Utilisation de Linux en ligne de commande

Table des matières

I Contexte

Durée : 2h

Environnement de travail : Linux en ligne de commande

Pré-requis : Savoir gérer des fichiers et des utilisateurs sous Linux

CLI Az Définition

Command Line Interface (interface en ligne de commande)

Pourquoi la ligne de commande ?

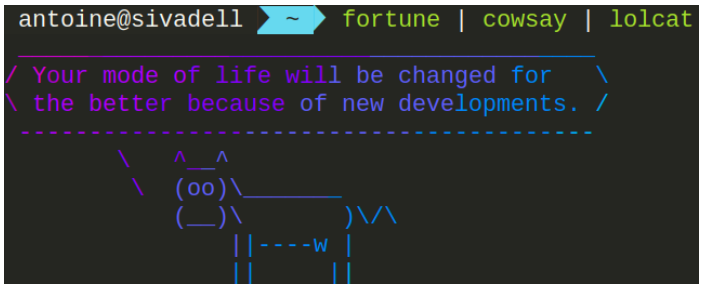
Pourquoi s'embêter à taper des commandes au lieu de cliquer sur des boutons ? Parce que c'est

- Flexible : Combinaison de commandes et accès sans écran
- Rapide : Plus économe en ressources, et rapide à utiliser (une fois pris en main)
- Éducatif : Découvrir comment les choses fonctionnent vraiment
- Puissant : Automatisation et scripting

C'est donc le couteau suisse de Linux !

Une commande quotidiennement utile Exemple

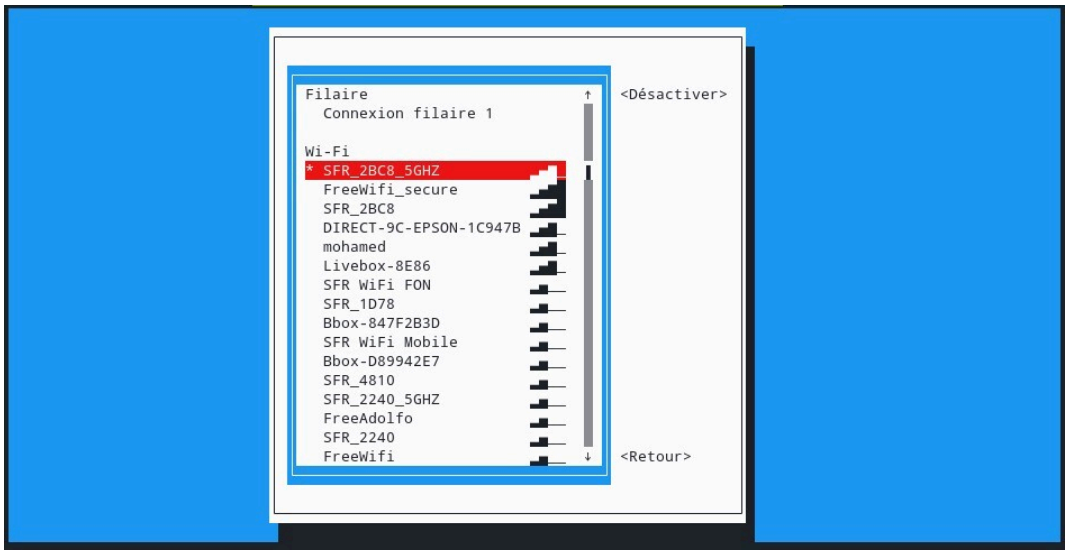
```
1 fortune | cowsay | lolcat
```



Un gestionnaire de Wifi en CLI ?

Exemple

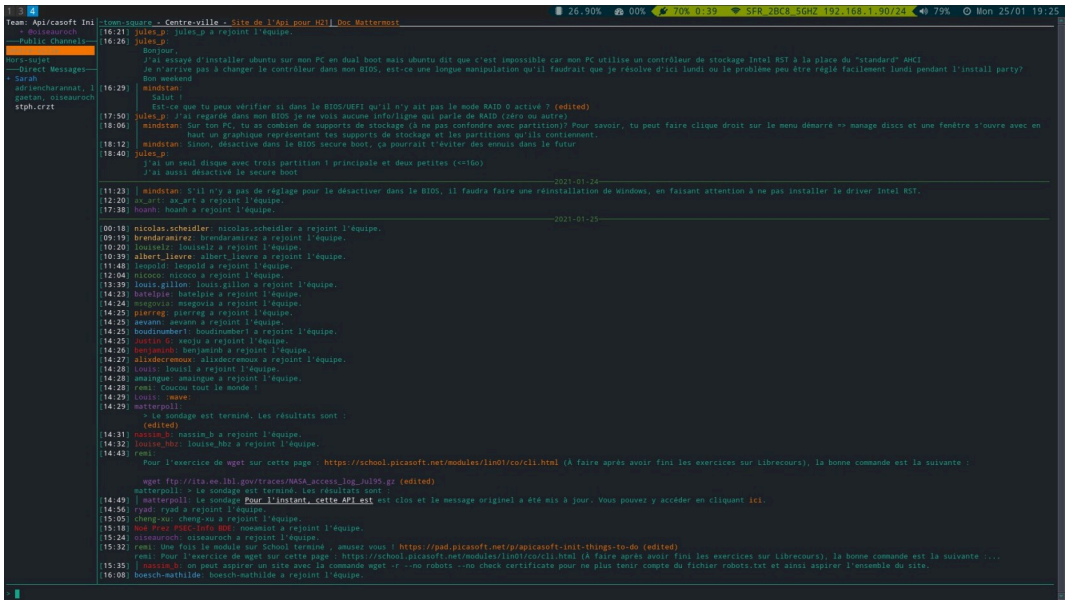
```
1 nmtui
```



Mattermost en CLI !

Exemple

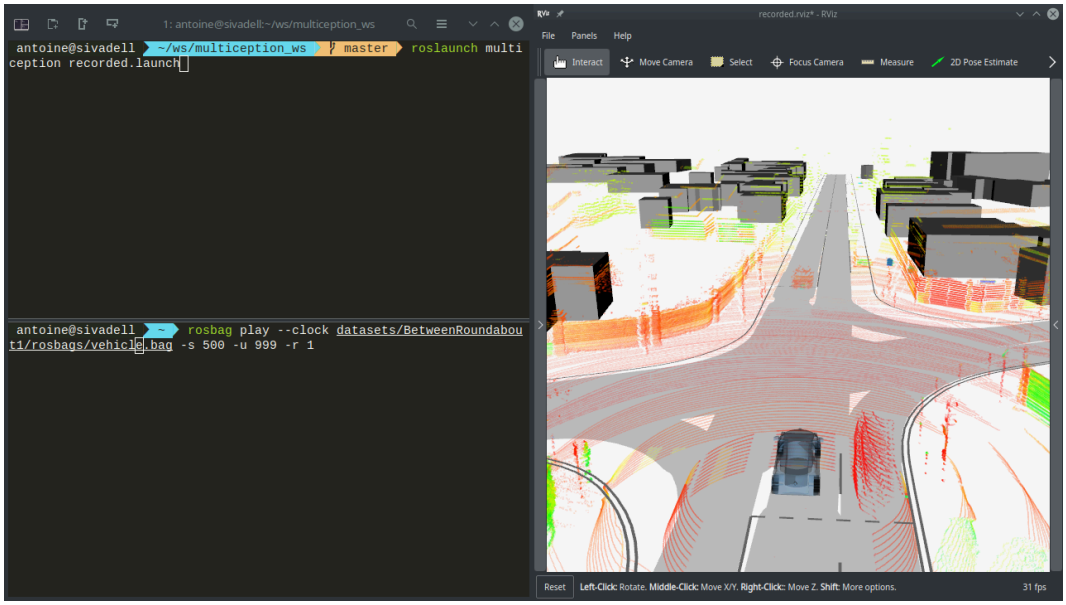
```
1 matterhorn
```



Lancer des robots voir des voitures autonomes !

Exemple

Faites SY31 ou SY27 pour faire ce genre de folleries !



II Rappels ou bases

Durée : 2h

Environnement de travail : Linux en ligne de commande

Les commandes de base en CLI

`ls`	Affiche le contenu d'un dossier
`cd`	Change de dossier
`cp`	Copie des fichiers
`mv`	Renomme ou déplace des fichiers
`rm`	Supprime des dossiers ou fichiers
`mkdir`	Crée un dossier
cat	Affiche le contenu d'un fichier

Ce sont les commandes de base à retenir !

À noter que ces commandes de bases sont directement fournies par le *shell* et sont dites *builtin*.

Shell Az Définition

C'est le programme qui interprète les commandes. On l'appelle shell car il forme une coquille autour du système d'exploitation.

C'est en fait une commande comme une autre, que vous pouvez lancer en tapant `bash`, et quitter en tapant `exit` (ou Ctrl+D pour les pros du clavier)

Organisation classique d'une commande

```
1 $ verbe --flag arguments ...
```

- Le verbe reflète généralement bien ce que fait la commande (`ls` va par exemple lister le contenu du dossier par exemple).
- Les flags sont des options passée à la commande pour préciser comment on veut s'en servir (`-v` signifiera souvent verbose, ou "donne moi un max d'infos sur ce que tu fais", utile quand une commande ne fait pas ce qu'on veut...). Il en existe deux formes, longs (`--verbose`) qui peuvent transmettre des idées complexes et courts (`-v`) qui seront généralement des idées simples (activer une fonctionnalité particulière typiquement).
- Les arguments sont les chemins ou les données sur lesquels vont s'exécuter la commande.
- Le nombre d'espaces entre les mots n'a aucune importance, du moment qu'ils séparent les différents éléments de la ligne.

Il existe souvent des flags pour rendre une commande récursive lorsqu'elle s'applique sur un dossier. Par exemple `rm -r mon_dossier` supprimera mon_dossier et tout son contenu récursivement. Dans la même veine, `mkdir -p chemin/vers/mon/dossier` créera tous les dossiers nécessaires pour arriver a ce chemin.

III Manipuler des fichiers

Objectifs

- Savoir utiliser la commande `grep`
- Savoir utiliser la commande `wc`
- Savoir utiliser la commande `sort`

grep

Méthode

La commande `grep` est l'une des plus utilisées sous Linux. Elle permet de rechercher un motif au sein de un ou plusieurs fichiers.

```
$ grep astrona /usr/share/dict/words
astronaut
astronaut's
astronautics
astronautics's
astronauts
```

La commande recherche dans le fichier `/usr/share/dict/words` toutes les lignes qui contiennent la suite de lettre "astrona".

Exemple

`grep` peut aussi être utilisée sur un dossier entier, de manière récursive

```
$ grep -r astrona /usr/share/dict
/usr/share/dict/french:astronaute
/usr/share/dict/french:astronautes
/usr/share/dict/french:astronautique
/usr/share/dict/american-english:astronaut
/usr/share/dict/american-english:astronaut's
/usr/share/dict/american-english:astronautics
/usr/share/dict/american-english:astronautics's
/usr/share/dict/american-english:astronauts
```

On constate que, pour chaque occurrence qui est trouvée, il est précisé le nom du fichier dans lequel elle se trouve.

Exemple

`grep` est une commande puissante, qui peut aussi s'appuyer sur des expressions régulières.

```
$ grep "^anti-" /usr/share/dict/french
anti-américanisme
anti-impérialisme
anti-impérialiste
anti-impérialistes
anti-inflammatoire
anti-inflammatoires
anti-inflationniste
anti-inflationnistes
```

```
10 anti-scientifique
11 anti-sous-marin
```

Ici on ne capture que les occurrences qui commencent par "anti", c'est ce que signifie le caractère ^. C'est ce que l'on appelle des expressions régulières. Cela mériterais un cours à part entière, pour approfondir, il existe de bonnes ressources[1] en ligne.

La commande wc

Méthode

La commande `wc` (pour *Word Count*) permet de compter, dans un fichier, le nombre de lignes, de mots et la taille du fichier.

```
1 $ wc lettre.tex
2   53  171 1510 lettre.tex
```

Dans l'ordre on récupère le nombre de lignes (53), le nombre de mots (171) et la taille (1510 octets) du fichier. Il est possible de n'obtenir qu'une seule de ces valeurs à l'aide d'options, par exemple `-l` pour le nombre de lignes.

```
1 $ wc -l lettre.tex
2   53 lettre.tex
```

La commande sort

Méthode

Une autre commande très utile est `sort`. Comme son nom l'indique elle permet de retourner le contenu d'un fichier, mais trié.

```
1 sort /usr/share/dict/words
```

Par défaut elle trie par ordre alphabétique, mais il est possible de trier selon un ordre croissant numérique avec l'option `-n`.

À retenir

On peut facilement effectuer des recherches avec `grep`, du tri avec `sort` et compter les lignes/mots d'un fichier avec `wc`.

1. https://openclassrooms.com/fr/courses/918836-concevez-votre-site-web-avec-php-et-mysql/916990-les-expressions-regulieres-partie-1-2

IV Exercice : Appliquer la notion

On souhaite obtenir le nombre de mots qui contiennent la lettre w en Français. Pour cela on s'aide du fichier `/usr/share/dict/french` qui une liste de mots français. Elle est présente par défaut, sinon il est possible de l'installer avec la commande suivante :

```
1 sudo apt-get install wfrench
```

Question

[solution n°1 p. 23]

Obtenez le nombre de mot qui contiennent la lettre w dans ce fichier.

Indice :

On peut utiliser un fichier temporaire pour stocker les mots trouvés.

V Les flux de redirections

Objectifs

- Connaître les 3 canaux de communications (stdin, stdout, stderr)
- Savoir rediriger les différents canaux d'une commande

Mise en situation

On a l'habitude d'utiliser des commandes qui retournent un résultat directement dans notre terminal. En réalité il est possible de rediriger ce résultat ailleurs, par exemple dans un fichier, en utilisant les différents canaux de communications d'une commande.

Sortie standard Az Définition

La sortie standard d'un programme, que l'on nomme `stdout`, est un flux qui va contenir tout ce qui est affiché à l'écran par la commande. Par défaut, cette sortie standard dirigée vers le terminal, pour que l'on puisse consulter le résultat de la commande.

Sortie d'erreurs Az Définition

La sortie d'erreurs, que l'on nomme `stderr`, est similaire à la sortie standard à la différence que ce flux est destiné aux erreurs renvoyés par la commande. Par défaut cette sortie est dirigée vers le terminal, et s'affiche donc en même temps que la sortie standard.

Flux de redirections

Les flux de redirections (ou simplement, redirections) sont des opérateurs qui permettent de diriger la sortie standard ou d'erreurs d'une commande vers une autre destination, en particulier vers un fichier.

Syntaxe

Les opérateurs > et >> permettent de rediriger la sortie standard d'une commande dans un fichier :

- > efface complètement le contenu du fichier avant de rediriger le flux,
- >> ajouter en fin de fichier la sortie standard.

Opérateurs de redirection de stdin

Exemple

On peut rediriger le résultat de la commande `cat`.

```
$ cat TODO-perso.txt
- Faire les courses
- Déclarer mes impôts
- Réparer le pommeau de douche
$ cat TODO-boulot.txt
- Mettre en place les sauvegardes des postes
- Déployer le nouveau site web
- Poser les jours de congés de cet été
$ cat TODO-perso.txt TODO-boulot.txt > TODO.txt
$ cat TODO.txt
- Faire les courses
- Déclarer mes impôts
- Réparer le pommeau de douche
- Mettre en place les sauvegardes des postes
- Déployer le nouveau site web
- Poser les jours de congés de cet été
```

Ici on voit que les 2 fichiers contiennent du texte, la commande `cat` a permis de les concaténer, et le résultat a été envoyé dans le fichier `TODO.txt` (au lieu du terminal) grâce à l'opérateur >.

(cf. TODO-perso.txt)

(cf. TODO-boulot.txt)

Exemple

Si l'on reprend la commande `cat` précédente, mais avec un fichier qui n'existe pas

```
$ cat TODO-perso.txt TODO-fauxfichier.txt > TODO.txt
cat: TODO-fauxfichier.txt: Aucun fichier ou dossier de ce type
$ cat TODO.txt
- Faire les courses
- Déclarer mes impôts
- Réparer le pommeau de douche
```

On constate que la sortie de la commande (pour le fichier qui existe bien) a été redirigée dans le fichier `TODO.txt`, mais l'erreur pour le fichier n'existant pas s'affiche dans la console.

Opérateurs de redirection de stderr

Méthode

De la même manière que pour la sortie standard, la sortie d'erreurs peut-être redirigée à l'aide de deux opérateurs : 2> et 2>>.

```
$ cat TODO-perso.txt TODO-fauxfichier.txt > TODO.txt 2> erreur.txt
$ cat erreur.txt
cat: TODO-fauxfichier.txt: Aucun fichier ou dossier de ce type
```

Le fonctionnement est similaire à celui des redirections de `stdin`, ici c'est le fichier `erreur.txt` qui reçoit le flux d'erreurs. La différence entre 2> et 2>> est la même que entre > et >>.

Fusionner les sorties

Méthode

Il est aussi possible de rediriger `stdin` et `stderr` ensembles dans un même fichier. On utilise toujours > ou >> pour la sortie standard, et on se contente de rajouter l'opérateur 2>&1 pour indiquer de rediriger `stderr` vers la même destination.

```
$ cat TODO-perso.txt TODO-fauxfichier.txt > TODO.txt 2>&1
$ cat TODO.txt
- Faire les courses
- Déclarer mes impôts
- Réparer le pommeau de douche
cat: TODO-fauxfichier.txt: Aucun fichier ou dossier de ce type
```

Entrée standard

Az Définition

L'entrée standard d'un programme, que l'on appelle `stdin`, est le flux d'informations qu'il va recevoir en entrée. C'est l'inverse de la sortie standard.

Redirection d'un fichier vers stdin

Méthode

L'opérateur < permet de rediriger le contenu d'un fichier sur l'entrée standard d'une commande. Par exemple la commande wc permet de récupérer le nombre de lignes, de mots, et la taille en octets d'un contenu envoyé sur son entrée standard.

```
$ wc < TODO-perso.txt
 3 14 76
```

Le contenu de `TODO-perso.txt` a été envoyé sur l'entrée standard de wc, qui compte ainsi 3 lignes, 14 mots et 76 octets.

À retenir

Chaque programme se voit attribuer 3 flux par le système d'exploitation : l'entrée standard (`stdin`), la sortie standard (`stdout`) et la sortie d'erreurs (`stderr`). Par défaut les flux de sorties sont redirigés vers le terminal, mais des opérateurs de redirection permettent de diriger ces flux dans des fichiers.

VI Exercice : Appliquer la notion

On souhaite lister le contenu de 3 dossiers de la machine dans un fichier `files.txt` :

- `/etc/network/` (qui devrait exister)
- `/var/lib/apt` (qui devrait exister)
- `/etc/fake-folder` (qui ne devrait pas exister)

Le fichier `files.txt` devra contenir uniquement la liste des fichiers, et les éventuelles erreurs doivent être écrites dans un fichier `errors.txt`.

Question

[solution n°2 p. 23]

Donnez la commande pour réaliser cette opération.

Indice :

On utilisera `ls -l` et quelques redirections.

VII Chaîner les commandes

Objectifs

- Comprendre ce qu'est un *pipe*
- Savoir chaîner des commandes avec un *pipe*

Le pipe

Az Définition

Un *pipe* (tube ou tuyau en anglais) est un outil qui permet de connecter la sortie standard d'une commande avec l'entrée standard d'une autre commande. Cela permet de chaîner plusieurs commandes, le résultat de l'une formant les données d'entrée de la suivante.

Utiliser un pipe

Méthode

Pour créer un *pipe* entre 2 commandes, il suffit d'écrire les deux commandes séparées pour un caractère `|` (que l'on appelle le *pipe*).

```
$ ls -R /home | less
```

La commande `ls -R /home` liste tout les fichiers et dossiers de `/home` de manière récursive, ce qui produit énormément de résultats dans le terminal. L'utilisation du *pipe* permet de rediriger la sortie standard vers la commande `less` qui va nous permettre de naviguer dans le résultat de la commande (son `stdin` donc).

Filtrer des mots

Exemple

La commande `grep` permet de faire des recherches et/ou de filtrer du texte dans un fichier. Mais elle permet aussi de traiter un flux reçu sur son entrée standard.

```
$ ls -R /home | grep "picasoft"
/home/kyane/code/picasoft/registre:
/home/kyane/code/picasoft/stats-page:
/home/kyane/documents/picasoft/documents:
/home/kyane/documents/picasoft/api:
/home/kyane/documents/picasoft/doc:
```

Ici on récupère la liste des fichiers, mais on filtre toutes les lignes qui contiennent le texte "picasoft". Grâce au *pipe* la commande `grep` travaille directement sur le résultat de la commande `ls`.

À retenir

Il est possible de créer un *pipe*, un tuyau, entre la sortie standard d'une commande et l'entrée standard d'une autre commande, à l'aide du caractère `|`.

VIII Exercice : Appliquer la notion

On souhaite trier les notes d'un fichier CSV donné ici.

(cf. notes.csv)

On voit que le format de ce fichier est le suivant pour chaque ligne :

- en premier un prénom
- en second, séparé par une virgule, une note (sur 20)

L'objectif est d'afficher la liste des notes par ordre croissant dans le terminal.

On pourra s'appuyer sur la commande `cut` qui permet de découper chaque ligne d'un fichier selon un délimiteur précis et de restituer certains champs.

Question

[solution n°3 p. 24]

En lisant le manuel des deux commandes et en utilisant un *pipe*, donnez une commande permettant d'afficher la liste des notes, uniquement, dans l'ordre croissant.

Indice :

`cut` permet de découper les lignes selon les virgules à l'aide de l'option `-d","`

Indice :

`cut` permet de restituer uniquement les notes, c'est à dire la deuxième colonne, avec l'option `-f2`

IX Astuces en ligne de commande

Objectifs

- Savoir utiliser l'autocompletion
- Savoir naviguer dans le terminal

L'autocomplétion

Méthode

L'autocomplétion est un mécanisme qui permet de compléter automatiquement une commande. Après avoir tapé le début de la commande, on appuie sur la touche TAB pour que la console complète, si possible la commande en entier. Par exemple si l'on écrit :

```
$ sor
```

L'appui sur la touche TAB va compléter la commande pour écrire :

```
$ sort
```

Si il existe plusieurs commandes possibles commençant par ce qui a été écrit, alors en appuyant 2 fois sur la touche TAB la liste des commandes possibles va s'afficher. Par exemple :

```
$ gre
gregorio   grep       gresource
```

Conseil

L'autocomplétion fonctionne pour les commandes, mais aussi pour les arguments, les chemins de fichiers, etc. Elle est très pratique au quotidien pour aller plus vite dans l'utilisation de la ligne de commande.

Naviguer dans une ligne

Méthode

Dans la console, il est possible de se déplacer à l'aide des touches directionnelles du clavier. Mais il est aussi possible de se déplacer au tout début de la ligne en faisant CTRL+a ou tout à la fin avec CTRL+e.

Historique de commandes

Méthode

Le terminal maintient un historique des commandes et permet de les rappeler pour ne pas avoir à les retaper. Les touches haut et bas permettant de naviguer dans cet historique de commandes. Il est aussi possible de l'afficher à l'aide de la commande `history`. Enfin, en appuyant sur CTRL+r il est possible de faire une recherche dans l'historique.

```
$ grep "w" /usr/share/dict/french > mots_w.txt
bck-i-search: grep
```

Ici la recherche du mot `grep` dans l'historique retourne la dernière commande qui correspond à ce terme. En appuyant sur Entrée, il est possible de la relancer.

À retenir

Au quotidien différents outils permettent d'aller plus vite dans une console : l'autocomplétion, les raccourcis et l'historique.

X Exercice : Appliquer la notion

Question

[solution n°4 p. 24]

À l'aide de l'autocomplétion, comment obtenir toutes les commandes qui commencent par `lo` ?

XI Quiz

Exercice 1 : Quiz - Culture

[solution n°5 p. 24]

Exercice

La sortie standard est :

- **A** Le programme qui permet de gérer l'extinction de la machine
- **B** Le statut de succès ou non d'une commande à la fin de son exécution
- **C** Le flux sur lequel un programme va pouvoir écrire des informations en sortie

Exercice

Sous Linux, un *pipe* est :

- **A** un outil pour transférer le contenu d'un fichier vers un autre
- **B** un outil pour diriger la sortie standard d'une commande vers l'entrée standard d'une autre commande
- **C** une instruction permettant d'exécuter une instruction de manière conditionnelle
- **D** le logo officiel de Linux, symbole de la célèbre pipe utilisée par Linus Torvalds

Exercice

À quoi sert la commande `grep` ?

- **A** À rechercher un fichier sur le disque
- **B** À rechercher dans un fichier
- **C** À trier un fichier
- **D** À compter les mots d'un fichier

Exercice 5 : Quiz - Méthode

[solution n°6 p. 25]

Exercice

Pour rediriger le résultat d'une commande dans un fichier, je peux utiliser.

- **A** `macommande > monfichier`
- **B** `macommande | monfichier`
- **C** `macommande &> monfichier`
- **D** `macommande >> monfichier`

Exercice

Pour autocompléter un commande on utilise

- **A** CTRL+a dans le terminal
- **B** La touche TAB
- **C** La flèche directionnelle vers la droite
- **D** La touche INSER

Exercice

Pour retourner au début d'une ligne dans la console on utilise

- **A** CTRL+a
- **B** La touche TAB
- **C** La flèche directionnelle vers la gauche
- **D** La touche INSER

Exercice

Comment retrouver une commande dans l'historique des commandes ?

- **A** Utiliser la flèche du haut du clavier
- **B** Utiliser le raccourci CTRL+r et recherche la commande
- **C** Cherche dans le retour de la commande `history`

Exercice 10 : Quiz - Code

[solution n°7 p. 26]

Exercice

De quelle(s) manière(s) est-il possible de récupérer le résultat d'une commande pour l'utiliser dans une autre commande ?

- **A** `commande1 | commande2`
- **B** `commande1 >> commande2`
- **C** `res=$(commande1) commande2 $res`
- **D** `commande1 < commande2`

Exercice

Que fait la commande suivante `cat monfichier | wc -m`

- **A** Elle retourne le nombre de lignes, de mots et la taille du fichier `monfichier`
- **B** Elle affiche le contenu du fichier `monfichier`
- **C** Elle retourne le nombre de lettre dans le fichier `monfichier`

Solutions des exercices

Solution n°1

[exercice p. 10]

On commence par récupérer les mots qui contiennent un `w`, en redirigeant le résultat dans un autre fichier.

```
$ grep "w" /usr/share/dict/french > mots_w.txt
```

Ensuite il suffit de compter le nombre de lignes dans ce fichier avec la commande `wc`

```
$ wc -l mots_w.txt
594 mots_w.txt
```

Solution n°2

[exercice p. 14]

On utilise la commande `ls -l` avec les 3 dossiers en paramètres pour lister le contenu des dossiers. L'opérateur > permet de rediriger la sortie standard (donc la liste des fichiers) dans le fichier `files.txt` tandis que l'opérateur 2> redirige les erreurs dans le fichier `errors.txt`.

```
ls -l /etc/network /var/lib/apt /etc/fake-folder > files.txt 2> errors.txt
```

On peut constater que le résultat a été correctement redirigé, et que l'erreur du dossier n'existant pas se trouve bien à part.

```
$ cat files.txt
/etc/network:
total 24
drwxr-xr-x 2 root root 4096  7 déc.  16:16 if-down.d
drwxr-xr-x 2 root root 4096  7 déc.  16:16 if-post-down.d
drwxr-xr-x 2 root root 4096  7 déc.  16:16 if-pre-up.d
drwxr-xr-x 2 root root 4096  7 déc.  16:16 if-up.d
-rw-r--r-- 1 root root  313  3 oct.   2016 interfaces
drwxr-xr-x 2 root root 4096 13 mars   2015 interfaces.d
lrwxrwxrwx 1 root root   12 26 nov.   2015 run -> /run/network

/var/lib/apt:
total 240
-rw-r--r-- 1 root root    269 26 nov.   2015 cdroms.list
-rw-r--r-- 1 root root      0  8 déc.  12:30 daily_lock
-rw-r--r-- 1 root root 188799  7 déc.  19:01 extended_states
-rw-r--r-- 1 root root  12288  7 déc.  16:14 listchanges.db
-rw-r--r-- 1 root root  12288 30 nov.  09:51 listchanges-old.db
drwxr-xr-x 4 root root  12288  7 déc.  16:36 lists
drwxr-xr-x 3 root root   4096 26 nov.   2015 mirrors
drwxr-xr-x 2 root root   4096 10 juin   2015 periodic
$ cat errors.txt
ls: impossible d'accéder à '/etc/fake-folder': Aucun fichier ou dossier de ce type

```

Solution n°3

[exercice p. 16]

On utilise la commande suivante.

```
$ cut -d"," -f2 notes.csv | sort -n
5
8
8
10
12
15
18
19
```

`cut -d","` permet de découper notre fichier CSV en colonnes en utilisant la virgule comme séparateur. L'option `-f2` permet de récupérer la seconde colonne, c'est à dire la liste des notes.

À l'aide d'un *pipe* on redirige cette sortie sur la commande `sort` qui permet de trier par ordre croissant avec l'option `-n`.

Solution n°4

[exercice p. 19]

On commence à écrire `lo` dans le terminal et on appuie ensuite sur la touche TAB.

```
$ lo
loadkeys        local           localectl       lodraw          logger
logname         logsave         lomath          loweb
loadunimap      localc          localedef       loffice         login
logout          loimpress       look            lowntfs-3g
lobase          locale          locale-gen      lofromtemplate  loginctl
logrotate       lollipop        losetup         lowriter
```

Solution n°5

[exercice p. 20]

Exercice

La sortie standard est :

- **A** Le programme qui permet de gérer l'extinction de la machine
- **B** Le statut de succès ou non d'une commande à la fin de son exécution
- (C) Le flux sur lequel un programme va pouvoir écrire des informations en sortie

Exercice

Sous Linux, un *pipe* est :

- **A** un outil pour transférer le contenu d'un fichier vers un autre
- **B** ◯ un outil pour diriger la sortie standard d'une commande vers l'entrée standard d'une autre commande
- **C** une instruction permettant d'exécuter une instruction de manière conditionnelle

D le logo officiel de Linux, symbole de la célèbre pipe utilisée par Linus Torvalds

Exercice

À quoi sert la commande `grep` ?

A À rechercher un fichier sur le disque

B À rechercher dans un fichier

C À trier un fichier

D À compter les mots d'un fichier

Solution n°6

[exercice p. 20]

Exercice

Pour rediriger le résultat d'une commande dans un fichier, je peux utiliser.

A `macommande > monfichier`

B `macommande | monfichier`

C `macommande &> monfichier`

D `macommande >> monfichier`

Exercice

Pour autocompléter un commande on utilise

A CTRL+a dans le terminal

B La touche TAB

C La flèche directionnelle vers la droite

D La touche INSER

Exercice

Pour retourner au début d'une ligne dans la console on utilise

A CTRL+a

B La touche TAB

C La flèche directionnelle vers la gauche

D La touche INSER

Exercice

Comment retrouver une commande dans l'historique des commandes ?

A Utiliser la flèche du haut du clavier

B Utiliser le raccourci `CTRL+r` et recherche la commande

C Cherche dans le retour de la commande `history`

Techniquement les 3 méthodes sont valables, mais la 3ème est assez inefficace, tandis que la second est très rapide.

Solution n°7

[exercice p. 22]

Exercice

De quelle(s) manière(s) est-il possible de récupérer le résultat d'une commande pour l'utiliser dans une autre commande ?

A `commande1 | commande2`

B `commande1 >> commande2`

C `res=$(commande1) commande2 $res`

D `commande1 < commande2`

Exercice

Que fait la commande suivante `cat monfichier | wc -m`

A Elle retourne le nombre de lignes, de mots et la taille du fichier `monfichier`

B Elle affiche le contenu du fichier `monfichier`

C Elle retourne le nombre de lettre dans le fichier `monfichier`