Systèmes 1
Description des commandes

Table des matières

1 Introduction

Ce document contient les descriptions des commandes utilisées dans les cours/TD et dans les TP. Toutes les options de toutes les commandes ne sont pas systématiquement décrites. Seules les options les plus courantes sont présentées. Chaque description donne les caractéristiques de la commande (étymologie, descriptif rapide), sa ou ses syntaxes d'appel, une brève description du rôle de ses éventuelles options, des exemples d'utilisation, ses codes de retour et, éventuellement, quelques remarques. Dans les exemples, les noms de répertoires sont généralement formés de lettres majuscules.

Pour une recherche de la description d'une commande à partir de son nom, il est conseillé d'utiliser l'index qui se trouve à la fin de ce document.

2 Où sont les commandes ?

Les codes exécutables de toutes ces commandes sont situés dans les répertoires `/usr/bin` ou `/bin`. Néanmoins, un certain nombre d'entre elles (par exemple `echo`) existent également en tant que commandes internes, c'est-à-dire que leur code exécutable est également inclus dans le code de l'interpréteur de commandes lui-même. Dans ce cas, c'est la version interne qui est lancée, sauf si on écrit la désignation du fichier exécutable de la commande externe (par exemple : `/usr/bin/echo` ou `/bin/echo`). C'est ce qui explique que les commandes internes puissent différer d'un shell à l'autre, alors que les commandes externes sont les mêmes pour tous les shells.

3 À l'aide !

☞ `man`

➥*Caractéristiques :*

1. `man` vient de *manual* (« manuel »).
2. Cette commande permet d'obtenir la description d'une commande Unix à l'écran (généralement en anglais).
3. Elle peut apparaître en début de branchement.

➥*Syntaxe :*

`man` [`-`] [`-adFlrt`] [`-M` arbre] [`-T` macro] [`-s` section] nom_commande [`...`]

➥*Description :* cf. la description de la commande `man`, en tapant `man man`

➥*Exemple :* la commande `man chmod` affiche la description de la commande `chmod`

➥*Codes de retour :*

- Pas d'erreur : `0`
- Opération interrompue pour cause d'erreur : `> 0`

---

4 Manipulation de l'arborescence des fichiers

☞ `ls`

➥*Caractéristiques :*

1. `ls` vient de *list* (« lister »).

2. Cette commande permet d'afficher le contenu d'un ou de plusieurs répertoires, ou des renseignements concernant un ou plusieurs fichiers. S'il n'y a aucun paramètre, alors la commande affiche le contenu du répertoire courant. Si un nom de répertoire est passé en paramètre, alors la commande affiche son contenu. Si un nom de fichier est passé en paramètre, alors la commande indique si ce fichier est accessible.
3. Elle peut apparaître en début de branchement.

➥*Syntaxes :*

- `ls` [`-RadLCxmlnogrtucpFbqisf1AM`] [*`désignation_répertoire ...`*]
- `ls` [`-RadLCxmlnogrtucpFbqisf1AM`] [*`désignation_fichier ...`*]

➥*Description :*

- `-a` : affiche également les fichiers commençant par le caractère `.` (fichiers « cachés »).
- `-l` : affiche une première ligne indiquant la taille totale du répertoire en nombre de « blocs » puis, pour chaque sous-répertoire ou fichier accessible, ses principales caractéristiques sur une ligne : nature (répertoire ou fichier), droits d'accès, nombre de liens, nom du propriétaire, nom du groupe du propriétaire, taille en octets, date et heure de dernière modification, nom (qui est le 9e champ de la ligne). Si cette option est utilisée et qu'un fichier est passé en paramètre, alors ces informations ne sont affichées que pour ce fichier.
- Autres options : cf. la description de la commande `ls`, en tapant `man ls`

➥*Exemple :*

```
$ ls -al
total 1224
drwxr-xr-x   9 alaincrouzil  staff    512 Dec  3 14:58 .
drwxr-xr-x  13 alaincrouzil  staff    512 Oct  6 17:21 ..
drwxr-xr-x   2 alaincrouzil  staff   2048 Dec  4 01:42 TD
drwxr-xr-x   2 alaincrouzil  staff   1024 Dec  3 18:12 TP
-rw-r--r--   1 alaincrouzil  staff   12105 Dec  4 00:45 welcome.html
$
```

Dans cet exemple, les répertoires `.` et `..` désignent respectivement le répertoire courant et le répertoire père du répertoire courant. Ce sont des répertoires cachés qui apparaissent systématiquement avec l'option `-a`

➥*Codes de retour :*

- Pas d'erreur : `0`
- Opération interrompue pour cause d'erreur : `> 0`

➥*Remarque :*

La commande `ls` affiche un nom de fichier ou de sous-répertoire par ligne sur sa sortie standard. Néanmoins, lorsque l'affichage s'effectue à l'écran, c'est-à-dire lorsque la sortie standard n'est pas redirigée, les noms de fichiers et de sous-répertoires sont alignés en plusieurs colonnes. Le nombre de ces colonnes est fonction, entre autres, de la taille de la fenêtre dans laquelle la commande `ls` est interprétée.

➥*Caractéristiques :*

1. `cp` vient de *copy* (« copier »).

2. Cette commande permet de faire une copie d'un ou de plusieurs fichiers. Si on copie un seul fichier, on peut changer son nom (cf. le premier exemple ci-dessous). En revanche, si on désire copier plusieurs fichiers en une seule commande (cf. le deuxième exemple ci-dessous), on ne peut pas changer leurs noms.

➥*Syntaxes :*
- `cp [-fip] ancienne_désignation_fichier nouvelle_désignation_fichier`
- `cp [-fip] désignation_fichier [...] désignation_répertoire`
- `cp -r|-R [-fip] ancienne_désignation_répertoire [...] nouvelle_désignation_répertoire`

➥*Description :* cf. la description de la commande `cp`, en tapant `man cp`

➥*Exemples :*
- `cp PREPA/TD/td02ex01.c TP/td02ex01.txt`
  Cette commande copie le fichier `td02ex01.c`, situé dans le sous-répertoire `PREPA/TD`, dans le sous-répertoire `TP`, sous le nom `td02ex01.txt`
- `cp PREPA/TD/td02ex*.c TP`
  Cette commande copie tous les fichiers du sous-répertoire `PREPA/TD`, de nom commençant par `td02ex` et comportant l'extension `.c`, dans le sous-répertoire `TP`, sans changer leurs noms.
- `cp -r PREPA/TD/ WIKI`
  Cette commande copie le sous-répertoire `PREPA/TD` et l'ensemble de son contenu dans le sous-répertoire `WIKI`, sans changer les noms.

➥*Codes de retour :*
- Pas d'erreur : `0`
- Opération interrompue pour cause d'erreur : `> 0`

➥*Remarque :* on ne peut pas faire une copie d'un fichier dans son répertoire d'origine sans changer son nom, puisque deux fichiers frères ne peuvent pas porter le même nom.

☞ `mv`

➥*Caractéristiques :*

1. `mv` vient de *move* (« déplacer »).
2. Cette commande permet de déplacer un ou plusieurs fichiers ou répertoires dans l'arborescence. Si on déplace un seul fichier ou répertoire, on peut changer son nom (cf. le premier exemple ci-dessous), ce qui permet en particulier de renommer un fichier ou un répertoire sans le déplacer. En revanche, si on désire déplacer plusieurs fichiers ou répertoires en une seule commande, on ne peut pas changer leurs noms (cf. le deuxième exemple ci-dessous).

➥*Syntaxes :*
- `mv [-fi] ancienne_désignation nouvelle_désignation`
- `mv [-fi] désignation [...] désignation_répertoire`

➥*Description :* cf. la description de la commande `mv`, en tapant `man mv`

➥*Exemples :*
- `mv TP ../TP_2023`
  Cette commande permet de renommer `TP` en `TP_2023`, et de le remonter d'un étage dans l'arborescence.
- `mv TD/td02ex* TD/td03ex* PREPA`
  Cette commande permet de déplacer dans le sous-répertoire `PREPA`, les fichiers du sous-répertoire `TD` dont le nom commence par `td02ex` ou `td03ex`, sans changer leurs noms.

➥*Codes de retour :*
- Pas d'erreur : `0`
- Opération interrompue pour cause d'erreur : `> 0`

☞ `rm`

➥*Caractéristiques :*
1. `rm` vient de *remove* (« supprimer »).
2. Cette commande permet de supprimer un ou plusieurs fichiers passés en paramètres.

➥*Syntaxes :*
- `rm` [`-f`] [`-i`] `désignation_fichier` [`...`]
- `rm -rR` [`-f`] [`-i`] `désignation_répertoire` [`...`]

➥*Description :*
- `-f` : ne supprime les fichiers désignés que s'ils existent (cela évite d'éventuels messages d'erreur).
- `-i` : demande confirmation avant effacement.
- Autres options : cf. la description de la commande `rm`, en tapant `man rm`

➥*Exemple :*

```
rm *
```

Cette commande doit être évitée en général, car elle efface tous les fichiers non cachés du répertoire courant !

➥*Codes de retour :*
- Pas d'erreur : `0`
- Opération interrompue pour cause d'erreur : `> 0`

Attention :
L'expérience montre que beaucoup d'utilisateurs sont tentés de placer la commande `rm` en fin de branchement, comme dans l'exemple suivant :

```
ls *.bak | rm
```

Cette syntaxe est incorrecte, car la commande `rm` n'est pas un filtre. Pour supprimer les fichiers du répertoire courant comprenant l'extension `.bak`, on peut bien sûr utiliser l'une des deux syntaxes suivantes :

```
rm *.bak
rm `ls *.bak`
```

➥*Remarque :*
Pour supprimer un fichier dont le nom commence par un caractère `-`, comme par exemple `-fichier`, on peut taper la commande :

```
rm -- -fichier
```

La commande `rm -fichier` provoquerait l'affichage d'un message d'erreur, car le shell interpréterait les caractères `-fichier` comme une liste d'options de la commande `rm`. On peut également taper :

```
rm ./-fichier
```

☞ `mkdir`

➥*Caractéristiques :*

1. `mkdir` vient de *make directory* (« créer répertoire »).
2. Cette commande permet de créer un ou plusieurs répertoires, dont les noms sont passés en paramètres.

➥*Syntaxe :*

`mkdir` [`-m` `mode`] [`-p`] `désignation_répertoire` [`...`]

➥*Description :*

- `-m` `mode` : indique les droits d'accès au répertoire à créer.
- `-p` : crée tous les répertoires intermédiaires, si nécessaire.

➥*Exemples :*

- `mkdir -m go-rw REP1`
  Cette commande crée le sous-répertoire `REP1` dans le répertoire courant, en enlevant les droits en lecture et en écriture (`r` pour *read* et `w` pour *write*) à toute personne non propriétaire du compte (`g` pour *group* et `o` pour *others*).
- `mkdir -p REP2/SREP2/SSREP2`
  Cette commande crée le sous-répertoire `REP2` du répertoire courant, le sous-répertoire `SREP2` de `REP2` et le sous-répertoire `SSREP2` de `SREP2`.

➥*Codes de retour :*

- Pas d'erreur : `0`
- Opération interrompue pour cause d'erreur : `> 0`

☞ `rmdir`

➥*Caractéristiques :*

1. `rmdir` vient de *remove directory* (« supprimer répertoire »).
2. Cette commande permet d'effacer un ou plusieurs répertoires passés en paramètres. Ces répertoires doivent être vides.

➥*Syntaxe :*

`rmdir` [`-ps`] `désignation_répertoire` [`...`]

➥*Description :* cf. la description de la commande `rmdir`, en tapant `man rmdir`

➥*Exemple :*

```
rmdir ..
```

Cette commande provoque systématiquement une erreur, car le répertoire père du répertoire courant n'est jamais vide. De même, la commande `rmdir .` provoquerait l'affichage d'un message d'erreur, car on ne peut pas supprimer le répertoire dans lequel on se trouve.

➥*Codes de retour :*

- Pas d'erreur : `0`
- Opération interrompue pour cause d'erreur : `> 0`

☞ `pwd`

➥*Caractéristiques :*

1. `pwd` vient de *print working directory* (« afficher le répertoire courant »).
2. Cette commande affiche la désignation absolue du répertoire courant.
3. Elle peut apparaître en début de branchement.

➥*Syntaxe :*

```
pwd
```

➥*Description :* Pas d'option.

Exemple :

```
$ pwd
/home/ufar/SYST1
$
```

➥*Codes de retour :*

- Pas d'erreur : `0`
- Opération interrompue pour cause d'erreur : `> 0`

☞ `cd`

➥*Caractéristiques :*

1. `cd` vient de *change directory* (« changer de répertoire »).
2. Cette commande permet de changer de répertoire courant, de manière relative ou absolue.
3. Il s'agit d'une commande interne, qui peut donc fortement varier d'un shell à l'autre.

➥*Syntaxe :*

`cd` [`désignation_répertoire`]

Lorsque la commande est tapée sans paramètre, on se retrouve dans le répertoire d'accueil (*home directory*).

➥*Description :* pas d'option.

➥*Exemple :*

```
cd ../TD
```

Cette commande permet de se déplacer dans l'arborescence vers le répertoire frère du répertoire courant, de nom `TD`

➥*Codes de retour :*

- Pas d'erreur : `0`
- Opération interrompue pour cause d'erreur : `> 0`

☞ `chmod`

➥*Caractéristiques :*

1. `chmod` vient de *change mode* (« changer de mode »).
2. Cette commande permet de modifier les droits d'accès à un ou à plusieurs fichiers ou répertoires dont on est le propriétaire.

➥*Syntaxe :*

`chmod` [`-fR`] [`personnes...`]`+|-`[`permissions...`] `désignation` [`...`]

- Les `personnes` (classes d'utilisateurs) sont `u` (*user*) s'il s'agit du propriétaire du fichier ou du répertoire, `g` (*group*) s'il s'agit des membres du groupe du propriétaire, `o` (*others*) s'il s'agit des autres utilisateurs ou `a` (*all*) s'il s'agit de tout le monde.
- Le signe `+` signifie qu'on ajoute des droits d'accès, alors que `-` signifie qu'on en enlève.
- Les `permissions` sont `r` (*read*) pour la lecture, `w` (*write*) pour l'écriture et `x` (*execute*) pour l'exécution.

➥*Description :* cf. la description de la commande `chmod`, en tapant `man chmod`

➥*Exemple :*

```
chmod g+rx /users/linfg/linfg0/omar
```

Cette commande rajoute aux membres du groupe les droits en lecture et en exécution sur le fichier spécifié.

➥*Codes de retour :*

- Pas d'erreur : `0`
- Opération interrompue pour cause d'erreur : `> 0`

➥*Remarques :*

- Pour connaître les droits d'accès à un fichier, il suffit de taper la commande `ls -l`
- Pour lancer l'exécution d'un script contenu dans le fichier `nom_script.sh`, on peut aussi le rendre exécutable, puis l'appeler directement, de la manière suivante :

```
$ chmod u+x nom_script.sh
$ nom_script.sh [paramètre_script ...]
```

☞ `find`

➥*Caractéristiques :*

1. `find` signifie « trouver ».
2. Cette commande permet de rechercher les fichiers et les répertoires qui se trouvent dans les sous-arborescences dont les racines sont les répertoires dont les désignations sont passées en paramètres. Les derniers paramètres permettent de choisir les fichiers et les répertoires qui nous intéressent ainsi que de préciser ce que l'on souhaite en faire.
3. Elle peut apparaître en début de branchement.

➥*Syntaxe :*

Parmi toutes les possibilités, voici deux utilisations très classiques de la commande `find` :

`find` `désignation_répertoire` [`...`] `-name` `modèle_nom` [`-exec` `commande` `\;`]

➥*Description :*

- `-name` `modèle_nom` : sélectionne les fichiers et les répertoires dont les noms respectent le modèle `modèle_nom`. En général, ce modèle contient des métacaractères du shell qui doivent être interprétés par la commande `find`. Pour empêcher que le shell ne les interprète, il est nécessaire d'entourer `modèle_nom` de guillemets ou d'apostrophes. Si l'option `-exec` n'est pas utilisée, alors les désignations (relatives aux racines des sous-arborescences) des fichiers et des répertoires sélectionnés sont affichées (une par ligne) sur la sortie standard.
- `-exec` `commande` : les désignations des fichiers et des répertoires concernés ne sont pas affichées sur la sortie standard, mais la commande `commande` est appliquée à chacun d'entre eux. Dans cette commande, les caractères `{}` représentent la désignation de chacun des fichiers et les caractères `\;` terminent la commande.

- autre options : cf. la description de la commande `find`, en tapant `man find`

➥*Exemples :*

- `find . -name "??????"`
  Cette commande affiche les désignations relatives des sous-répertoires et des fichiers qui sont des « descendants » du répertoire courant, et dont les noms comportent six caractères exactement.
- `find $HOME -name 'core' -exec rm {} \;`
  Cette commande efface tous les fichiers de nom `core` du compte de l'utilisateur.

➥*Codes de retour :*

- Pas d'erreur : `0`
- Opération interrompue pour cause d'erreur : `> 0`

☞ dirname

➥*Caractéristiques :*

1. `dirname` vient de *directory name* (« nom du répertoire »).
2. Cette commande permet d'afficher sur la sortie standard la chaîne de caractères constituée de la désignation d'un fichier ou d'un répertoire privée du dernier caractère `/` et des caractères suivants. Si la désignation ne contient que le nom d'un fichier ou d'un répertoire, alors le caractère `.` qui désigne le répertoire courant est affiché.
3. Elle peut apparaître en début de branchement.

➥*Syntaxe :*

`dirname` désignation

➥*Description :* pas d'option.

➥*Exemples :*

-
```
$ dirname /home/ufar/SYST1/REP/fich.txt
/home/ufar/SYST1/REP
$
```

-
```
$ dirname fich.txt
.
$
```

➥*Codes de retour :*

- Pas d'erreur : `0`
- Opération interrompue pour cause d'erreur : `> 0`

☞ basename

➥*Caractéristiques :*

1. `basename` vient de *base name* (« nom de base »).
2. Cette commande permet d'afficher sur la sortie standard la chaîne de caractères constituée de la désignation d'un fichier ou d'un répertoire privée des caractères situés avant le dernier caractère `/` ainsi que d'un éventuel suffixe.
3. Elle peut apparaître en début de branchement.

➥*Syntaxe :*
`basename` désignation [suffixe]
➥*Description :* pas d'option.
➥*Exemples :*

- `$ basename /home/ufar/SYST1/REP/fich.txt`
  `fich.txt`
  `$`

- `$ basename /home/ufar/SYST1/REP/fich.txt .txt`
  `fich`
  `$`

➥*Codes de retour :*

- Pas d'erreur : `0`
- Opération interrompue pour cause d'erreur : `> 0`

---

5 Accès au contenu des fichiers

☞ `cat`

➥*Caractéristiques :*

1. `cat` vient de *concatenate* (« concaténer »).
2. Cette commande permet d'afficher le contenu de l'entrée standard ou le contenu du ou des fichiers passés en paramètres, en les concaténant.
3. Cette commande est un filtre.

➥*Syntaxe :*
`cat` [`-nbsuvet`] [désignation_fichier ...]
➥*Description :*

- `-n` : numérote les lignes, avant de les envoyer sur la sortie standard.
- `-b` : identique à `-n`, sans numérotation des lignes vides.
- Autres options : cf. la description de la commande `cat`, en tapant `man cat`

➥*Exemples :*

- `cat fich1.txt fich2.txt > fich3.txt`
  Cette commande recopie `fich1.txt` puis, à la suite, `fich2.txt`, le tout dans `fich3.txt`
- Dans l'exemple suivant, ce sont les caractères tapés sur l'entrée standard qui sont écrits dans le fichier `fich.txt` :
  `$ cat > fich.txt`
  `ligne1`
  `ligne2`
  `<Ctrl>D`
  `$`

➥*Codes de retour :*

- Pas d'erreur : `0`
- Opération interrompue pour cause d'erreur : `> 0`

☞ `more`

➥*Caractéristiques :*

1. `more` signifie « encore ».
2. Cette commande permet d'afficher le contenu de l'entrée standard (ou, s'il y a lieu, le contenu du ou des fichiers passés en paramètres, en les concaténant), page par page. Elle est une version plus interactive de la commande `cat` :
   - Pour afficher la page suivante, il faut taper sur la barre d'espace.
   - Pour afficher seulement une nouvelle ligne, il faut taper sur la touche « entrée ».
   - Pour interrompre l'affichage, il suffit de taper le caractère `q`.
3. C'est un filtre (cf. tout de même la remarque ci-dessous).

➥*Syntaxe :*

`more` [`-cdflrsuw`] [`-lignes`] [`+numéro_ligne`] [`+/modèle`] [`désignation_fichier` `...`]

➥*Description :* cf. la description de la commande `more`, en tapant `man more`

➥*Exemples :*

La commande `more ‘ls‘` affiche le contenu de tous les fichiers du répertoire courant (les sous-répertoires du répertoire courant sont ignorés). Le branchement de commandes `ls -l | more` affiche page par page les informations détaillées concernant les fichiers et les répertoires contenus dans le répertoire courant.

➥*Codes de retour :* cf. la description de la commande `more`, en tapant `man more`

➥*Remarque :* en fait, la commande `more` n'est pas un vrai filtre, dans la mesure où elle n'accepte de données par l'intermédiaire de l'entrée standard que si l'entrée standard a été redirigée (sur ce point, la commande `more` diffère de la commande `cat`).

☞ `tee`

➥*Caractéristiques :*

- `tee` désigne un « embranchement en forme de T ».
- Cette commande permet d'afficher les données reçues sur l'entrée standard simultanément sur la sortie standard et sur la liste (éventuellement vide) de fichiers passés en paramètres.
- C'est un filtre.

➥*Syntaxe :*

`tee` [`-ai`] [`désignation_fichier` `...`]

➥*Description :*

- `-a` : les données reçues sur l'entrée standard sont concaténées aux fichiers passés en paramètres.
- Autres options : cf. la description de la commande `tee`, en tapant `man tee`

➥*Exemple :*

```
ls | tee liste_fichiers.txt
```

Cette commande affiche, d'une part, le contenu du répertoire courant à l'écran et écrit, d'autre part, ce contenu dans le fichier `liste_fichiers.txt`

➥*Codes de retour :*

- Pas d'erreur : `0`
- Opération interrompue pour cause d'erreur : `> 0`

☞ `head`

➥*Caractéristiques :*

1. `head` signifie « tête ».
2. Cette commande affiche l'entrée standard, ou le ou les fichiers passés en paramètres, jusqu'à une position désignée. Sans option, elle affiche les `10` premières lignes.
3. C'est un filtre.

➥*Syntaxe :*

`head` [`-position`|`-n position`] [`désignation_fichier ...`]

➥*Description :*

- `-position` : affiche les `position` premières lignes du fichier ou de l'entrée standard.
- Autres options : cf. la description de la commande `head`, en tapant `man head`

➥*Exemple :*

```
head -n 5 fichier.c
```

Cette commande affiche les `5` premières lignes de `fichier.c`

➥*Codes de retour :*

- Pas d'erreur : `0`
- Opération interrompue pour cause d'erreur : `> 0`

☞ `tail`

➥*Caractéristiques :*

1. `tail` signifie « queue ».
2. Cette commande affiche l'entrée standard, ou le fichier passé en paramètre, à partir de la position désignée. Sans option, elle affiche les `10` dernières lignes.
3. C'est un filtre.

➥*Syntaxe :*

`tail` [`-n` [`+`] `position`|`-c` [`+`] `position`] [`désignation_fichier`]

➥*Description :*

- `-n position` : affichage des `position` dernières lignes de `désignation_fichier`
- `-n +position` : affichage de `désignation_fichier` à partir de la ligne numéro `position` (cette ligne étant comprise).
- `-c position` : affichage des `position` derniers caractères de `désignation_fichier`
- `-c +position` : affichage de `désignation_fichier` à partir du caractère numéro `position` (ce caractère étant compris).
- Autres options : cf. la description de la commande `tail`, en tapant `man tail`

➥*Exemples :*

- `tail -n +2 fic.txt`
  Cette commande affiche toutes les lignes de `fic.txt`, sauf la première.
- `tail -c 15 fichier.c`
  Cette commande affiche les `15` derniers caractères du fichier `fichier.c`

➥*Codes de retour :*

- Pas d'erreur : `0`
- Opération interrompue pour cause d'erreur : `> 0`

☞ grep

➥*Caractéristiques :*

1. `grep` vient de *global regular expressions parsing* (« analyse d'expressions régulières globales »).
2. Cette commande permet de rechercher dans un ou plusieurs fichiers passés en paramètres (ou dans chaque ligne de l'entrée standard) les lignes qui correspondent à une expression régulière et affiche ces lignes sur la sortie standard. Si plusieurs fichiers sont passés en paramètres, chaque ligne correspondant à l'expression régulière est précédée du nom du fichier la contenant, suivi d'un caractère `:`
3. Cette commande est un filtre.

➥*Syntaxe :*

`grep` [`-bchilnsvw`] `expression_régulière` [`désignation_fichier` `...`]

➥*Description :*

- `-l` : la commande n'affiche que le nom de chaque fichier où une ligne correspond (au lieu d'afficher les lignes qui correspondent), ou `<stdin>` si une ligne qui correspond est trouvée sur l'entrée standard.
- `-c` : affiche uniquement, pour chaque fichier, le nombre de lignes qui correspondent.
- `-v` : affiche les lignes qui ne correspondent pas à l'expression régulière.
- `-n` : chaque ligne qui correspond est précédée de son numéro (la première ligne a le numéro `1`), suivi d'un caractère `:`
- Autres options : cf. la description de la commande `grep`, en tapant `man grep`

➥*Exemple :* la commande `grep -c '.*' fich` affiche le nombre de lignes du fichier `fich`

➥*Codes de retour :*

- Une ligne au moins correspond : `0`
- Aucune ligne ne correspond : `1`
- Opération interrompue pour cause d'erreur : `2`

➥*Remarque :*

L'utilisation d'expressions régulières contenant des caractères de contrôle, à l'aide de la commande `grep`, pose quelques problèmes. Par exemple, la commande `grep '\tabc'` recherche, non pas les lignes de l'entrée standard contenant un caractère de tabulation suivi des trois lettres `abc`, mais les lignes contenant la séquence de quatre lettres `tabc`. Il en serait de même si l'expression régulière était écrite `"\tabc"` ou `\tabc`. Le seul moyen de rendre correcte cette recherche est de taper réellement sur la touche de tabulation du clavier, pour signifier que le premier caractère de l'expression régulière doit être un caractère de tabulation.

☞ sort

➥*Caractéristiques :*

1. `sort` signifie « trier ».
2. Cette commande lit un ou plusieurs fichiers ou l'entrée standard, et affiche les différentes lignes, triées sur un ou plusieurs champs (par défaut, sur un seul champ qui est la ligne), selon l'ordre lexicographique. Les séparateurs de deux champs sont l'espace et la tabulation.
3. C'est un filtre.

➥*Syntaxe :*

`sort` [`-cmu`] [`-o` `sortie`] [`-T` `répertoire`] [`-dfiMnr`] [`-b`] [`-t` `caractère`] [`-k` `clé`] [`+`pos_1 [`-`pos_2]] [`désignation_fichier` `...`]

➥*Description :*

- `-k clé` : permet de préciser les champs de la ligne à prendre en compte pour le tri. L'argument `clé` doit être écrit sous la forme `numéro_premier_champ[,numéro_dernier_champ]`. La partie entre crochets est optionnelle : si elle est absente, cela signifie que le tri prend en compte tous les champs à partir du champ de numéro `numéro_premier_champ`
- `-u` : affichage d'un seul exemplaire des lignes identiques.
- `-c` : si le fichier à trier est déjà trié, aucun affichage. Sinon, affichage des lignes triées, et retour d'un code égal à `1`.
- Autres options : cf. la description de la commande `sort`, en tapant `man sort`

➥*Exemple :*

Supposons que le fichier de nom `etat_civil.txt` soit structuré de la manière suivante :

```
Anne DUPONT 26 ans
Jean DUPONT 22 ans
Mathieu BARDON 23 ans
```

Voici deux exemples d'exécution de la commande `sort` :

```
$ sort -k 2,2 etat_civil.txt
Mathieu BARDON 23 ans
Anne DUPONT 26 ans
Jean DUPONT 22 ans
$ sort -k 2 etat_civil.txt
Mathieu BARDON 23 ans
Jean DUPONT 22 ans
Anne DUPONT 26 ans
$
```

➥*Codes de retour :*

- Pas d'erreur ou option `-c` sur fichier déjà trié : `0`
- Option `-c` sur fichier non trié : `1`
- Opération interrompue pour cause d'erreur : `> 1`

☞ `cut`

➥*Caractéristiques :*

1. `cut` signifie « couper ».
2. Cette commande permet de supprimer une partie des lignes d'un ou de plusieurs fichiers, ou de l'entrée standard.
3. Cette commande est un filtre.

➥*Syntaxes :*

- `cut -c liste [désignation_fichier ...]`
- `cut -f liste [-d délimiteur] [-s] [désignation_fichier ...]`

➥*Description :*

- `-c` : ne retient sur chaque ligne que les caractères situés à une position donnée par `liste`. Par exemple, l'option `-c1-50` ne garde que les `50` premiers caractères de chaque ligne.
- `-f` : ne retient qu'une liste de champs (séparés soit par des tabulations, soit par un `délimiteur` spécifié par l'option `-d`). Par exemple, l'option `-f1,3-5` conduira la commande à ne garder que les champs numéros `1`, `3`, `4` et `5` de chaque ligne.
- Autres options : cf. la description de la commande `cut`, en tapant `man cut`

➥*Exemples :*

- Soit un fichier `calepin.txt`, représentant un annuaire téléphonique, dans lequel chaque ligne apparaît sous la forme :
  `DUPONT Jean 05.61.75.18.47`
  La commande suivante :
  `cat calepin.txt | cut -f3 -d' ' | cut -f1 -d'.' | grep '05' | wc -l`
  affiche le nombre de personnes figurant dans ce calepin et résidant dans le Sud-Ouest (le numéro de téléphone de ces personnes commence par les chiffres `05`).
- Le format d'affichage de la commande `wc désignation_fichier` est en fait le suivant :
  `<espaces>nb_l<espaces>nb_m<espaces>nb_c<espaces>désignation_fichier`
  où `nb_l` désigne le nombre de lignes, `nb_m` le nombre de mots et `nb_c` le nombre de caractères de `désignation_fichier`. Quant au symbole `<espaces>`, il désigne une chaîne non vide, de longueur variable, composée du seul caractère espace. Par conséquent, si on désire afficher le nombre de mots du fichier `fichier.txt`, on peut taper l'une des trois commandes suivantes :

```
wc fichier.txt | tr -s ' ' | cut -f3 -d' '
wc -w fichier.txt | tr -s ' ' | cut -f2 -d' '
cat fichier.txt | wc -w | tr -d ' '
```

➥*Codes de retour :*

- Pas d'erreur : `0`
- Opération interrompue pour cause d'erreur : `> 0`

➥*Remarque :* on peut également écrire `-c liste`, `-f liste` et `-d délimiteur` en collant `liste` ou `délimiteur` aux options `-c`, `-f` ou `-d`

☞ `WC`

➥*Caractéristiques :*

1. `wc` vient de *word count* (« compter les mots »).
2. Cette commande permet de compter le nombre de lignes, de mots ou de caractères dans le ou les fichiers passés en paramètres, ou sur l'entrée standard.
3. Cette commande est un filtre.

➥*Syntaxe :*

`wc [-c|-m|-C] [-lw] [désignation_fichier ...]`

➥*Description :*

- Appelée sans option, la commande `wc` affiche, sur chaque ligne, le nombre de lignes, le nombre de mots, le nombre de caractères et le nom des fichiers passés en paramètres. Ces différentes valeurs sont précédées et séparées entre elles par des espaces, dont le nombre est difficile à prévoir *a priori*.
- `-l`, `-w`, `-c` : permettent de retourner, respectivement, les nombres de lignes, de mots et de caractères.
- Autres options : cf. la description de la commande `wc`, en tapant `man wc`

➥*Exemple :*

```
wc -l exam_S3_2022_2023.txt
```

Cette commande affiche le nombre de lignes du fichier `exam_S3_2022_2023.txt` suivi du nom du fichier.

➥*Codes de retour :*

- Pas d'erreur : `0`
- Opération interrompue pour cause d'erreur : `> 0`

☞ cmp

➥*Caractéristiques :*

- `cmp` vient de *compare* (« comparer »).
- Cette commande effectue la comparaison des contenus de deux fichiers, ou du contenu d'un fichier et de l'entrée standard. Appelée sans option, elle affiche sur `stdout` le numéro du premier octet qui diffère. Elle n'affiche rien si les deux fichiers sont identiques.
- C'est un filtre (avec la deuxième syntaxe uniquement).

➥*Syntaxes :*

- `cmp` [`-l`] [`-s`] `désignation_fichier`$_1$ `désignation_fichier`$_2$ [`décalage`$_1$] [`décalage`$_2$]
- `cmp` [`-l`] [`-s`] `-` `désignation_fichier`$_2$ [`décalage`$_1$] [`décalage`$_2$]
  Dans la deuxième syntaxe, le caractère `-` (qui doit précéder `désignation_fichier`$_2$) désigne l'entrée standard. Il ne faut donc pas oublier le caractère espace entre `-` et `désignation_fichier`$_2$.

➥*Description :*

- `-l` : affiche les numéros de tous les octets qui diffèrent et la valeur de la différence.
- `-s` : n'affiche aucun message, mais gère les codes de retour.

➥*Exemples :*

- `cmp ‘ls | head -1‘ ‘ls | tail -1‘`
  Cette commande compare les contenus du premier et du dernier fichiers contenus dans le répertoire courant.
- `cat fich1.txt | cmp - fich2.txt > fich3.txt`
  Cette commande a le même effet que la commande suivante :
  `cmp fich1.txt fich2.txt > fich3.txt`
  Elles écrivent toutes deux, dans le fichier `fich3.txt`, le résultat de la comparaison des contenus des fichiers `fich1.txt` et `fich2.txt`

➥*Codes de retour :*

- Fichiers identiques : `0`
- Fichiers différents : `1`
- Opération interrompue pour cause d'erreur : `> 1`

➥*Remarque :* les arguments optionnels `décalage`$_1$ et `décalage`$_2$ permettent de comparer le contenu du premier fichier, à partir de l'octet numéro `décalage`$_1$, au contenu du deuxième fichier, à partir de l'octet numéro `décalage`$_2$.

☞ tr

➥*Caractéristiques :*

1. `tr` vient de *translate* (« traduire »).
2. Cette commande permet de remplacer ou de supprimer des caractères provenant de l'entrée standard. Elle fonctionne également avec les caractères de contrôle présentés dans la description de la commande `echo`, page 25.
3. Cette commande est un filtre.

➥*Syntaxes :*

- `tr -s|-d` [`-c`] `chaîne`
- `tr` [`-cs`] `chaîne`$_1$ `chaîne`$_2$
- `tr -ds` [`-c`] `chaîne`$_1$ `chaîne`$_2$

➥*Description :*

- `-s` : élimine les répétitions des caractères composant la chaîne de caractères `chaîne`, en n'en laissant qu'un à chaque fois.
- `-d` : supprime toutes les occurrences des caractères composant la chaîne de caractères `chaîne` (ou $\texttt{chaîne}_1$)
- Autres options : cf. la description de la commande `tr`, en tapant `man tr`

➥*Exemples :*

- `tr 'abc' 'ABC'`
  Cette commande attend l'entrée de données au clavier. Si on tape `calebasse` chaque occurrence d'un `a` est remplacé par un `A`, chaque occurrence d'un `b` est remplacé par un `B` et chaque occurrence d'un `c` est remplacé par un `C`, ce qui donne dans le cas présent :
  `CAleBAsse`
- La commande `tr '\n' ' '` remplace chaque saut de ligne par un espace, parmi les caractères provenant de l'entrée standard.

➥*Codes de retour :*

- Pas d'erreur : `0`
- Opération interrompue pour cause d'erreur : `> 0`

➥*Remarques :*

- La commande `tr 'ab' 'A'` n'effectue aucun remplacement pour le caractère `b`, et la commande `tr 'a' 'AB'` ne remplace le caractère `a` que par le caractère `A`.
- La commande `tr` ne permet pas de remplacer des chaînes de caractères. Pour cela, il est possible d'utiliser la commande `sed`, page 18.

☞ sed

➥*Caractéristiques :*

1. `tr` vient de *stream editor* (« éditeur de flux »).
2. Cette commande permet d'effectuer des manipulations complexes sur des fichiers de texte qui peuvent être de grande taille. Les noms des fichiers peuvent être passés en arguments de la commande `sed`. Sans nom de fichier, `sed` travaille sur le texte provenant de l'entrée standard. Les manipulations sont décrites par des « commande d'édition ». Une commande d'édition peut être transmise à la commande `sed` sous la forme d'une chaîne de caractères placée après l'option `-e`. Plusieurs commandes peuvent être regroupées dans un fichier dont le nom doit être placé après l'option `-f`.
3. Cette commande est un filtre.

➥*Syntaxes :*

- `sed` [`-n`] `commande_edition` [`désignation_fichier` `...`]
- `sed` [`-n`] [`-e` `commande_edition`] `...` [`-f` `désignation_fichier_commandes`] `...` [`désignation_fichier` `...`]

➥*Description :*

- `-e` `commande_edition` : demande à `sed` d'appliquer la commande d'édition `commande_edition`.
- `-f` `désignation_fichier_commandes` : demande à `sed` d'appliquer les commandes d'édition qui se trouvent dans le fichier `désignation_fichier_commandes`.
- Parmi les différentes commandes d'édition disponibles, la fonction de substitution est très utile. Parmi plusieurs possibilités, la syntaxe la plus employée est la suivante :
  `s/`$\texttt{exp\_reg}_1$`/`$\texttt{exp\_reg}_2$`/`[g]
  où $\texttt{exp\_reg}_1$ est l'expression régulière qui décrit ce qui doit être remplacé et $\texttt{exp\_reg}_2$ décrit par

quoi cela doit être remplacé. Le caractère optionnel `g` à la fin de la commande signifie que toutes les occurrences doivent être remplacées. En son absence, seule la première occurrence de chaque ligne est remplacée.

- Autres options et autres commandes d'édition : cf. la description de la commande `sed`, en tapant `man sed`

➥*Exemple :* `sed -e 's/TP/TD Machine/g' sujets.txt`
Cette commande lit le fichier `sujets.txt` et affiche son contenu sur la sortie standard en remplaçant toutes les occurrences de la chaîne de caractères `TP` par la chaîne de caractères `TD Machine`.

➥*Codes de retour :*

- Pas d'erreur : `0`
- Opération interrompue pour cause d'erreur : `> 0`

---

6 Gestion des processus

☞ `ps`

➥*Caractéristiques :*

1. `ps` vient de *process status* (« situation des processus »).
2. Cette commande permet d'afficher la liste des processus en cours d'activité. Lancée sans option, elle n'affiche que les processus actifs de l'utilisateur associés à la fenêtre d'où la commande est lancée.
3. Elle peut apparaître en début de branchement.

➥*Syntaxe :*
`ps` [`-aAcdefjlLPy`] [`-g` nom] [`-n` nom] [[`-o` format] `...`] [`-p` nom] [`-s` nom] [`-t` terminaux] [`-u` utilisateurs] [`-U` nom] [`-G` nom]
Les processus sont affichés, à raison d'un par ligne, suivant un format variable. Le format par défaut comprend les quatre champs suivants :

- `PID` : numéro d'identification du processus.
- `TTY` : terminal associé au processus.
- `TIME` : temps d'exécution cumulé du processus.
- `CMD` : nom de la commande associée au processus.

➥*Description :*

- `-e` : affichage des informations sur tous les processus.
- `-f` : affichage de la description complète des processus, comportant quatre champs de plus que l'affichage par défaut, et en particulier le nom du propriétaire du processus (champ `UID`) et l'heure de lancement du processus (champ `STIME`).
- `-u` utilisateurs : affichage des informations sur tous les processus dont le propriétaire est contenu dans la liste utilisateurs.
- Autres options : cf. la description de la commande `ps`, en tapant `man ps`

➥*Exemples :*

- `ps -e | grep 'nedit' | wc -l`
  Cette commande affiche le nombre de processus en cours d'exécution associés à l'éditeur de texte `nedit`
- `ps -u "alaincrouzil johndeuf"`
  Cette commande affiche tous les processus dont le propriétaire est `alaincrouzil` ou `johndeuf` (quatre champs affichés par processus).

- `ps -u alaincrouzil -f`
  Cette commande affiche tous les processus dont le propriétaire est `alaincrouzil` (huit champs affichés par processus).

➥*Codes de retour :*
- Pas d'erreur : `0`
- Opération interrompue pour cause d'erreur : `> 0`

☞ `kill`

➥*Caractéristiques :*
1. `kill` signifie « tuer ».
2. Cette commande permet d'envoyer un signal à un ou à plusieurs processus dont on connaît les identificateurs.
3. Il s'agit d'une commande interne, qui peut donc fortement varier d'un shell à l'autre.

➥*Syntaxes :*
- `kill` [`-signal`] `identificateur_processus` [`...`]
- `kill -l`

➥*Description :*
- `-signal` : envoi du signal `signal` au processus.
- `-l` : affichage de la liste des signaux qui peuvent être envoyés à un processus.

➥*Exemple :*

```
kill -9 1345
```

Cette commande envoie le signal `9` au processus d'identificateur `1345`. Le signal `9`, qui permet de « tuer » un processus, est le plus souvent utilisé. Le signal envoyé par défaut est `15`.

➥*Codes de retour :*
- Pas d'erreur : `0`
- Opération interrompue pour cause d'erreur : `> 0`

☞ `exit`

➥*Caractéristiques :*
1. `exit` signifie « sortie ».
2. Cette commande permet de sortir d'un shell, en précisant ou non la valeur du code de retour.
3. Il s'agit d'une commande interne, qui peut donc fortement varier d'un shell à l'autre.

➥*Syntaxe :*
`exit` [`code_retour`]

➥*Description :* pas d'option.

➥*Exemple :* Si la commande `exit` est tapée dans la fenêtre « racine », sans qu'aucun shell fils n'ait été lancé, cela provoque une déconnexion.

➥*Codes de retour :*
- Si l'argument `code_retour` existe : la valeur (nécessairement entière) de `code_retour`.
- Sinon : le code de retour de la dernière commande précédemment exécutée.

---

7 Communication avec les autres utilisateurs

☞ mail

➥*Caractéristiques :*

1. `mail` signifie « courrier ».
2. Cette commande permet d'envoyer un message par l'intermédiaire de l'entrée standard, ou de lire des messages de manière interactive.
3. Elle peut apparaître en fin de branchement.

➥*Syntaxe :*

- Envoi d'un message : `mail` [`-tw`] [`-m` type_du_message] adresse_destinataire [...]
- Lecture du courrier : `mail` [`-ehpPqr`] [`-f` désignation_fichier]

➥*Description :* cf. la description de la commande `mail`, en tapant `man mail`

➥*Exemple :*

```
mail gerard.manvussa@gmail.com < message.txt
```

Cette commande envoie le contenu du fichier `message.txt` à l'adresse indiquée.

➥*Codes de retour :*

- Pas d'erreur : `0`
- Pas de courrier, ou opération interrompue pour cause d'erreur : `> 0`

➥*Remarque :* de nos jours, les configurations par défaut ne permettent généralement pas à cette commande d'envoyer des courriers électroniques.

---

8 Interaction avec l'interpréteur de commandes

☞ sh

➥*Caractéristiques :*

- `sh` est l'abréviation de *shell*.
- Cette commande permet de lancer un shell de Bourne. Elle permet aussi d'exécuter un script.

➥*Syntaxes :*

- `sh` [`-acefhiknprstuvx`]
  Le shell qui est lancé lit et interprète les commandes tapées sur l'entrée standard. Pour terminer l'exécution de ce shell, il faut taper la commande `exit`
- `sh` [`-acefhiknprstuvx`] nom_script [paramètre_script ...]
  Les paramètres de `sh` sont le nom d'un script, suivi des paramètres éventuels de ce script.

➥*Description :* cf. la description de la commande `sh`, en tapant `man sh`

➥*Exemples :*

- ```
  > sh
  $ pwd
  /home/ufar
  $ exit
  >
  ```
  Dans cet exemple, on remarque que le nouveau shell se distingue du shell initial par un prompt différent.
- `sh nom_script.sh` [paramètre_script ...]
  Cette commande lance l'exécution du script contenu dans le fichier `nom_script.sh`
  ```

➥*Codes de retour :* le code de retour de la commande `sh` est le code de retour de la dernière commande interprétée par le nouveau shell (dans le premier exemple ci-dessus, c'est le code de la commande `pwd`, c'est-à-dire `0`).

☞ `eval`

➥*Caractéristiques :*

1. `eval` vient de *evaluate* (« évaluer »).
2. Cette commande procède en deux temps :
   - Dans un premier temps, la commande est remplacée par la liste de ses paramètres, avec interprétation des métacaractères du shell.
   - Dans un deuxième temps, cette liste de paramètres est considérée comme une commande Unix, et exécutée en tant que telle, donc en particulier avec, à nouveau, interprétation des métacaractères du shell.
3. Elle peut apparaître en début de branchement.
4. Il s'agit d'une commande interne, qui peut donc fortement varier d'un shell à l'autre.

➥*Syntaxe :*

`eval` [`argument`$_1$ `...`]

➥*Description :* pas d'option.

➥*Exemple :*

```
a=1
c='$a'
eval echo $c
```

Cette séquence de commandes (tapées les unes à la suite des autres, ou à l'intérieur d'un script) affichera la valeur `1`, car la première évaluation de la commande `eval` produit la commande suivante :

```
echo $a
```

En revanche, la séquence suivante afficherait `$a` :

```
a=1
c='$a'
echo $c
```

➥*Codes de retour :*

- Pas d'erreur : `0`
- Opération interrompue pour cause d'erreur : `> 0`

☞ `set`

➥*Caractéristiques :*

1. `set` signifie « placer ».
2. Cette commande permet d'affecter des valeurs aux paramètres positionnels `$1`, `$2`, `$3`, ⋯, `$9` du shell courant. Appelée sans paramètre, elle permet aussi d'afficher les valeurs de toutes les variables du shell courant :
   - les variables prédéfinies ;
   - les variables définies par l'utilisateur.
3. Elle peut apparaître en début de branchement.
4. Il s'agit d'une commande interne, qui peut donc fortement varier d'un shell à l'autre.

➥*Syntaxe :*
`set` [`-aefhkntuvx`] [valeur] [...]
➥*Description :* cf. la description de la commande `set`, en tapant `man set`
➥*Exemples :*
- `set alpha beta`
  Cette commande affecte la valeur `alpha` au paramètre positionnel `$1` et la valeur `beta` au paramètre positionnel `$2`
- `x=1`
  `set`
  produira un affichage du type suivant :
  `DISPLAY=141.115.12.137:0`
  `...`
  `HOME=/home/ufar`
  `...`
  `x=1`
  Dans cet affichage, la dernière ligne concerne une variable définie par l'utilisateur, alors que toutes les lignes précédentes concernent les variables prédéfinies du shell courant.

➥*Codes de retour :* cf. la description de la commande `set`, en tapant `man set`
➥*Remarque :* la commande `set` met automatiquement à jour les paramètres `$#`, `$*` et `$@`.

☞ shift

➥*Caractéristiques :*
- `shift` signifie « déplacer ».
- Cette commande permet de réaliser un décalage vers la gauche (d'un rang ou plus) des valeurs des paramètres positionnels du shell courant.
- Il s'agit d'une commande interne, qui peut donc fortement varier d'un shell à l'autre.

➥*Syntaxe :*
`shift` [n]
➥*Description :* pas d'option.
➥*Exemples :*
- Séquence utilisant la commande `shift` :
```
$ set ab
$ echo $1
ab
$ shift
$ echo $1

$
```
- Script `deca.sh` utilisant la commande `shift` :
```
echo "Avant décalage : $1 $2 $3 $4 $5 $6 $7 $8 $9"
shift
echo "Après décalage : $1 $2 $3 $4 $5 $6 $7 $8 $9"
```
  Voici deux exemples d'exécution de ce script, avec des nombres de paramètres différents :
```
$ deca.sh 1 2 3 4 5 6 7 8 9 10
Avant décalage : 1 2 3 4 5 6 7 8 9
Après décalage : 2 3 4 5 6 7 8 9 10
$ deca.sh 1 2 3 4 5
```

```
Avant décalage : 1 2 3 4 5
Après décalage : 2 3 4 5
$
```

Donc `$1` reçoit la valeur de `$2`, `$2` reçoit la valeur de `$3`, etc.

➥*Codes de retour :*

- Pas d'erreur : `0`
- Opération interrompue pour cause d'erreur : `> 0`

➥*Remarques :*

- Lors de l'appel de la commande `shift`, l'absence d'argument `n` équivaut à une valeur de cet argument égale à `1`.
- De même que pour la commande `set`, les valeurs de `$#`, `$*` et `$@` sont mises à jour automatiquement par la commande `shift` (le paramètre `$#` est décrémenté).

---

9 Entrées et sorties

☞ `read`

➥*Caractéristiques :*

1. `read` signifie « lire ».
2. Cette commande permet d'affecter des valeurs à des variables à partir de l'entrée standard. Elle est souvent utilisée à l'intérieur d'une boucle `while`.
3. Elle peut apparaître en fin de branchement.
4. Il s'agit d'une commande interne, qui peut donc fortement varier d'un shell à l'autre.

➥*Syntaxe :*

`read` nom_variable [...]

➥*Description :* pas d'option.

➥*Exemples :*

- L'exemple suivant montre que si le nombre de mots (séparés par des espaces ou des tabulations) est supérieur au nombre d'arguments de la commande `read`, alors la variable correspondant au dernier argument reçoit toute la fin de la ligne :

```
$ read a b
c'est un exemple
$ echo $a
c'est
$ echo $b
un exemple
$
```

-
```
while read ligne
do
        ...
done < fichier
```

Une séquence de ce type permet de « passer en revue » toutes les lignes du fichier `fichier`, de la première à la dernière.

➥*Codes de retour :*

- Pas d'erreur : `0`
- Opération interrompue pour cause d'erreur : `> 0`

➥*Remarque :* les caractères tapés sur l'entrée standard ne peuvent pas contenir de caractère saut de ligne, car le premier saut de ligne tapé a pour effet de renvoyer le prompt.

☞ `echo`

➥*Caractéristiques :*

1. *echo* signifie tout simplement « écho ».
2. Cette commande permet d'afficher une ou plusieurs chaînes de caractères sur la sortie standard.
3. Elle peut apparaître en début de branchement.
4. Il s'agit d'une commande interne, qui peut donc fortement varier d'un shell à l'autre.

➥*Syntaxe :*

`echo` [`chaîne ...`]

➥*Description :* pas d'option.

➥*Exemple :* la commande `echo *` affiche la liste des noms de fichiers et de sous-répertoires du répertoire courant (fichiers cachés non compris), séparés par des espaces.

➥*Codes de retour :*

- Pas d'erreur : `0`
- Opération interrompue pour cause d'erreur : `> 0`

➥*Remarques :*

- La commande `echo` reconnaît également certains caractères spéciaux commençant par \ et appelés « caractères de contrôle » : `\n` pour le saut de ligne, `\t` pour la tabulation, `\c` pour supprimer le retour à la ligne, etc.
- Pour qu'un caractère de contrôle soit interprété en tant que tel, il faut absolument qu'il apparaisse à l'intérieur d'une chaîne de caractères délimitée par des guillemets `"` ou par des apostrophes `'`. Par exemple, les commandes `echo "\n"` ou `echo '\n'` provoqueront bien l'affichage d'un saut de ligne, mais la commande `echo \n` affichera le caractère `n` (en raison de l'interprétation du métacaractère \).
- La commande `echo` provoque toujours un saut de ligne, à la fin de l'affichage, sauf si le dernier caractère est le caractère de contrôle `\c`.

---

10 Opérations arithmétiques

☞ `expr`

➥*Caractéristiques :*

1. `expr` vient de « expression ».
2. Cette commande permet d'effectuer des opérations arithmétiques et logiques, ainsi que des opérations sur les chaînes de caractères plus sophistiquées que la simple concaténation, et affiche le résultat de ces opérations sur la sortie standard.
3. Elle peut apparaître en début de branchement.

➥*Syntaxe :*

`expr` argument$_1$ opérateur$_1$ argument$_2$ [opérateur$_2$ argument$_3$ ...]

où les arguments argument$_1$, argument$_2$, argument$_3$ sont des expressions du shell courant, et où les opérateurs opérateur$_1$, opérateur$_2$ appartiennent à la liste non exhaustive suivante :

- `+` (addition), `-` (soustraction), `\*` (multiplication), `/` (quotient de la division entière), `%` (reste de la division entière), si argument$_1$ et argument$_2$ ont comme valeurs des chaînes de caractères constituées uniquement de chiffres (et donc interprétables en tant qu'entiers).
- `=` (égalité), `!=` (différence), `\>`, `\>=`, `\<`, `\<=` (comparaisons), si argument$_1$ et argument$_2$ ont comme valeurs des chaînes de caractères quelconques. Le résultat de l'opération vaut `1` si la comparaison vaut VRAI et `0` si elle vaut FAUX. Attention : c'est exactement l'inverse pour le code de retour de la commande (cf. ci-dessous la rubrique « codes de retour »).
- `:` (extraction de sous-chaînes de caractères). Avec cet opérateur, l'argument argument$_1$ peut avoir comme valeur une chaîne de caractères quelconque, mais l'argument argument$_2$ doit être une expression régulière. Même s'il n'est pas présent, `expr` suppose que argument$_2$ commence par le métacaractère des expressions régulières `^` (début de ligne). On peut spécifier l'ensemble des caractères devant être retirés de la chaîne argument$_1$ par des caractères et des métacaractères des expressions régulières, et on doit indiquer la séquence de caractères à extraire de argument$_1$ grâce à l'expression régulière `\(.*\)`
  Dans le cas où la correspondance est impossible, le résultat de l'opération est la chaîne vide. Dans le cas où plusieurs correspondances sont possibles, le résultat de l'opération est la chaîne de caractères la plus longue.

➥*Description :* pas d'option.

➥*Exemples :*

- ```
  a=2
  b=7
  a=‘expr $b % $a‘
  echo $a
  ```
  Cette séquence produit l'affichage de `1`
- ```
  chaine="taratata"
  sschaine=‘expr $chaine : '\(.*\)at'‘
  echo $sschaine
  ```
  Cette séquence produit l'affichage suivant :
  ```
  tarat
  ```
  On aurait pu écrire l'expression régulière sous d'autres formes, comme par exemple :
  ```
  '\(.*\)at.*'
  '\(.*\)ata'
  ```

➥*Codes de retour :*

- Résultat de l'opération différent de `0` et de la chaîne vide : `0`
- Autre résultat : `1`
- Syntaxe incorrecte : `2`
- Opération interrompue pour cause d'erreur : `> 2`

➥*Remarques :*

- L'utilisation des cinq opérateurs `*`, `>`, `>=`, `<` et `<=`, parmi tous ceux qui ont été décrits ci-dessus, nécessite une vigilance particulière, car `*`, `>` et `<`, étant des métacaractères du shell, doivent être protégés. Pour ce faire, on peut soit les faire précéder d'un caractère `\`, soit les entourer de délimiteurs `"` ou `'`.
- Dans le cas où la commande `expr` est lancée avec plusieurs opérateurs, il faut être vigilant, car l'ordre des opérations ne se fait ni de gauche à droite, ni de droite à gauche, mais suivant un

ordre prédéfini de priorité des opérateurs.

11 Évaluation de conditions

☞ `test`

➥*Caractéristiques :*

1. `test` a une signification explicite.
2. Cette commande permet d'effectuer des tests de comparaison, en renvoyant le code de retour `0` lorsque le résultat est vrai, et une autre valeur sinon. Cette commande est surtout utile pour les structures de contrôle, comme la structure de contrôle de condition. Elle ne fournit pas de résultat sur sa sortie standard, mais elle renvoie un code de retour qui est égal à 0 si le test est VRAI ou à 1 si le test est FAUX.

➥*Syntaxes et description :*

- `test` comparaison ou `[` comparaison `]`
  Il faut vraiment taper les crochets et respecter les espaces, avec cette deuxième écriture.
  La comparaison comparaison peut être la combinaison booléenne de plusieurs comparaisons élémentaires :
  - comparaison_1 `-a` comparaison_2 pour le `ET` logique.
  - comparaison_1 `-o` comparaison_2 pour le `OU` logique.
  - `!` comparaison_3 pour le `NON` logique.

  Des parenthèses précédées de caractères `\` (pour protéger les parenthèses, qui sont des métacaractères du shell) peuvent être nécessaires en cas de combinaisons un peu plus compliquées, pour préciser les priorités, comme dans : `!` `\(` comparaison_1 `-o` comparaison_2 `\)`
  Chaque comparaison élémentaire doit être écrite avec la syntaxe suivante :
  expr_1 comparateur expr_2
  où expr_1 et expr_2 désignent des « expressions » du shell courant.
  Le comparateur comparateur s'écrit :
  - si les valeurs de expr_1 et expr_2 sont des chaînes de caractères constituées de chiffres :
    — `-eq` désigne l'égalité (abréviation de `eq`ual)
    — `-ne` désigne la non égalité (abréviation de `n`ot `e`qual).
    — `-gt` signifie « strictement supérieur » (abréviation de `g`reater `t`han).
    — `-ge` signifie « supérieur ou égal » (abréviation de `g`reater or `e`qual).
    — `-lt` signifie « strictement inférieur » (abréviation de `l`ess `t`han).
    — `-le` signifie « inférieur ou égal » (abréviation de `l`ess or `e`qual).
  - si les valeurs de expr_1 et expr_2 sont des chaînes de caractères quelconques :
    — `=` désigne l'égalité
    — `!=` désigne la non égalité.

  Il faut bien prendre garde à ne pas confondre les comparateurs `=` et `-eq`. Alors que `test 2 = 02` retourne FAUX, que `test 2 -eq 02` retourne VRAI et que `test baba = bobo` retourne FAUX, ce qui semble tout à fait cohérent, on sera surpris de constater que `test baba -eq bobo` retourne VRAI. Il ne faut donc pas utiliser le comparateur `-eq` pour comparer des chaînes de caractères.
  Pour tester si la variable `chaine` a comme valeur la chaîne vide, on ne peut pas écrire la commande `test $chaine = ""` car, si cette chaîne était vraiment vide, après évaluation des métacaractères, cette commande serait évaluée sous la forme `test = ""`, écriture qui est incorrecte. Pour obtenir une écriture correcte, il est alors nécessaire d'entourer l'expression contenant le nom de la variable

de guillemets : `test "$chaine" = ""`
Mais il est plus pratique d'utiliser la deuxième syntaxe de la commande `test` :

- `test $variable`
  Dans cette écriture, la commande `test` teste si la variable `variable` n'a pas pour valeur la chaîne vide. Si tel est le cas, la commande `test` retourne `0` (c'est-à-dire VRAI).
  Avec cette deuxième syntaxe, les opérateurs logiques `-a`, `-o` et `!` nécessitent l'utilisation de guillemets partout où peut apparaître la chaîne vide.
- `test -f|-d|-r|-w|-x désignation [...]`
  Avec cette syntaxe, la commande `test` vérifie si `désignation` est le nom d'un fichier (option `-f`) ou d'un répertoire (option `-d`), et si `désignation` est accessible en lecture (option `-r`), en écriture (option `-w`) ou en exécution (option `-x`) par l'utilisateur. Les opérateurs logiques `-a`, `-o` et `!` s'appliquent à cette troisième syntaxe.

➥*Exemples :*

- `test $# -eq 2 -a $1 -gt 0`
  Cette commande teste si le nombre de paramètres positionnels du shell courant ayant reçu une valeur est égal à `2`, et si le paramètre `$1` a une valeur entière strictement supérieure à `0`.
- `test $chaine`
  ```
  if [ $? -ne 0 ]
  then
          ...
  ```
  Cette séquence permet de tester si la variable `chaine` a pour valeur la chaîne vide. Le même test peut être effectué plus simplement en utilisant la séquence suivante :
  ```
  if [ ! "$chaine" ]
  then
          ...
  ```
- `test -f nom -a -r nom`
  Cette commande teste si `nom` est le nom d'un fichier accessible en lecture.

➥*Codes de retour :*

- `comparaison` évaluée à VRAI : `0`
- `comparaison` évaluée à FAUX : `1`
- Opération interrompue pour cause d'erreur : `> 1`

---

12 Accès à la date et à l'heure

☞ `date`

➥*Caractéristiques :*

1. `date` a une signification explicite.
2. Cette commande affiche la date.
3. Elle peut apparaître en début de branchement.

➥*Syntaxe :*

`date [-u] [+format]`

➥*Description :* cf. la description de la commande `date`, en tapant `man date`

➥*Exemple :*

La commande `date` affiche la date sous une forme qui dépend des paramètres d'installation du système. Voici deux exemples de formes d'affichage :

```
Mon Dec 8 21:05:47 MET 1997
Ven  8 sep 2023 09:37:58 CEST
```

➥*Codes de retour :*

- Pas d'erreur : `0`
- Opération interrompue pour cause d'erreur : `> 0`

Index