Commandes du système Linux

GUIDE D'UTILISATION DU SYSTÈME

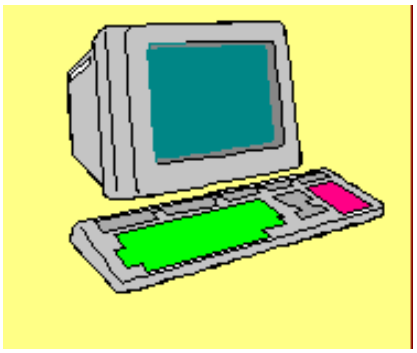


Table des matières

A. Environnement de l'usager

A.1 Branchement

C'est l'étape au cours de laquelle un usager s'identifie au système. La procédure consiste à taper son nom d'usager et son mot de passe à l'invitation du système.

Exemple:

```
Darkstar login: mailhotp
Password:

Last   successful login for mailhotp: Thu Mar 26 00:51:20 EST 1992 on
tty1A
Last unsuccessful login for mailhotp: Mon Mar 23 17:34:57 EST 1992 on
tty01

Linux 2.0.0

            ** Backup aujourd'hui a 23:00 **

you have mail

$
```

La permière ligne identifie le nom du système auquel l'usager veut accéder et lui demande de s'identifier.

La seconde ligne demande à l'usager d'entrer son mot de passe (cette information n'est pas affichée).

Si le système reconnaît l'usager (d'après les informations contenues dans le fichier /etc/passwd), il affiche les dates du dernier branchement réussi et du dernier branchement raté. Ces renseignements permettent à l'usager de détecter les intrusions possibles.

Le système affiche ensuite la version d'UNIX utilisée, puis le message du jour (fichier /etc/motd). Ce dernier est généralement utilisé pour informer les usagers des actions effectuées sur le système (arrêt, backup, ajout d'un périphérique, etc).

Si l'usager a reçu du courrier électronique depuis son dernier branchement au système, il en est avisé par un "**you have mail**".

Dans les environnements où plusieurs types de terminaux sont utilisés, l'usager est invité à préciser le type de terminal par lequel il est relié au système. Cette information est importante pour certains programmes, en particulier les éditeurs.

Finalement, UNIX affiche le caractère "$", l'indicatif du Bourne Shell, pour signaler à l'usager que le celui-ci est prêt à recevoir ses commandes. Si vous êtes en mode "super-usager", l'indicatif du shell devient "#" .

A.1.1 Le compte usager

Tout utilisateur d'un système UNIX doit posséder un "compte usager". Un compte usager est formé de plusieurs éléments qui, une fois combinés, donne accès aux ressources d'un système UNIX.

A.1.2 Nom d'usager

Un nom d'usager doit être unique et ne pas comporter plus de 8 caractères. Il est généralement constitué à partir du nom de famille et du prénom d'un individu.

A.1.3 Mot de passe

Il s'agit d'une séquence de lettres et de chiffres, connue seulement de l'usager, constituant une clé pour "entrer" dans son compte d'usager.

A.1.4 Répertoire maison

Le répertoire maison d'un usager (home directory) est le répertoire principal de l'usager, celui où prend racine sa propre arborescence de répertoires. C'est le répertoire initial dans lequel vous êtes lorsque vous vous branchez au système.

A.1.5 UID et GID

Pour identifier un usager UNIX utilise, en plus du nom d'usager, un numéro d'identification unique, le UID (User IDentification number).

Un usager se voit attribué également un numéro de groupe, le GID (Group IDentification number). Le concept de groupe sera abordé plus loin.

A.1.6 Boîte aux lettres

Pour être en mesure de recevoir du courrier électronique, un usager doit voir son compte associé à une boíte aux lettres électronique. Le système avertit l'usager lorsque du courrier l'attend.

A.1.7 Fichiers de démarrage

Des fichiers de démarrage sont exécutés après que l'usager se soit identifié au système et avant que n'apparaisse l'indicatif du shell. Ces fichiers de démarrage, qui sont aussi des fichiers de configuration de l'environnement de l'usager, seront abordés plus loin.

A.1.8 Débranchement

Le débranchement est le processus par lequel un usager quitte le système. La procédure de débranchement est simple. Il s'agit de taper "logout" ou CTRL-d. À l'intérieur du C-Shell, on peut aussi taper "logout".

Exemple :

```
$ logout
```

A.2 Forme générale des commandes

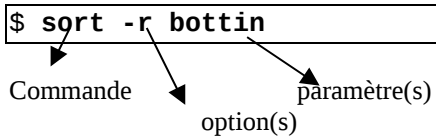
Les commandes, appelées souvent "utilitaires", sont les outils de base du système UNIX.

A.2.1 Syntaxe

Une commande est composée de trois types d'informations: le nom de la commande, les options et les paramètres.

Exemple:

```
$ sort -r bottin
```



A.2.2 Nom

Le nom de la commande indique au système quelle commande UNIX ou quel autre programme il doit exécuter.

Il existe deux types de commandes. Les commandes internes sont des commandes du shell. Les commandes externes correspondent à des programmes conservés dans /bin, /usr/bin ou dans un autre répertoire. Dans l'exemple précédent, "sort" est le nom d'une commande UNIX qui effectue le tri du contenu d'un fichier.

A.2.3 Options

Les options d'une commande indique à celle-ci que l'usager lui demande d'opérer d'une façon différente de la manière habituelle. Certaines commandes autorisent des options, d'autres non.

Dans l'exemple, l'option "-r" (reverse) demande à sort de faire un tri inversé (ordre alphabétique descendant).

A.2.4 Paramètres

Les paramètres d'une commande indiquent généralement les objets manipulés par celle-ci. Il s'agit le plus souvent de fichiers.

Dans l'exemple, on demande à sort de trier en ordre inverse le fichier "bottin".

A.3 Aide

Les commandes UNIX sont si nombreuses (plus de 200) qu'il est souvent nécessaire de consulter la documentation sur l'usage correcte de celles-ci.

Il n'est heureusement pas nécessaire de parcourir la documentation écrite du système puisqu'une aide en direct (on-line help) est généralement disponible.

La commande man (reference manual) fournit une information relativement complète sur n'importe quelle commande UNIX.

Syntaxe:

man *[section] nom_de_commande*

A.3.1 Sections du manuel

La "section" de la syntaxe est généralement une chaîne de caractères ou un nombre correspondant à une section du manuel de référence d'un système UNIX.

Exemple :

Le tableau ci-dessous indique le numéro des sections de manuel ainsi que le type de pages qu'elles contiennent.

1 Programmes exécutables ou commandes de l'interpréteur de commandes (shell) ;
2 Appels système (Fonctions fournies par le noyau) ;
3 Appels de bibliothèque (fonctions fournies par les bibliothèques des programmes) ;
4 Fichiers spéciaux (situés généralement dans /dev) ;
5 Formats des fichiers et conventions. Par exemple /etc/passwd ;
6 Jeux ;
7 Divers (y compris les macro-paquets et les conventions). Par exemple, man(7), groff(7) ;
8 Commandes de gestion du système (généralement réservées au super-utilisateur) ;
9 Sous-programmes du noyau [hors standard].

A.3.2 Utilisation

Lorsque la section n'est pas spécifiée par l'usager, man s'arrête à la première section qui contient la commande recherchée. Par exemple, "man tty" ou "man C tty" affiche des informations sur la commande tty au niveau de son utilisation par l'usager, tandis que "man HW tty" affiche des informations sur les "drivers" de terminaux.

Exemple :

```
$ man tty

      TTY(1)                 USER COMMAND STTY(1)

      NAME
      tty - display the name of the terminal

      SYNOPSIS
      tty [ -s ]

      DESCRIPTION
      tty prints the pathname of the user's terminal unless the -s
(silent)    option is given. In either case, the exit value is zero if
the standard      input is a terminal, and one if it is not.

      OPTIONS
            -s Silent. Does not print the pathname of the user's
terminal.

      Sun Release 4.1   Last change: 9 September 1987
      1
```

Certains systèmes comme XENIX n'offrent pas la commande man. Celle-ci est remplacée par ***help*** qui fournit une information beaucoup plus condensée.

Syntaxe:

help *nom_de_commande*

A.4 Fichiers et répertoires d'un système UNIX

A.4.1 Types de fichiers

Le système de fichiers UNIX comprend trois types de fichiers: fichiers ordinaires, répertoires et fichiers spéciaux.

A.4.2 Fichiers ordinaires

Un fichier ordinaire peut contenir n'importe quoi: un texte comme celui-ci, un programme (compilé ou non), des données, une image numérisée, etc...

Exemples:

/u/robert/data	un fichier de données
/etc/passwd	le fichier de mots de passe du système
/usr/local/prog/main.c	un programme c
/bin/clear	un fichier de commandes

A.4.3 Répertoires

Un répertoire est un fichier particulier qui contient des références à d'autres fichiers (de n'importe quel type), permettant à l'usager d'organiser ses informations de manière logique et structurée.

Exemples:

/	le répertoire racine (root directory)
/dev	le répertoire des fichiers spéciaux
/home/robert	le répertoire de l'usager robert

Le système n'impose aucune structure particulière à un fichier ordinaire qu'il considère simplement comme une suite d'octets. L'usager ou le programme d'application peut l'interpréter comme bon lui semble.

Il en va autrement des répertoires qui eux ont une structure fixe. Un répertoire est en réalité une table ou chaque nom de fichier est associé à un numéro de inode. Le inode est un descripteur de fichier (ce concept sera abordé plus loin).

A.4.4 Fichiers spéciaux

Un fichier spécial est associé à une composante matérielle du système. Il permet principalement de manipuler les périphériques à l'aide des mêmes commandes que celles utilisées pour les fichiers ordinaires.

Exemples:

/dev/mem	la mémoire principale du système
/dev/tty02	un terminal
/dev/lp1	une imprimante
/dev/fd0	la première unité de disquette

A.4.5 Noms de fichiers

Tout fichier dans le système doit avoir un nom. Ce nom obéit à certaines règles et certaines conventions qui sont présentées ici.

Plus loin sont présentés aussi les caractères spéciaux (wildcards) qui permettent de nommer un fichier ou un groupe de fichiers.

A.4.5.1 Règles

Un nom de fichier peut être formé à l'aide de n'importe quels caractères disponibles sur le clavier. Sauf:

Espace, F1 à F12, Insert, Home, Delete, end, PageUp, PageDown, Esc

Dans la plupart des versions de Linux, ce nom est limité à 255 caractères.

ATTENTION

Linux fait une distinction entre caractères majuscules et minuscules.
"FICHIER", "Fichier" et "fichier" sont des noms différents.

A.4.5.2 Conventions

Bien que tous les caractères soient permis dans les noms de fichiers, il est fortement suggéré de se limiter aux lettres et aux chiffres, ainsi qu'au point (.), au trait d'union (-) et au souligné (_).

Bien que moins répandue que dans d'autres systèmes, l'utilisation des suffixes est quand même pratiquée dans Linux. L'extension n'a cependant aucune signification pour le système.

Exemples:

prog.c	programme source en C
a.exe	programme exécutable
passwd.old	version antérieure d'un fichier

A.4.5.3 Caractères spéciaux

Certains caractères peuvent être employés pour former des noms génériques servant à désigner un ou plusieurs fichiers. Ces caractères sont l'astérisque (*), le point d'interrogation (?) et les crochets ([ ]).

A.4.5.4 L'astérisque (*)

Représente zéro caractère ou plus. N'importe quel(s) caractère(s).

Exemples:

*t	t, tt, rapport, produit
pr*t	present, produit

A.4.5.5 Le point d'interrogation (?)

Représente un et un seul caractère. N'importe quel caractère.

Exemples:

/dev/tty?	/dev/tty0, /dev/tty1
test??	tester, test92

A.4.5.6 Les crochets ([ ])

Permettent la sélection d'un caractère à l'intérieur d'une séquence donnée.

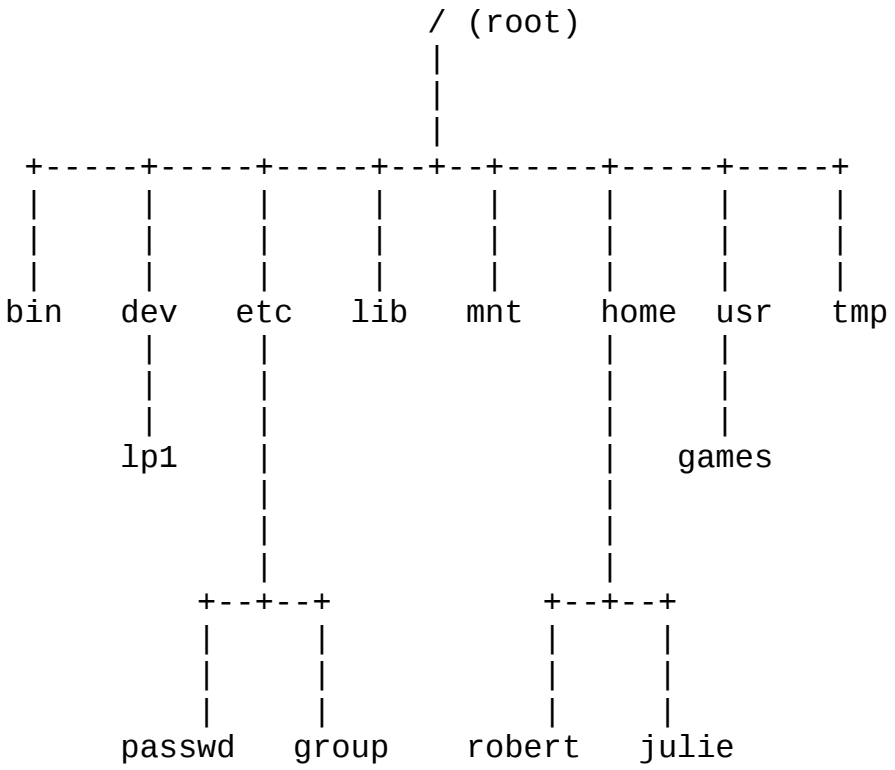
Examples:

[a-e]	a, b, c, d, e
[abcde]	a, b, c, d, e
[!f-z]	tout les caractères possibles sauf f, g, ... y, z

A.4.6 Arborescence de répertoires

A.4.6.1 Structure type

La figure suivante représente une arborescence typique:



Voici le standard des noms et signification des répertoires dans un système Linux. Ce standard suit la norme telle que décrite dans le FSH (File System Hierarchy) qui est disponible à l'adresse suivante :
http://www.pathname.com/fhs/pub/fhs-2.3.html#THEFILESYSTEM

On y indique, entre autre, la signification de chaque répertoire et ce qu'il devrait contenir au minimum.

A.5 Commandes du système

Le reste de cette section est consacré à des commandes UNIX d'usage courant. Les commandes de gestion des fichiers et des répertoires sont présentées à la section 4.

A.5.1 Commande sans paramètres

A.5.1.1 clear

La commande clear efface le contenu de l'écran et ramène le curseur dans le coin supérieur gauche.

Exemple:

```
$ clear
```

A.5.1.2 date

La commande date affiche la date et l'heure courante.

Exemple:

```
$ date
Sun Feb 11 18:42:05 EST 1998
$
```

La sortie se divise en 6 champs: le jour de la semaine, le mois de l'année, la date en format de 24 heures, le fuseau horaire et l'année.

EST veut dire "**E**astern **S**tandard **T**ime" (Heure normale de l'est)

A.5.1.3 id

La commande id affiche le nom de l'usager, son numéro d'identification, le nom de son groupe et le numéro d'identification de son groupe.

Exemple:

```
$ id
uid=299(mailhotp) gui=60(prof)
$
```

A.5.1.4 lock

La commande lock permet à l'usager de vérouiller son terminal afin d'empêcher quiconque de l'utiliser.

Exemple:

```
$ lock
Password:
Re-enter password:
terminal locked by mailhotp 0 minutes ago
```

Le mot de passe est n'importe quelle chaíne de caractère choisie immédiatement par l'usager et non le mot de passe qu'il utilise lors du branchement au système. Pour accéder de nouveau à son terminal il suffit de taper le mot de passe choisi.

A.5.1.5 logout

Cette commande permet à un usager de se débrancher du système.

A.5.1.6 passwd

La commande passwd permet à l'usager de changer en tout temps la valeur de son mot de passe.

Exemple:

```
$ passwd
Old password:
New password:
Re-enter password:
$
```

En demandant la valeur de l'ancien mot de passe, la commande évite que quelqu'un puisse changer le mot de passe d'un usager qui aurait quitté son poste de travail sans se débrancher du système.

A.5.1.7 tty

La commande tty affiche le nom du terminal par lequel l'usager est relié au système.

Exemple:

```
$ tty
/dev/tty1
$
```

A.5.1.8 who

La commande who affiche la liste des usagers actuellement branchés au système.

Exemple:

```
$ who
mailhotp    ttyp03      Mar 11 17:42
root        tty01       Mar 11 18:01
$
```

Chaque ligne de la sortie correspond à un usager branché au système. Les trois champs correspondent au nom de l'usager, au nom de son terminal ainsi qu'à la date et à l'heure de son branchement.

A.5.1.9 whoami

La commande **whoami** permet d'afficher le nom de l'usager (username) que vous utilisez actuellement. Ceci peut être utile lorsque vous vous êtes branché sous plusieurs noms d'usager et que vous voulez connaître sous quel nom d'usager vous êtes branché présentement.

```
$ whoami
chassest
$
```

A.5.2 Commande avec paramètres

A.5.2.1 cal

La commande cal affiche le calendrier de n'importe quel mois ou de n'importe quelle année de l'an 1 à l'an 9999.

Syntaxe:

cal *[[mois] année]*

En présence de deux paramètres, la commande affiche le calendrier du mois spécifié de l'année spécifiée.

Exemple:

```
$ cal 2 1992
   February 1992
 S  M Tu  W Th  F  S
                   1
 2  3  4  5  6  7  8
 9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
$
```

Dans l'exemple précédent le "2" indiquant le mois pourrait être remplacé par une chaíne de caractères comme "feb". Toute chaíne qui ne présente pas d'ambiguïté est acceptable.

En présence d'un seul paramètre, la commande réagit différemment selon qu'il s'agit d'un nombre ou d'une chaíne de caractères. Si le paramètre est un nombre, cal le considère comme une année dont elle affiche le calendrier complet. Si le paramètre est une chaíne, cal le considère comme le mois pour lequel l'usager veut un calendrier.

Utilisée sans paramètre, la commande affiche les calendriers du mois précédent, du mois courant et du mois suivant.

A.5.2.2 echo

La commande echo affiche une chaíne de caractères à l'écran.

Exemple:

```
$ echo Bonjour la visite!
Bonjour la visite!
$
```

Cette commande a d'autres usages qui seront présentés dans la section sur les scripts.

A.5.2.3 finger

La commande finger affiche une série de renseignements sur l'usager dont le nom est passé en paramètre.

Exemple:

```
$ finger mailhotp
Login name: mailhotp          In real life: Pierre Mailhot
Directory: /home/mailhotp     Shell: /usr/local/bin/tcsh
On since Feb  1 10:15:13 on ttyp1 from alpha.CC.UMontre
No unread mail
Project: Analyste-Programmeur au laboratoire de VLSI
Plan:
To seek out new bugs and new programs.
To boldly go where no computer scientist has gone before...
$
```

Les renseignements affichés sont le nom d'usager, le nom véritable, le répertoire maison, le shell de départ, la date du dernier branchement, l'état de la boíte aux lettres et quelques renseignements personnels (fichiers .project et .plan).

A.5.3 Commandes reliées à la gestion du disque et des fichiers

A.5.3.1 cat

Cette commande permet d'afficher sur la sortie standard (habituellement l'écran) le contenu d'un ou plusieurs fichiers donnés en paramètres

Exemple:

```
$ cat fruits.unix
Banane
Kiwi
Orange
Melon
Pomme
Fraise
$
```

Le contenu du fichier **fruits.unix** est affiché à l'écran.

Exemple de concaténation de fichiers:

```
$ cat > dicton
Le chat parti,
les souris dansent.
CTRL-d
$ cat proverbe dicton > mon_texte
$ cat mon_texte
Les chiens aboient,
la caravane passe.
```

```
Le chat parti,
les souris dansent.
$
```

Le " > " est un symbole de redirection de la sortie. Le concept de redirection des entrées-sorties est présenté dans une autre section.

A.5.3.2 cd

Cette commande permet de se déplacer dans l'arborescence du système de fichiers. Elle peut être utilisé de 3 façons:

- cd permet de revenir au répertoire "maison" directement
- cd ~ revient au répertoire "maison"
- cd nom_du_répertoire change au nom_du_répertoire

Exemple:

```
$ cd jeux
$ pwd
/u/mailhotp/jeux
$ cd golf
$ pwd
/u/mailhotp/jeux/golf
$ cd
$ pwd
/u/mailhotp
$
```

A.5.3.3 chgrp

Syntaxe:

chgrp <*groupe*> <*fichier*>

Cette commande change le nom du groupe auquel les protections du niveau groupe du ou des fichiers s'appliquent. Seul le propriétaire du fichier peut exécuter cette commande.

Exemple:

```
$ ls -l  chap4
-rw-------   1   jmr   usagers  46009  Jan 28 12:20   chap4
$ chgrp  sys  chap4
$ ls -l chap4
-rw-------   1   jmr   sys      46009  Jan 28 12:20   chap4
```

A.5.3.4 chmod

Syntaxe:

chmod *<mode> <fichier(s)>*

Cette commande modifie les attributs de protection des fichiers ou répertoire. Les paramètres sont définies ci-dessous.

Droits d'accès

Trois catégories d'usagers:

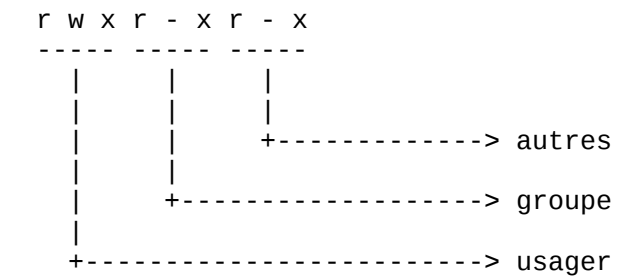
usager	l'usager propriétaire du fichier, celui qui l'a créé et qui décide des droits d'accès
groupe	le groupe auquel appartenait le propriétaire du fichier au moment de sa création
autres	les autres usagers du système

Trois types d'accès (pour tous les types de fichiers):

read	lecture
write	écriture
execute	exécution

Le second champ de la sortie de la commande "ls -l" indique les droits d'accès au fichier pour chaque catégories d'usager.

Exemple:



code	type d'accès
r	lecture
w	écriture
x	exécution
-	aucun accès

La commande chmod permet la modification des droits d'accès à un fichier. Cette commande peut être utilisée selon un mode symbolique ou absolu.
Mode symbolique

Syntaxe:

chmod *[qui] opérateur droits fichier(s)*

où on a:

qui	u	propriétaire
	g	groupe
	o	autres usagers
	a	tous les usagers (valeur de défaut)
opérateur	+	ajoute des droits
	-	enlève des droits
	=	donne des droits spécifiques
droits	r	lecture
	w	écriture
	x	exécution

Exemples:

`chmod u+w tp2.c`	*Donne au propriétaire de tp2.c le droit d'écrire dans le fichier.*
`chmod o-rw tp2.c`	*Enlève au public les droits de lecture et d'écriture du fichier tp2.c.*
`chmod u+w,o-rw tp2.c`	*Combine les deux commandes précédentes.*
`chmod a+x tp2.c`	*Donne à tout le monde le droit d'exécuter le fichier tp2.c.*
`chmod +x tp2.c`	*Est l'équivalent de la commande précédente.*
`chmod g=rx tp2.c`	*Donne au groupe les droits de lecture et d'exécution du fichier tp2.c et lui enlève (si existant) le droit d'écrire dans le fichier.*

Mode absolu

Syntaxe:

chmod *mode fichier(s)*

où le mode se calcule ainsi:

	accès		value	cat égorie
	lecture		4 0 0	propriétaire
+	écriture	2 0 0	propriétaire	
+	exécution		1 0 0	propriétaire
+	lecture		0 4 0	groupe
+	écriture	0 2 0	groupe	
+	exécution		0 1 0	groupe
+	lecture		0 0 4	autres
+	écriture	0 0 2	autres	
+	exécution		0 0 1	autres
=			mode	

Exemples:

`chmod 755 tp2.c`	*Donne tous les droits au propriétaire, donne les droits de lecture et d'éxécution au groupe et au public.*
`chmod 751 tp2.c`	*Donne tous les droits au propriétaire, donne les droits de lecture et d'exécution au groupe et le droit d'exécution au public.*
`chmod 700 tp2.c`	*Donne tous les droits au propriétaire et aucun aux autres catégories d'usagers.*
`chmod 640 tp2.c`	*Donne les droits de lecture et d'écriture au propriétaire et le droit de lecture au groupe.*

A.5.3.5 chown

Syntaxe:

```
chown  <nom> <fichier(s)>
```

Cette commande permet de rendre l'utilisateur d'identification <*nom*> propriétaire des différents fichiers énumérés par le paramètre <*fichier(s)*>.

Exemple:

```
$ ls -l  exemples/toto.c
-rw-rw-r--        1        chassest usagers  84        Jan 27 19:09       exemples/toto.c
$ chown  root       exemples/toto.c
$ ls -l  exemples/toto.c
-rw-rw-r--        1        root               usagers  84        Jan 27 19:09       exemples/toto.c
```

A.5.3.6 cmp

Syntaxe:

```
cmp  [options] <fichier1>  <fichier2>
```

Cette commande permet de comparer les deux fichiers <*fichier1*> et <*fichier2*> . Si les fichiers sont différents, le numéro de la première ligne contenant une différence est affiché ainsi que le numéro (dans le fichier) du caractère différent.

[options]:

- -l Toutes les différences sont signalées par le numéro du caractère en décimal et sa valeur octale.
- -s La commande renvoit un code de retour parmi les suivants:
  - 0: fichiers identiques
  - 1: fichiers différents
  - 2: en cas d'erreur (fichier inexistant par exemple)

Exemple:

```
$ cat  f1
Kiwi
Banane
Fraise
Melon
$ cat f2
Kiwi
Banane
Framboise
Melon
$ cmp f1  f2
f1  f2  differ: char 16, line 3
```

A.5.3.7 cp

La commande cp (copy) sert à copier un fichier dans un autre ou à copier un ou plusieurs fichiers dans un répertoire.

Syntaxe:

cp *fichier1 fichier2*
cp *fichier(s) répertoire*

ATTENTION: Si fichier2 existe déjà il sera remplacé sans avertissement! Exemple (dans un fichier):

```
$ ls
test.c
$ cp test.c bug.c
$ ls
bug.c
test.c
$
```

Exemple (dans un répertoire):

```
$ mkdir rep
$ cp test.c bug.c rep
$ ls rep
bug.c
test.c
$
```

A.5.3.8 cut

La commande cut sert à découper les lignes d'un fichier en champs et d'extraire dans ces lignes des champs particuliers.

Syntaxe:

cut *[options]* <*fichier*>

où *[options]* peut prendre les formes suivantes:

-c <liste> définit la position dans la ligne des caractères à supprimer.
-d <caractère> est le délimiteur de champ (par défaut, il s'agit du caractère <tab>)
-f <liste> permet de garder les champs mentionnés dans le paramètre <liste>

Exemple:

```
$ cat  departement.college
Francais:Viau:S-402
Philosophie:Bleau:S-503
Informatique:Boileau:F-309
Arts:Kiefer:S-304
$ cut -f2-3 -d: departement.college
Viau:S-402
Philosophie:Bleau:S-503
Boileau:F-309
Kiefer:S-304
```

```
$
```

Ainsi, le premier paramètre -f2-3 spécifie que seul les colonnes 2 et 3 seront conservées.
Le deuxième paramètre -d: spécifie que le délimiteur entre les champs est le caractère ":".

A.5.3.9 df

Donne le nombre de blocs (de 1024 octets) et de inodes libres pour les différents disques logiques montés.

Exemple:

```
$ df
/dev/hda1:      4612 blocs      1289 inodes
/dev/hda2:      1354 blocs      1809 inodes
$
```

A.5.3.10 du

Donne le nombre de blocs utilisés par chaque fichier ou répertoire indiqué

```
du <fichier>
```

Exemple:

```
$ du
1       ./exemples/tp1
9       ./exemples
520     .
$
```

Ainsi, le répertoire exemples/tp1 prends 1 bloc soit 1024 octets
Le répertoire **exemples** en prends 9 * 1024 = 9216 octets au total
Le répertoire courant **.** prends 520 * 1024 = 532480 octets

A.5.3.11 file

Commande qui permet de déterminer le type de fichier correspondant au paramètre.

Syntaxe:

```
file <fichier>
```

Exemple:

```
$ pwd
/usr/jmr/livre
$ file .
.: directory
$ file c/prog1.c
c/prog1.c:  c program text
$ file c/prog1
c/prog1:  executable file
```

A.5.3.12 find

Cette commande permet de rechercher récursivement, à partir du répertoire courant, toutes les occurances du "pattern" de fichier tapé en paramètre.

Syntaxe:

```
find [options] <"pattern">
```

où options:

-iname recherche le "pattern" sans porter attention aux majuscules ou au minuscules.

Exemple:

```
$ pwd
/prof
$ find -iname "*.cpp"
/prof/chassest/essai1.cpp
/prof/chassest/essai2.cpp
/prof/boileauf/mtr86.cpp
/prof/jacques/tp1/exemple/tp1.cpp
```

A.5.3.13 grep

Cette commande recherche dans le fichier spécifié en paramètre, une chaîne de caractère donné en paramètre.

Syntaxe:

```
grep  [options] <"pattern"> <fichier>
```

où options:

- -v: seules les lignes ne contenant pas le motif sont écrites.
- -c: seul le nombre de lignes satisfaisantes est écrit.
- -l: seuls les noms de fichiers contenant les lignes recherchées sont affichés.
- -n: chaque ligne est précédée de son numéro dans le fichier qui la contient.

Exemple:

```
$ find -n "if" essai.cpp
22: if (n == 2)
32: if (strcmp("Banane", Chaine))
38: else if (Hauteur >= 18)
```

A.5.3.14 head

Donne les n premières lignes du fichier passé en paramètre.

Syntaxe:

```
head  -n  <fichier>
```

Exemple:

```
$ head -2  fruits
Kiwi
Banane
```

A.5.3.15 ln

Cette commande crée un nouveau lien <fichier2> qui pointe au fichier <fichier1>. Il n'y a pas recopie du contenu du fichier, ni création d'un nouvel inode mais simplement augmentation du compteur des références.

Syntaxe:

ln [option] <*fichier*> <*fichier2*>

où option:

-s crée un lien symbolique.

Exemple:

```
$ ls -l fruits
-rw-r--r--      1       chassest prof   9288    Nov 12 13:37            fruits
$ ln -s punch fruits
$ ls -l
-rw-r--r--      2       chassest prof   9288    Nov 12 13:37            fruits
-rw-r--r--      1       chassest prof   4,2     Nov 12 13:37            punch -> fruits
```

Ainsi, le nombre de lien est augmenté de 1.

A.5.3.16 ls

La commande ls (list) affiche le contenu du répertoire dont le nom est passé en paramètre. Si aucun répertoire n'est spécifié, ls donne la liste des fichiers du répertoire courant.

Syntaxe:

ls *[options] [répertoires et/ou fichiers]*

Exemple:

```
$ ls
bin
dead.letter
jeux
bilan.91
$
```

La commande ls offre une foule d'options dont les principales sont énumérées ici:

-a (all) affiche le nom de tous les fichiers (y compris les fichiers cachés)

-F ajoute " / " aux noms de répertoires et " * " aux noms de fichiers exécutables

-l (long) affiche la plupart des attributs

-R (Recursive) affiche récursivement le contenu de tous les sous-répertoires

-r (reverse) affiche les noms dans l'ordre inverse

-s (size) affiche la taille en blocs de 512 octets

-t (time) affiche d'après la date de la dernière modification

-i (inode) affiche le numéro de inode

Lorsqu'un nom de fichier est passé en paramètre, ls permet de vérifier directement la présence de celui-ci.

Exemple:

```
$ ls dead.letter
dead.letter
$ rm dead.letter
$ ls dead.letter
$ /bin/ls:file not found
```

A.5.3.17 mkdir

La commande mkdir (make directory) crée un ou plusieurs répertoires.

Syntaxe:

mkdir *[-p] répertoire(s)*

Exemple:

```
$ mkdir alpha beta
$ cd alpha
$ pwd
/u/mailhotp/alpha
$ mkdir gamma
$ cd gamma
$ pwd
/u/mailhotp/alpha/gamma
$
```

L'option -p permet de créer plusieurs niveaux de répertoires en une seule commande.

Exemple:

```
$ mkdir -p alpha/gamma beta
$ cd alpha/gamma
$ pwd
/u/mailhotp/alpha/gamma
$
```

A.5.3.18 more

La commande cat est inadéquate pour l'affichage d'un fichier assez long étant donné qu'elle ne permet pas d'arrêter le défilement du texte à l'écran.

La commande more affiche le contenu d'un fichier en procédant un écran à la fois. A la fin de chaque "écran" de texte, more attend que l'usager presse une des touches suivantes:

SPACE pour l'affichage de l'écran suivant,

ENTER pour l'affichage de la ligne suivante,

q pour cesser l'affichage.

Syntaxe:

```
more [options] [fichier(s)]
```

Les options les plus utilisées sont les suivantes:

-c évite le défilement du texte en débutant toujours l'affichage

-n (où n est un nombre entier) fixe la taille de la fenêtre

+n (où n est un nombre entier) débute l'affichage à la n ième

A.5.3.19 mv

La commande mv (move) sert à renommer un fichier ou un répertoire ainsi qu'à déplacer un ou plusieurs fichiers d'un répertoire à l'autre.

Syntaxe:

```
mv fichier1 fichier2
mv répertoire1 répertoire2
mv fichier(s) répertoire
```

ATTENTION: Si fichier2 existe déjà il sera remplacé sans avertissement!

Exemple (fichiers):

```
$ ls
test.c
$ mv test.c newtest.c
$ ls
newtest.c
$
```

Exemple (répertoires):

```
$ mkdir rep
$ ls
newtest.c
rep
```

```
$ mv rep newrep
$ ls
newrep
newtest.c
$
```

Exemple (fichier et répertoire):

```
$ ls
newrep
newtest.c
$ mv newtest.c newrep
$ ls
newrep
$ ls newrep
newtest.c
$
```

A.5.3.20 paste

Cette commande permet de concaténer des fichiers.

Syntaxe:

paste <*fichier1*> <*fichier2*>

Exemple:

```
$cat insectes
mouche
guepe
taon
bourdon
coccinelle
$ cat caracteristiques
ne pique pas
peut piquer
peut piquer
peut piquer
ne pique pas
$paste insectes caracteristiques
mouche ne pique pas
guepe peut piquer
taon peut piquer
bourdon peut piquer
coccinelle ne pique pas
$
```

A.5.3.21 pwd

La commande pwd (print working directory) affiche le nom du répertoire courant. Le répertoire courant est parfois appelé répertoire de travail ou encore répertoire actif.

Syntaxe:

```
pwd
```

Exemple:

```
$ pwd
/u/mailhotp
$
```

A.5.3.22 rm

La commande rm (remove) détruit un ou plusieurs fichiers. Elle permet également de détruire un répertoire avec son contenu.

Syntaxe:

```
rm [options] fichier(s) ou répertoire(s)
```

Les options offertes sont les suivantes:

-i demande une confirmation avant de procéder
-r détruit un répertoire avec tout son contenu (fichiers et répertoires)

Exemple (fichier):

```
$ ls
un_fichier
$ rm -i un_fichier
rm: remove un_fichier? y
$ ls
$
```

Dans l'exemple précédent, n'importe quelle réponse autre que "y" laisserait le fichier intact.

Exemple (répertoire):

```
$ mkdir -p bonjour/la/visite
$ ls
bonjour
$ rm -r bonjour
$ ls
$
```

A.5.3.23 rmdir

La commande rmdir (remove directory) détruit un ou plusieurs répertoires.

Syntaxe:

rmdir *répertoire(s)*

Exemple:

```
$ rmdir junk
$ rmdir alpha/gamma alpha beta
$
```

ATTENTION: Un répertoire qui contient encore des fichier ne peut être détruit avec rmdir (voir rm -r).

A.5.3.24 sort

Cette commande trie les lignes d'un ou plusieurs fichiers en ordre alphabétique.

Syntaxe:

sort [options] <*fichier1*> <*fichier2*> ...

Exemple:

```
$ cat fruits
Banane
Kiwi
Melon
Fraise
Orange
$ sort fruits
Banane
Fraise
Kiwi
Melon
Orange
```

où options:

-f: aucune différence entre majuscules et minuscules.
-r: trie en ordre alphabétique inverse.

A.5.3.25 split

Cette commande permet d éclater un fichier en fichier de n lignes

Syntaxe:

```
split <fichier>
```

Exemple:

```
$ split -5 fruits
$
```

A.5.3.26 tail

Affiche les n dernières lignes du fichier passé en paramètre (commande contraire de **head**)

Syntaxe:

```
tail [option] <fichier>
```

où option:

-n indique qu'il faut faire afficher les n dernières lignes du fichier.

Exemple:

```
$ cat fruits
Banane
Orange
Melon
Pomme
Guava
$ tail -3 fruits
Guava
Pomme
Melon
$
```

A.5.3.27 tee

Cette commande permet de voir les informations s'afficher sur l'écran lorsque un filtre est utilisé.

Syntaxe:

```
tee <fichier>
```

Exemple:
On veut envoyer le contenu d'un répertoire (commande ls) dans un fichier ls.txt et en même temps voir l'information s'afficher à l'écran.

```
$ ls -al | tee ls.txt
-rwxr--r--  1     chassest    prof  236   Jan 18 12:06      allo.txt
-rwx------  1     chassest    prof  14453 Nov 12 19:02      examen1
$ cat ls.txt
-rwxr--r--  1     chassest    prof  236   Jan 18 12:06      allo.txt
-rwx------  1     chassest    prof  14453 Nov 12 19:02      examen1
```

A.5.3.28 uniq

Les lignes adjacentes du fichier sont comparées.. Si une ligne est identique à ses suivantes, seul un exemplaire en est fourni.

Syntaxe:

```
uniq <fichier>
```

Exemple:

```
$ cat fruits
Pomme
Banane
Banane
Pomme
Melon
Melon
Yannis
$ uniq fruits
Pomme
Banane
Pomme
Melon
Yannis
```

Remarques:

Cette commande est fréquemment employé en filtre avec la commande sort.

Exemple:

```
$ sort  fruits | uniq
Banane
Melon
Pomme
Yannis
```

A.5.3.29 wc

Compte le nombre de ligne, de mots et de caractères des fichiers fournis en paramètre.

Syntaxe:

```
wc [options] <fichier>
```

où options:

-l: nombre de lignes
-w: nombre de mots
-c: nombre de caractères

Exemple:

```
$ wc -w valentin.txt
239
```

Il y a donc 239 mots présents dans le fichier **valentin.txt**

A.6 Commandes reliées a la gestion des processus

A.6.1 Caractères ||

Ces caractères permettent d'exécuter une commande selon le non-succès d'une autre commande.

Syntaxe:

```
commande1 || commande2
```

Exemple:

```
$ cat fichier1.cpp || cp fichier1.cpp /tp1/fichier1.cpp
```

A.6.2 Caractères &&

Ces caractères permettent d'exécuter une commande selon le succès d'une autre commande.

Syntaxe:

```
commande1 && commande2
```

Exemple:

```
$ cp fichier1.cpp tp1/fichier1.cpp && cd tp1
```

A.6.3 Exécution de plusieurs commandes consécutives: caractère ;

Il est possible d'exécuter plusieurs commandes les unes à la suite des autres en les séparant par un ";".

Syntaxe:

```
commande1;commande2;commande3;commmande4;...
```

Exemple:

```
$ cd tp1;cp essai.cpp ..;rm -r tp1
```

A.6.4 Exécution de processus en arrière-plan (background)

Il est possible de lancer une commande et de la faire exécuter en arrière-plan (background). Celle-ci s'exécutera aussitôt et l'usager reviendra à l'indicatif (prompt) pendant que sa commande s'exécutera.

Syntaxe:

commande **&**

Exemple:

```
$ find -iname "x*.*" &
[404]
$
```

Le système répond en donnant le numéro du processus qui correspond à la commande "**find -iname "x*.*" "** et vous redonne le "prompt" prêt à exécuter une autre commande pendant que la première s'exécute en arrière-plan.

A.6.5 jobs

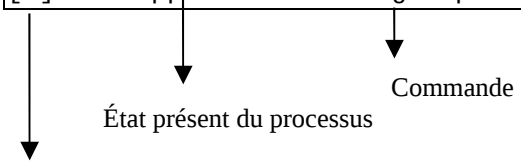
Cette commande affiche l'état des processus présentement en activité.

Syntaxe:

jobs

Exemple:

```
$ jobs
[1] + Stopped     find -iname "*.x"
[2] - Stopped     emacs group
```



A.6.6 bg

Cette commande permet d'ammener un processus en arrière-plan.

Syntaxe:

```
bg %<numéro_de_processus>
```

Exemple:

```
$ jobs
[1] + Stopped  emacs  passwd
[2] - Swapped     find -iname essai.cpp
[3] - Stopped     find -iname "fruits*" | grep "Orange"
$ fg %1   ————————▶     Emacs revient en avant-plan
$ ctrl-Z  ————————▶     Emacs retourne en arrière-plan
$ fg %3   ————————▶     La commande continue son exécution
```

A.6.7 kill

Cette commande "tue" un processus et l'enlève du système.

Syntaxe:

```
kill -9 <numéro du processus>
```

Exemple:

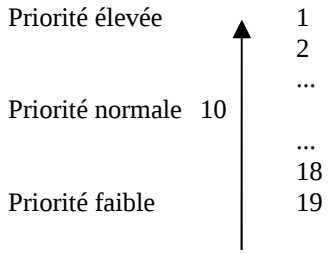
```
$ ps -u

$ kill -9 numéro
```

Le processus de numéro ... est tué. Il n'existe plus dans le système.
Ce numéro est obtenu grâce à la commande « ps -u » telle que tapée précédemment.

A.6.8 nice

Exécute une commande avec une priorité donné en paramètre. Par défaut la priorité est de 10.

Le tableau suivant indique l'importance du nombre donné par rapport à sa priorité dans le système:



Syntaxe:

```
nice [nombre] <commande>
```

Exemple:

```
$ nice 15 g++ -o essai essai.cpp &
[1269]
$
```

Remarques:

Nice est la commande idéale pour baisser la priorité des processus qui risquent de prendre beaucoup de temps CPU. En exécutant la commande en arrière-plan (&), vous pouvez ainsi vous débrancher du système et votre compilation se continuera quand même.

A.7 Commandes fréquemment utilisées dans des filtres

Voici une liste des commandes les plus fréquemment utilisées dans des filtres. Un filtre a la forme suivante:

commande | commande

Commande	*Description*
cut	
find	
grep	
head	
more	
paste	
sort	
split	
tail	
tee	
uniq	

A.8 Commandes reliées au développement de programme

A.8.1 Compilateurs

Cette section présente les 2 compilateurs les plus utilisés sous Unix. Il s'agit de g++ et gcc, un compilateur c++ et c standard respectivement.

A.8.1.1 Compilateur C++: g++

Ce compilateur accepte les instructions du langage C++.

Syntaxe:

```
g++ [options] <fichier objet> <fichier exécutable>
```

Exemple:

```
$ g++ -o tp1 tp1.cpp
$
```