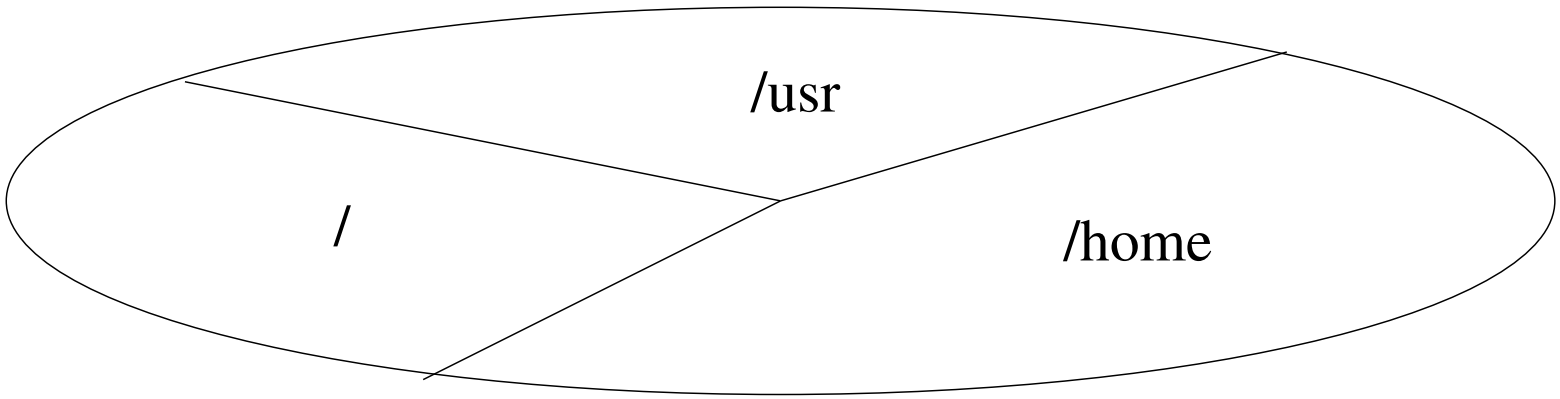
Quelques caractéristiques importants des systèmes UNIX :

- Multi-utilisateurs et multi-tâches
- Temps partagé
- Système de fichiers hiérarchique
- Entrées-Sorties intégrées au système de fichiers
- Interface utilisateur interactive (shell ou interpréteur de commandes)

Organisation du système de fichiers sous UNIX

Système de fichiers :

- Structure logique permettant la gestion de l'espace disque
- Chaque disque logique (partition) possède un système de fichiers

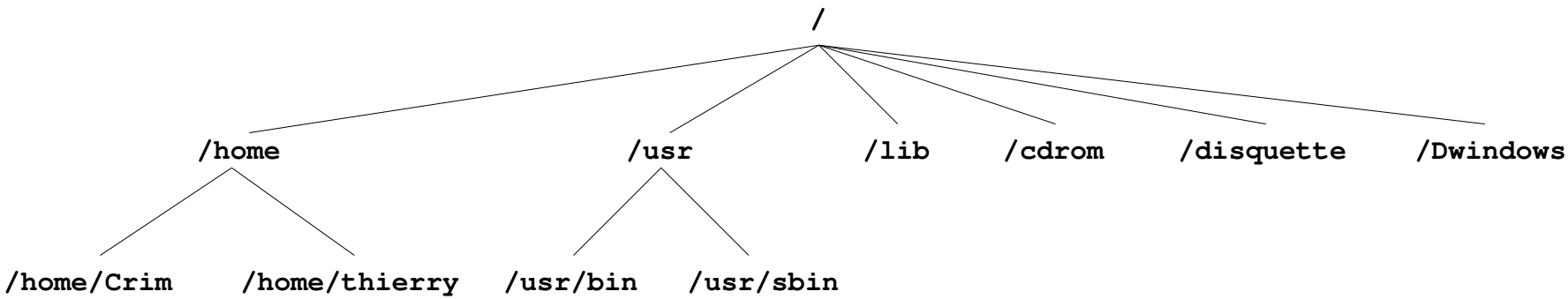


Organisation des systèmes de fichiers dans une arborescence

Hiérarchisation des systèmes de fichiers

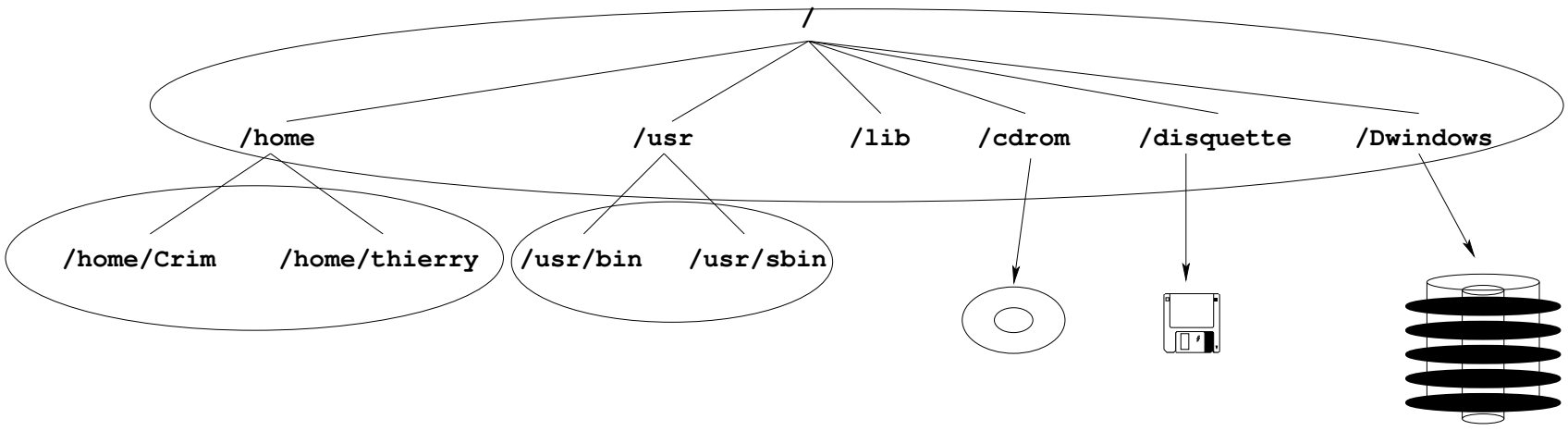
Structure arborescente d'UNIX (1)

Rattachement des systèmes de fichiers *secondaires* au système de fichiers principal



Structure arborescente d'UNIX (2)

Accès aux disques (disquettes, CDROM, etc.) comme à un répertoire



Les fichiers (1)

Représentation de toutes les ressources disponibles (fichiers *classiques*, dispositifs diverses, périphériques)
Plusieurs types de fichiers :

- Les fichiers réguliers ou ordinaires
- Les fichiers répertoires
- Les fichiers spéciaux (clavier, écran, etc.)
- Les tubes nommés (points de communication)
- Les liens symboliques (identification de données avec plusieurs noms)

Les fichiers (2)

Nom des fichiers : jusqu'à 255 caractères alphanumériques
Sous UNIX : différenciation des lettres en majuscule et en minuscule
Exemple : `air` et `AIR` sont deux chaînes de caractères différentes
Fichiers cachés : nom commençant par un point (`.cshrc`, `..`)
Deux fichiers particuliers :

. : le répertoire courant

.. : le répertoire père

Les fichiers ordinaires

Pour le système, pas de distinction au niveau du contenu
Mais pour les commandes, distinction importante :

- Les fichiers binaires
- Les fichiers textes : structurés en ligne

Ligne :

- suite de caractères se terminant par un caractère `<fin de ligne>`, non visible `a l'´ecran

Informations sur les fichiers (1)

Chaque fichier :

appartient à un utilisateur (user) et un groupe (group)

possède des droits décrivant leurs modes d'utilisation et d'accès :

- lisible
- modifiable
- exécutable (pouvant être ouvert, dans le cas d'un répertoire)

Informations sur les fichiers (2)

Répartition des droits sur trois niveaux correspondant à trois classes d'utilisateurs :

- Le propriétaire du fichier (*user*)
- Le groupe auquel appartient le propriétaire (fixé par l'administrateur du système) (*group*)
- Les autre utilisateurs (*other*)

Un utilisateur à part possédant tous les droits : le *super utilisateur* (`root`)

Informations sur les fichiers (3)

En résumé, chaque fichier possède des droits en

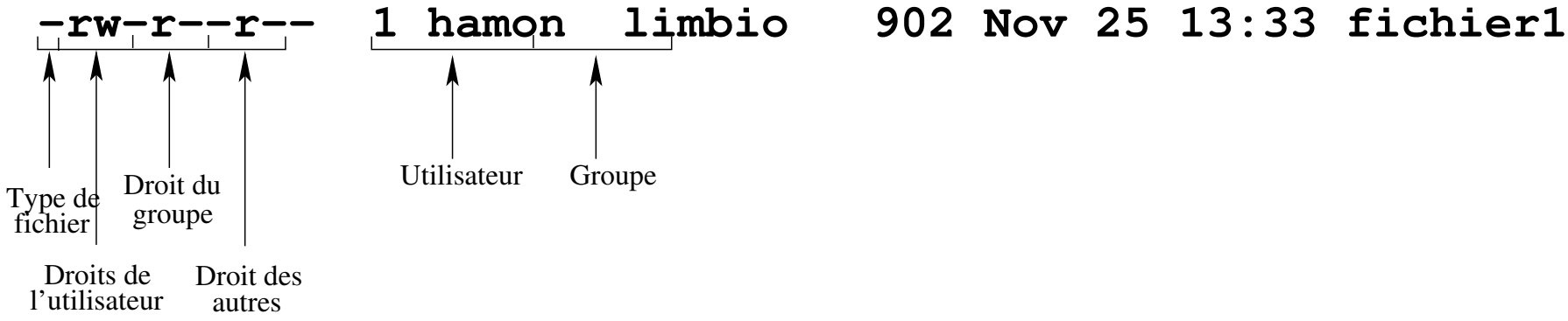
- lecture (r)
- écriture (w)
- exécution (x)

pour

- l'utilisateur `user`
- le groupe `group`
- les autres `other`

Visualisation des informations sur un fichier

À l'aide de la commande `ls` et de l'option `-l` (ligne de commande `ls -l` :



Les processus

Rappel : Programme en exécution dans un système UNIX

Toute activité correspond à un processus

Tout processus correspond à un fichier ou un ensemble de fichiers (dans le répertoire `/proc` sous Linux)

Création avec fork ou exec dans un programme C

Par défaut, trois fichiers standards associés à un processus :

Entrée standard

Sortie standard

Sortie des erreurs

Description d'un processus

PID : l'identifiant du processus

PPID : l'identifiant du père du processus

UID : identifiant de l'utilisateur propriétaire

priorité : valeur entre -20 (priorité élevée) et +20 (priorité faible)

terminal de contrôle (tty)

répertoire courant

mémoire utilisée (vive et swap)

temps d'exécution (réel, et au niveau de l'utilisation processeur)

Organisation

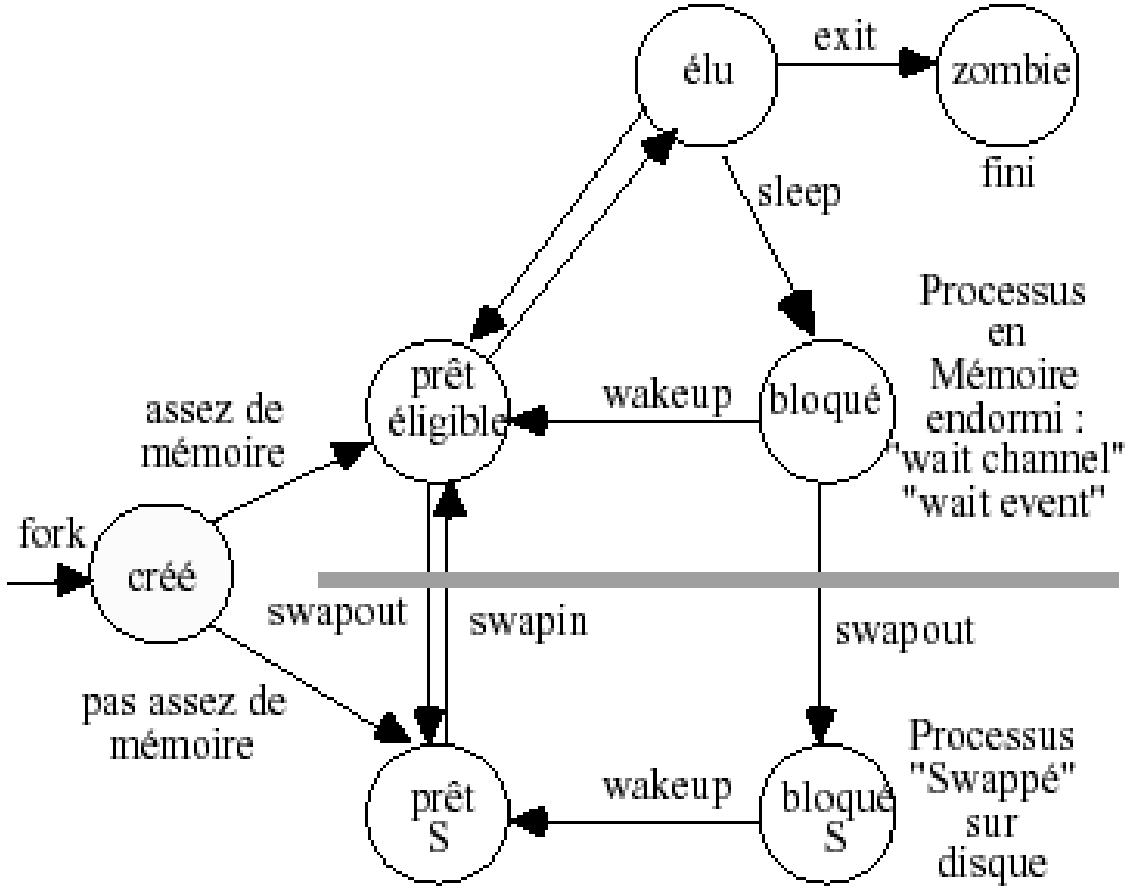
2 types de processus :

- processus système : Processus lié au fonctionnement du système (init, udevd, rpcbind, etc.)
- processus utilisateur : exécution d'une commande, d'un programme ou d'une application (apache, ls, ...)

Organisation des processus selon un arbre hiérachique

- La racine : processus `init` (PID = 1)
- Chaque nœud : un processus
- Un processus créé est lié à son père (PPID) tant que celui-ci existe, sinon il est lié au processus de PID 1

Etats d'un processus



Etats d'un processus

`IDLE` : Le processus en cours de création

`RUN` : Le processus est en exécution. Il dispose du processeur.

`Pret` : Le processus attend que le système lui attribue le processeur.

`SLEEP` : Le processus est en attente d'un évènement particulier (entrée/sortie parexemple).

`STOP` : Le processus est prêt mais ne demande pas l'accès au processeur.

`ZOMBIE` : Le processus se termine. Il attend que son père prenne en compte sa terminaison et que le système libère ses ressrources.

Signaux

Mécanisme simple de communication entre processus et de contrôle d'exécution

En général, un nombre (ou un code mnémonique) entre 1 et 31 (extensible avec la norme POSIX) :

- 2 (`INT`) : interruption au clavier (`CTRL-c`)
- 9 (`KILL`) : terminaison du processus (non modifiable)
- 10 (`USR1`) : définition par l'utilisateur
- 11 (`SEGV`) : violation mémoire (référence invalide)
- 17 (`CHLD`) : terminaison d'un fils
- 20 (`TSTP`) : arrêt du processus (`ctrl-z`)

(voir `/usr/include/bits/signum.h` sous Linux)

Signaux

Comportement :

- actions prédéfinies (par défaut)
- actions redéfinies par le développeur
- actions ignorées

Envoi de signaux à partir du terminal : `CTRL-C`, `CTRL-Z`, ...
(voir le résultat de la commande `stty -a`)

Utilisation d'UNIX

Procédures de connexion et de déconnexion

Commandes de base

- Gestion et déplacement dans l'arborescence
- Gestion et manipulation des fichiers
- Manuel en ligne

Connexion et déconnexion

Généralités

Sur une machine UNIX, un utilisateur doit être référencé pour pouvoir l'utiliser

Authentification d'un utilisateur :

- `nom d'utilisateur` (*login*)
- `mot de passe` (*password*)

Possibilité d'accéder à son répertoire personnel (*répertoire personnel*)

Connexion

Deux modes : texte et graphique
Connexion : indiquer

- son nom d'utilisateur (entrée `login`)
- son mot de passe (entrée `password`)

Par exemple :

```
login:  hamon <Enter>
Password:  ******** <Enter>
```

Déconnexion

En mode texte, 3 possibilités :

`Votre Nom@Nom de la Machine:1:>` logout

`Votre Nom@Nom de la Machine:1:>` exit

`Votre Nom@Nom de la Machine:1:>` CTRL-D

En mode graphique :
(Variation d'un gestionnaire de fenêtres à un autre)
Dans les menus, demander la déconnexion, et confirmer la demande

Multiboot et arrêt de la machine

Multiboot :

- Chaque PC poss`ede deux syst`emes d'exploitation
- Possibilite´ du choix du syst`eme au d´emarrage

Arrêt : Après s'être déconnecté, dans un menu, cliquez sur `shutdown`

N.B. : Ne jamais arrêter brutalement la machine (sauf en cas d'extrême nécessité)

Redémarrage (*Reboot*), 2 possibilités :

- Dans un menu, cliquez sur `reboot`
- Tapez la séquence de touche : CRTL-ALT-SUPPR

Editeur de texte : emacs

Exécution : `emacs`
Utilisation du menu ou de commandes basées sur des combinaisons de touches
Utilisation important des touches ESC, META, ALT, CTRL, SHIFT

Sortie : CTRL-x CTRL-c

Lecture d'un fichier : CTRL-x CTRL-f

Sauvegarde d'un fichier : CTRL-x CTRL-s

Destruction d'un buffer d'Emacs : CTRL-x k

Aide :

Fonction d'une touche : CTRL-h c ou CTRL-h k
Description d'une fonction : CTRL-h f

Commandes UNIX de base

Généralités

Commande : programme effectuant une tache particulière (copie de fichier, tri d'un fichier, etc.)

(UNIX fournit de nombreux programmes de base)

Ligne de commande : série de caractères tapés et se terminant par un retour chariot (entrée, Enter)

La commande est la première série de caractères jusqu'à un séparateur

Les arguments sont les éléments suivants (séparés par des espaces)

Exécution des commandes à travers un interpréteur de commandes (*Shell*) dans un terminal

Syntaxe standard des commandes

```
commande [argument(s)]
```

En général :

- Lecture des données sur l'entrée standard (par défaut, le clavier)
- Écriture des données en sortie sur la sortie standard (par défaut l'écran)
- Écriture des erreurs sur la sortie en erreur standard (par défaut, l'écran)

Rappel : sous UNIX, les minuscules et les majuscules sont différenciés

Visualisation du contenu d'un répertoire

`ls` *options répertoires*

Options (les plus courantes) :

- `-a` : Affichage des fichiers cachés ou non
- `-l` : Affichage de toutes les informations sur les fichiers (droits d'accès, propriétaire, taille, dernière date de mise à jour)

Exemple :

```
ls -l
```

Affichage à l'écran du contenu du répertoire courant avec les informations sur les fichiers :

```
-rw-r--r-- 1 hamon limbio 902 Nov 25 13:33 exo1.c
drwxrwx--- 2 hamon limbio 512 Dec 30 10:43 Repertoire1
```

Déplacement dans un répertoire

`cd` *répertoire*

Exemple : `cd /home/air/hamon/Repertoire1` (Déplacement dans le répertoire `/home/air/hamon/Repertoire1`)
Utilisation particulière :

- `cd` : Retour au répertoire de connexion
- `cd ..` : Retour au répertoire père
- `cd -` : Retour au répertoire précédent

Affichage du répertoire courant

```
pwd
```

Exemple :

```
cd /users/limbio/hamon/Repertoire1
pwd
```

Affichage à l'écran :

```
/users/limbio/hamon/Repertoire1
```

Création et destruction d'un répertoire

Création : `mkdir répertoire`
Destruction (d'un répertoire vide) : `rmdir répertoire`
Destruction (d'un répertoire contenant des fichiers) : `rm -r répertoire`

Exemples :
`mkdir Repertoire1` (création du répertoire `Repertoire1`)
`rmdir Repertoire1` (destruction du répertoire `Repertoire1`)

Copie de fichiers (1)

cp *options fichier1 fichier2*
(Copie du contenu de fichier1 dans fichier2)
cp *options fichier1* [*fichier2* ...] *répertoire*
(Copie du contenu des fichiers fichier1 et fichier2 dans répertoire)

Options :

- *-i* : mode interactif (confirmation avant écrasement)
- *-R* ou *-r* : copie récursive de répertoires
- *-p* : conserve la date du fichier source

Copie de fichiers (2)

Exemples :

```
cp /etc/fstab .
cp -r rep1 rep2
```

Déplacement de fichiers

`mv` *options fichier1* [*fichier2* ...] *répertoire*
Déplace les fichiers *fichier1* et *fichier2* dans *répertoire*
Options :

- *-i* : mode interactif (confirmation avant écrasement).
- *-f* : force la commande

Exemple : `mv fichier1 rep1`
N.B. : La commande peut être utilisée pour renommer un fichier

Destruction de fichiers

`rm` *options fichier1* [*fichier2* ...]
Suppression des fichiers spécifiés
Options :

- *-i* : mode interactif (confirmation avant écrasement)
- *-f* : force la commande
- *-r* : supprime récursivement les répertoires

Exemples :

```
rm fichier1
rm -r repertoire1
```

Visualisation ou concaténation de fichiers

`cat` *fichier*
Affichage du(des) fichier(s) donnés en argument à l'écran (la sortie standard)
Utiliser pour ajouter plusieurs fichiers les uns à la suite des autres (concaténation)
Par défaut, la lecture s'effectue sur l'entrée standard et l'affichage sur la sortie standard
Exemple :
`cat /etc/fstab`
Affichage à l'écran du contenu du fichier `/etc/fstab`.

Visualisation du début/de la fin d'un fichier
Visualisation d'un fichier page par page

`head` *-n fichier* : affichage des n premières lignes d'un fichier

`tail` *-n fichier* : affichage des n dernières lignes d'un fichier

Par défaut, *n* vaut *10*

`more` *fichier* : Affichage à l'écran, page par page, des fichiers spécifiés

Tri des lignes d'un fichier (1)

`sort` *-ufnr -o fic fichier...*
Par défaut, tri par ordre alphabétique
Options :

- *-u* : n'affiche qu'une fois les lignes multiples
- *-f* : ne différencie pas les minuscules des majuscules
- *-n* : effectue un tri numérique
- *-r* : ordre décroissant
- *-o fic* : spécifie un fichier de sortie

Tri des lignes d'un fichier (2)

Autre option :
-k `<premier champ><type>,<dernier champ><type>`

Exemple : `sort -k 2n,2n` (Tri sur le deuxième champ)

Décompte sur un fichier

`wc` *-lwc fichier...*

Options :

-l : Affiche que le nombre de lignes

-w : Affiche que le nombre de mots

-c : Affiche que le nombre de caractères

Si aucune option n'est spécifiée, la commande affiche le nombre de lignes, de mots et de caractères

Substitution de caractères

`tr` *caracteres caracteres* $<$ *fichier*
Substitution d'un caractère par un autre
Suppression des caractères invisibles
On peut utiliser le code hexadécimal du caractère

Exemple :
`tr '\011' '@' < fichier1`

Affichage des colonnes d'un fichier

`cut` *-d delim -f champs fichier*
Options :

-d delim : spécifie le délimiteur (entre double quote, ") ;

-f champs : spécifie les champs à afficher (séparés par une virgule).

Le délimiteur par défaut : la tabulation
Exemple :
`cut -f1 /etc/fstab`
Affichage de la première colonne du fichier `/etc/fstab`
(séparateur par défaut)

Modification des droits d'un fichier/répertoire (1)

`chmod` *[-R] <classe d'utilisateur>< nature de la modification>< droit fichiers>*

`-R` : Opération récursive sur les fichiers et les sous-répertoires

classe d'utilisateur :

- u : le propriétaire
- g : le groupe
- o : les autres
- a : les trois classes

Modification des droits d'un fichier/répertoire (2)

nature de la modification :

- \+ ajout du droit
- − retrait du droit
- = affectation du droit

droit :

- r droit en lecture
- w droit en écriture
- x droit en exécution

Modification des droits d'un fichier/répertoire (3)

Exemples :

```
chmod g-rw fichier1
chmod -x fichier2
```

Recherche de fichiers ou de répertoires

`find` *répertoire options*
Recherche dans toute l'arborescence à partir du point spécifié
Options courantes :

-name *fichier* : recherche sur le nom *fichier*

-print : affiche le résultat de la recherche

! : négation de la recherche

Exemple :

```
find / -name "fstab" -print
```

Recherche des fichiers dont le nom est `fstab`, dans tous les répertoires

Recherche d'une chaîne de caractères dans un fichier

Recherche du motif indiqué dans le(s) fichier(s) indiqué(s)

`grep` *options expression fichier*

Famille des grep

- egrep (extended)
- fgrep (fast)

Caractères spéciaux

car	signification
.	n'importe quel caractère
$	fin de ligne
^	début de ligne
\	caractère de despécialisation
[]	groupement exclusif
[^]	négation dans les groupements
(|)	disjonction sur plusieurs caractères
?	0 ou 1
+	1 à n
*	0 à n
{n}	nombre précis d'occurrences

Grep

options

unité des traitements : la ligne

options :

- i : insensibilisation à la casse
- n : numéro de la ligne
- c : nombre de lignes
- v : tout sauf le motif indiqué

Recherche et transformation de motifs

sed

`sed s/motif/remplacement` :

- `motif` : expression régulière (voir `grep`)
  Définition de sous-bloc à l'aide de `\(` et `\)` (des sous-blocs peuvent être imbriqués)
- `remplacement` : chaine remplaçant le motif correspondant
  Utilisation des sous-blocs à l'aide de référence `\1`, `\2`, ... (dans l'ordre d'apparition des sous-blocs)

Lecture des données sur l'entrée standard ou d'un fichier donné en argument

Ecriture sur la sortie standard

Manuel UNIX et commandes diverses

Manuel UNIX :

man *nom* : visualisation de l'aide de la commande ou de la fonction spécifiée
Exemple : `man ls` : Affichage de l'aide de la commande ls

Commandes diverses :

- `echo` *-n message* (affichage d'un message à l'écran)
  Option : *-n* (Pas de retour chariot final)
- `clear` (effacement de l'écran)
- `tar` (archivage)
- `ps` (visualisation des programmes exécutés sur la machine – les processus)
- `kill` (Terminaison d'un processus)
- `which` (Recherche d'une commande)

Commandes associées aux processus

`ps [OPTIONS]` : affichage de la liste des processus (suivant les options)

`ps` : processus associés au terminal

`ps aux` : tous les processus en exécution dans le système

`kill -<NUMSIGNAL> PID` : Envoi d'un signal NUMSIGNAL au processus d'identifiant PID

`nice` / `renice` : Gestion de la priorité des processus

`top` : liste de processus avec évolution en temps réel (`q` pour quitter)

reprend les commandes précédentes

Les *jobs*

Gestion interactive des processus lancés dans un terminal (travaux ou *jobs*)
Remarques :

Lors de l'exécution d'une commande ou d'un programme dans un terminal, **le processus associé est en avant plan**
Tant que le processus n'est pas terminé, il n'est pas possible d'accèder à l'interpréteur de commande et de lancer l'excution d'une nouvelle commande

Le caractère & à la fin de la ligne de commande permet **lancer la commande en arrière plan**
Le processus reste associé au terminal (donc les sorties standards restent l'écran/le terminal)

Basculement en arrière plan pendant l'exécution :
`CTRL-Z` puis bg

Les *jobs*

Identification par numéro (1, 2, 3, etc.) correspondant à l'ordre d'exécution des processus

Commandes associées
NB : Identifiant précédé par %

- `jobs` : affichage de la liste des *jobs* associés au terminal
- `fg [identifiant]` : basculement en avant plan d'un processus (arrêté ou en arrière-plan
  `fg` (valable pour le dernier jobs)
  `fg %n` par le n-ème *job*
-
- `bg [identifiant` : basculement en arrière plan d'un processus (arrêté ou en arrière-plan
- Egalement `kill`

Autre commande

`nohup ligne de commande &` : exécution de la commande en arrière plan et détachée du terminal courant
(l'exécution continue après la fin de la session utilisateur)

Interpréteur de commande : introduction

Interpréteur de commande : l'interface interactive entre l'utilisateur et le système d'exploitation

- Rédaction par l'utilisateur de *lignes de commandes* ou de *scripts*
- Interprétation par le shell qui les exécute

Disponibilité de plusieurs mécanismes de communication au niveau du shell :

- Redirection d'entrée/sortie (programme = filtre)
- Tubes

Possibilité d'utilisation d'expressions régulières

Les fichiers standard

Chaque programme se voit associer trois fichiers standard :

- Le fichier d'entrée contenant les données utilisées par le programme (par défaut, le clavier)
- Le fichier de sortie contenant les résultats du programme (par défaut l'écran)
- Le fichier de sortie des erreurs lors de l'exécution du programme (par défaut, l'écran)

Redirection d'entrée/sortie (1)

Redirection sortante

Redirection de l'écran vers un fichier, pour stocker les résultats

Avec le signe de redirection sortante : >
Programme > *fichier*
Exemple :
`tail fichier.txt > 10premieres lignes.txt`
Attention, > écrase le fichier s'il existe. Pour ajouter (concaténer) les résultats dans un fichier, on utilise le signe >>

Option `-o` pour certaines commandes

Redirection d'entrée/sortie (2)

Redirection entrante

La plupart des commandes : données traitées par défaut à partir du clavier, si pas de mention de nom de fichiers en argument

Cas particulier : la commande `tr` lit obligatoirement le clavier.

Pour que les commandes lisent les données dans un fichier, on peut utiliser le symbole de redirection entrante <

Redirection d'entrée/sortie (3)

Redirection entrante

Exemple :

```
tr '\011' '\@' < fichier.txt
tail < fichier.txt
```

Les deux modes de redirection peuvent être combinés :

```
tail < fichier.txt > fin_fichier.txt
```

Les tubes

Visualiser les résultats d'une commandes :

sauvegarde dans un fichier, puis utilisation `cat` ou `more`

Passage d'informations d'une commande à une autre :

Création d'un canal de communication, symbolisé par `|` : le tube

Les données produites par la première commande sont communiquées à la seconde commande pour qu'elle puisse les traiter

Exemple :

```
head fichier1.txt | tail -2
```

Affichage de la neuvième et de la dixième ligne du fichier `fichier1.txt`

Les scripts (1)

Combinaison de commandes permettant d'effectuer une tâche spécifique
Pour les tâches reproduites fréquemment : rédaction de scripts shell qui exécute la suite de commandes
Exemple :

```
#!/bin/sh
# 10 dernieres lignes triees

sort $1 | tail
```

$1 est une variable correspondant au premier argument

Les scripts (2)

Si on ne met pas de variable (donc pas de prise en compte d'arguments), le programme agit comme un filtre et lit les données sur l'entrée standard

Pour exécuter le script, il faut donner les droit en exécution sur le script

Les scripts sont exécutables comme des programmes quelconques

Shells : Introduction (1)

Langage de programmation interprété
$\longrightarrow$: Interpréteurs de commande standard du système UNIX

Ouverture de session

Dialogue avec le système d'exploitation par l'intermédiaire d'un interpréteur de commandes

Avantages : clair et facile à relire
"Inconvénient" : demande de la rigueur lors de la programmation

Shells : Introduction (2)

Nombreux shells répartis en 2 familles :

- Famille des Bourne Shells (Steve Bourne) : `sh`, `bash`
- Famille des C Shells : `csh`, `tcsh`

`ksh` (David Korn) : famille des Bourne Shells avec améliorations issues des C Shells

Présentation du Bourne Shell et de sa programmation

Notions communes aux shells

Exécution de trois types de commandes :

- *commandes internes* faisant partie du shell
- *commandes externes* contenues dans les répertoires
- *fonctions ou alias* définissant de nouvelles commandes

Utilisation interactive du shell

Complétion d'une commande : Début de la commande suivi de `[TAB]`

Gestion de l'historique des commandes :

- Affichage : `history`
- Exécution de la n-ième commande de l'historique : `!n`

```
!324
gcc -o testIUW testIUW.c
```

- Exécution de la dernière commande commençant par la chaine `XXX` : `!XXX`

```
!g
gcc -o testIUW testIUW.c
```

Commandes internes

Pas de création de nouveau processus

- commandes normales, fréquemment utilisées : `cd`, `pwd`, `umask`, `set`, etc.
- directives algorithmiques : `if`, `then`, `else`, `while`, `case`, `for`, etc.

Commandes externes

Indépendantes du shell
Identiques quel que soit le shell utilisé
Accès grâce à la variable d'environnement `PATH` (énumération d'une liste de répertoires contenant les commandes)
Création d'un nouveau processus à chaque exécution

Priorité des commandes

commande définie dans une fonction shell (dans `.profile`)

commande interne

commande externe dans l'ordre des répertoires de `PATH`

Initialisation des shells

Utilisation de fichiers contenus dans le répertoire `HOME` (répertoire de connexion)

- `.profile`, `.bashrc` (Bourne Shell)
- `.cshrc`, `.tcshrc` (C Shell)

Permet :

- Exécution d'une commande
- Initialisation des variables d'environnement (utilisées par le shell ou les commandes)

Initialisation des Bourne Shells

Avant affichage de l'invite ($), exécution de 2 fichiers :

- `/etc/profile` géré par l'administrateur
- `.profile` dans le répertoire `HOME` géré par l'utilisateur

Après modification du fichier `.profile`, prise en compte de son contenu
Forcer l'exécution du fichier :

```
$ cd
$ . .profile
```

Variables d'environnement

Utilisation par certaines commandes
Exportées vers tous les fils du shell

- Initialisation automatique et la valeur ne doit pas être modifiée : `HOME`, `LOGNAME`, `USER`
- Initialisation automatique et la valeur peut être modifiée : `SHELL`, `PATH`, `TERM`
- Initialisation par l'utilisateur dans son fichier `.profile` ou en mode interactif : `PRINTER`, `LPDEST`

Initialisation (`export` indique qu'il s'agit d'une variable d'environnement) :

```
VARIABLE=valeur
export VARIABLE
```

Principales variables d'environnement (1)

`HOME` : (définie automatique) contient le nom du répertoire de connexion. La variable est utilisée par `cd`

`USER` (BSD 4.[23]) et `LOGNAME` (System V) : contient le nom de connexion (*login*) de l'utilisateur

`TERM` : (définie automatiquement par l'administrateur système) contient le nom du terminal utilisé.
Définition du terminal fournie aux commandes effectuant un gestion d'écran.

Principales variables d'environnement (2)

`TERMCAP` (BSD 4.[23]) : contient le chemin d'accès à la base des terminaux, ou la définition correspondant à la valeur de `TERM` lue dans le fichier `/etc/termcap`
Initialisation à l'aide de *tset*
Par défaut, non initialisée

`TERMINFO` (System V) : contient le chemin d'accès à la base des terminaux. Définition recherchée par défaut dans `/usr/lib/terminfo/X/$TERM`

`EXINIT` : contient les commandes exécutées lors de l'appel de l'éditeur *vi* (définition d'options de fonctionnement et des touches de fonctions)

Principales variables d'environnement (3)

`PATH` : contient la liste des répertoires dans lesquels un shell recherche une commande (nom des répertoires séparés par `:`)

`CDPATH` : contient la liste des répertoires balayés lors d'un changement de répertoire lorsque l'argument de *cd* est un nom relatif (nom des répertoires séparé par `:`)

`SHELL` : contient le nom du shell de connexion (utilisée par *make*, *mail*, *vi*, *more*, etc.

`MAIL` : contient le nom de la boîte aux lettres (`/var/mail/$LOGNAME`)

Principales variables d'environnement (4)

`PRINTER` (BSD 4.[23]) : définit le nom de l'imprimante utilisée (par défaut, *lp*)

`LPDEST` (System V) : redéfinit le nom de l'imprimante utilisée par défaut (redéfinition par l'administrateur système)

Caractères spéciaux

Caractères ayant une signification spéciale :

- Utilisés pour la génération de noms de fichiers (expressions régulières ou rationnelles) : *, ?, etc.
- Définis pour différentes fonctionnalités : #, $, &, etc.

Génération de noms de fichiers (1)

Génération par rapport aux fichiers du répertoire courant ou d'un répertoire désigné explicitement
Caractères de remplacement

`*` : n'importe quelle chaîne de caractères (y compris la chaîne vide)

`?` : un caractère quelconque (*joker*)

`[...]` : un des caractères entre crochets.
Définition par

- Énumeration (`[Aa]`)
- Intervalle suivant le code ASCII (`[A-Za-z]`)

La négation est exprimée avec le caractère `!` ou `^`

Génération de noms de fichiers (2)

Exemple :

```
$ echo .b*
.backups .bash_history .bash_logout .bash_profile
.bashrc

$ echo .[bc]*
.backups .bash_history .bash_logout .bash_profile
.bashrc .cshrc .cshrc.LIMBIO
```

Autres caractères spéciaux (1)

`#` : introduction d'un commentaire

```
# un commentaire
```

`$` : introduction d'une variable

```
echo $SHELL
```

`&` : terminaison d'une commande lancée de manière asynchrone (*background*)

```
sauvegarde.sh &
```

`;` : séparateur des commandes dans une ligne de commande

```
ls ; ps aux > fichier_ps.txt
```

Autres caractères spéciaux (2)

`>` `<` `>>` `<<` ` ‘ ` ` ‘ ` `|` `^` : définition de redirection (voir plus loin)

`’` : délimitation d’une chaîne de caractères. Pas de substitution des variables. Les caractères spéciaux perdent leur signification.

Exemple :

```
$ ls ’$HOME’
ls: $HOME: No such file or directory

$ echo ’$HOME’
$HOME
```

Autres caractères spéciaux (3)

" : délimitation d'une chaîne de caractères. Les variables sont substituées par leur valeur. Les caractères spéciaux perdent leur signification sauf " ' \ $
Exemple :

```
$ ls "$HOME"
Enseignement    Mail        tmp
Doc             Program     Recherche
AdminLinux      LIMBIO      Projets
Articles        Emacs       local
[thierry@broceliande thierry]$ echo "$HOME"
/home/hamon
```

\ : Perte de la signification du caractère spécial (*déspécialisation*)

Autres caractères spéciaux (4)

`(` et `)` : exécution des commandes entre parenthèses dans un sous-shell

```
$ pwd
/home/hamon
$ (cd /tmp; rm test.* ; ls )
$ pwd
/home/hamon
```

`{` et `}` : regroupement de commande dans une liste de commandes

Exécution d'une procédure de commande (`script`)

Deux manières :

- `sh fichierscript arg1 arg2`
  Utilisée lors de la mise au point, avec l'option `-x` (affichage des commandes et des arguments)
- `fichierscript arg1 arg2`
  Utilisée après mise au point
  Positionnement des droits d'exécution
  Script placé dans un répertoire défini dans la variable `PATH`

Appel quel que soit le shell courant : utilisation de la directive d'exécution `#!/bin/sh` ou `#!/bin/csh` (suivant le shell utilisé dans le script)

Passage des arguments

Argument reçu dans le script sous forme de variables $0 (nom du script), $1, $2, $3, $4, ..., $9.
Variables prédéfinies :

$# : nombre d'argument ($0 non compris)

$$: PID du sous-shell exécutant le script

$*, $@ : liste des paramètres

- "$*" équivalent à "$1 $2 $3 ..."
- "$@" équivalent à "$1" "$2" "$3" ...

$? : code de retour de la dernière commande

$! : numéro du dernier processus lancé de manière asynchrone

Redirections d'entrée/sortie standard (1)

L'entrée ou les sorties standard peuvent être redéfinies comme des fichiers

Redirection de l'entrée standard : <
L'entrée standard correspond à un fichier
Exemple :

```
$ tail < essai.c
```

Redirection de la sortie standard : >1
Exemple :

```
$ gcc essai.c -o essai >1 resultat-compil.out
```

Redirection de la sortie standard des erreurs : >2
Exemple :

```
$ gcc essai.c -o essai >2 resultat-compil.err
```

Redirections d'entrée/sortie standard (1)

Redirection des deux sorties : > et 2>&1
Exemple :

```
$ gcc essai.c -o essai >1 resultat-compil.out \
          >2 resultat-compil.err

$ gcc essai.c -o essai > \
                  resultat-compil.out+err 2>&1
```

Exécution différée (1)

Différentes méthodes d'exécution d'une commande :

Exécution interactive ou *foreground* : exécution dans le terminal et contrôle du terminal rendu après terminaison
Interruption avec le caractère de contrôle d'*interruption de processus* (en général `CTRL-C`)
Exemple :

```
$ gcc essai.c -o essai
```

Exécution différée (2)

Exécution asynchrone ou *background* : exécution dans le terminal suivie du caractère &. Le contrôle du terminal est rendu immédiatement
Utilisation pour l'exécution d'une commande dont le résultat n'est pas instantané (compilation longue, par exemple)
Exemple :

```
$ gcc essai.c -o essai &
```

Utilisation des redirections des sorties standard et des erreurs pour éviter les affichages intempestifs à l'écran

Exécution différée (3)

Abandon du shell : réception du signal `HANGHUP` par la commande en exécution asynchrone
Masquge possible du signal (et `TERM`) avec la commande *nohup*
Exemple :

```
$ nohup gcc essai.c -o essai > essai.out 2>&1  &
```

Structures de contrôle

Exécution de boucles

Mots clés reconnus en position syntaxique d'une commande (début de ligne ou précédé d'un caractère de séparation de commande

`&` `&&` `||`

test (1)

Permet des tests de fichiers et de comparaison élaborés
Retour :

- 0 si la condition est vraie
- différent de 0 si la condition est fausse

Syntaxe :

```
test expression
```

ou

```
[ expression ]
```

test (2)

Forme de l'expression :

- `-f name` : vrai si `name` est
  - un fichier régulier (sous System V) ;
  - un fichier au sens large (fichier régulier, fichier spécial bloc ou caractère, tube nommé) et n'est pas un répertoire
- `-d name` : vraie si `name` est un répertoire
- `-c name` : vraie si `name` est un fichier spécial caractère (System V)
- `-b name` : vraie si `name` est un fichier spécial bloc (System V)
- `-g name` : vraie si `name` existe avec le bit sgid positionné (System V)

test (3)

`-k name` : vraie si `name` existe avec le sticky bit positionné (System V)

`-u name` : vraie si `name` existe avec le bit suid positionné (System V)

`-p name` : vraie si `name` est un tube nommé (System V)

`-r name`, `-w name`, `-x name`, : vraie si `name` est un fichier avec accès en lecture, écriture, exécution

`-s name` : vraie si `name` est un fichier de taille non nulle

test (4)

`-z string` : vraie si `string` est une chaîne de caractères vide

`-n string` : vraie si `string` est une chaîne de caractères non vide

`s1 == s2` ou `s1 = s2` : vraie si les chaînes de caractères `s1` et `s2` sont égales (inégalité : `!=`)

`n1 -eq n2` : vraie si les chaînes de caractères `n1` et `n2` contenant des nombres entiers sont égales (utilisation aussi de `-ne`, `-gt`, `-ge`, `-lt`, `-le`

Opérateurs logiques : `!` (NON), `-a` (ET), `-o` (OU)

test (5)

Exemple :

```
if [ -f $1 -o -d $1 ]
then
    echo "Le fichier $1 est un fichier regulier
                          ou un repertoire"
else
    echo "Le fichier $1 n'est ni un fichier
                 regulier ni un repertoire"
fi
```

if (1)

Syntaxe :
(`else` facultatif)

```
if condition
then
      liste des commandes
else
      liste des commandes
fi
```

ou

if (2)

(`if` multiples)

```
if condition1
then
      liste des commandes
elif condition2
then
      liste des commandes
else
      liste des commandes
fi
```

if (Exemple)

```
if test $# != 1
then
     echo "Zero ou plusieurs arguments"
else
     echo "Un argument : $1"
fi
```

Calcul arithmétique (1)

Réalisation d'arithmétique élémentaire :
expr ou `$(( expression ))`
Opérateurs : `+ - * / %`
La multiplication `*` doit être déspécialisée si l'expression n'est pas quotée
Exemple :

```
$ a=10
$ a=‘expr $a + 1‘
$ # ou a= $(( $a + 1 ))
$ b=‘expr 10 + 5 \* 4‘
$ # ou b=$(( 10 + 5 \* 4 ))
$ echo "$a ; $b"
11 ; 30
```

Calcul arithmétique (2)

comparaison de chaînes

La forme `expr1 : expr2` permet la comparaison des deux motifs et l'affichage du nombre de caractères reconnus
Exemple :

```
$ a="a10zr"
$ nbchars=‘expr "$a" : ’.*’‘
$ echo $nbchars
5
```

Condition multiple *case*

Syntaxe :

```
case identificateur in
     motif1)
            liste de commandes
            ;;
     motif2)
            liste de commandes
            ;;
     motif3|motif4)
            liste de commandes
            ;;
...
     motifn)
            liste de commandes
            ;;
esac
```

Condition multiple *case* (Exemple)

```
case $1 in
        Sun) echo "Dimanche";;
        Mon) echo "Lundi";;
        Tue) echo "Mardi";;
        Wed) echo "Mercredi";;
        Thu) echo "Jeudi";;
        Fri) echo "Vendredi";;
        Sat) echo "Samedi";;
esac
```

for

Syntaxe :

```
for identificateur in liste_de_valeurs
do
        liste de commandes
done
```

for (Exemple)

```
for i in 1 2 3 4 5
do
   echo $i
done

for i in ‘seq 1 5‘
do
   echo $i
done
```

boucle *while*

Syntaxe :

```
while condition
do
   liste de commandes
done
```

Exemple :

```
i=1
fin=10
while [ $i -le $fin ]
do
   echo $i
   i=‘expr $i + 1‘
done
```

boucle *until*

Syntaxe :

```
until condition
do
   liste de commandes
done
```

Exemple :

```
i=1
until [ $i -gt 10 ]
do
   echo $i
   i=‘expr $i + 1‘
done
```

Les fonctions (1)

Mot clé : `function` ou non

Bloc d'instruction entre accolades

Valeur de retour : `return`

Localisation de variable (limite de visibilité) : `local`

Les fonctions (1)

```
function helloworld {

}

helloworld {
 echo "Hello World"

 return(1);
}
```