

# #!/bin/bash

```
~root: env X="() { ;;} ; echo shellshock" /bin/sh -c "echo completed"
```

```
> shellshock  
> completed
```

## La ligne de commande

---

### Le prompt

Le shell désigne un interpréteur de ligne de commande. La ligne de commande est l'unité d'interaction avec l'utilisateur.

Cette interaction est faite par du texte, et est associée à la notion de terminal. Le rôle du shell est de traiter la ligne de commande et d'appeler le ou les programmes concernés.

Ils peuvent être issus du Bash lui-même, du système d'exploitation, ou bien être propres à l'utilisateur. Découvrons ensemble la ligne de commande. Vous devriez apercevoir ce qu'on appelle une invite de commande aussi connue sous le nom de prompt.



Video 1

```
etienne@fixe:~$
```

Elle se termine par un curseur généralement représenté par un rectangle clignotant. La présence de cette invite indique que le shell est prêt à accepter une nouvelle ligne de commande. Intéressons-nous maintenant aux caractères qui précèdent le curseur.

Dans le terminal que nous utiliserons dans ce cours, le curseur ne sera précédé que par un dollar.

Cependant, l'invite de commande peut contenir des informations qui permettent de contextualiser le terminal.

Ainsi, il est souvent composé :

- d'un identifiant d'utilisateur pour qui les commandes vont être exécutées (**etienne**)
- du nom de la machine pour laquelle s'exécute le shell (**fixe**)
- du répertoire dans lequel sera exécutée la commande (~ qui signifie le répertoire courant de l'utilisateur)

Elle contient éventuellement d'autres informations que nous verrons dans les activités suivantes. L'invite de commande indique en un clin d'œil le contexte dans lequel sera exécutée la commande. Cette information réduit les chances de se tromper.

## Les commandes

Une commande est un appel d'un programme. Après avoir saisi la ligne de commande, il suffit d'appuyer sur Entrée pour lancer son exécution.

Le résultat s'affiche alors à l'écran et l'utilisateur peut être sollicité pour entrer des données supplémentaires.

Une fois l'exécution terminée, l'invite de commande s'affiche à nouveau ce qui indique que le shell est prêt à recevoir une nouvelle ligne de commande.

```
etienne@fixe:~$ date
ven. 12 févr. 2021 08:14:49 EAT
etienne@fixe:~$
```

Une ligne de commande est constituée par des mots séparés par des espaces. Le format général est le nom de la commande suivi d'arguments qui seront interprétés par cette dernière. Le nombre d'arguments dépend de la commande et de la tâche qu'elle doit effectuer. Par exemple, la commande `date` peut être appelée sans aucun argument.

```
etienne@fixe:~$ date
ven. 12 févr. 2021 08:14:49 EAT
etienne@fixe:~$
```

La commande `echo` accepte un nombre quelconque d'arguments. Son rôle est simplement d'afficher les mots qui lui ont été passés en arguments. Notons au passage que le nombre d'espaces qui séparent les arguments d'une commande n'a pas d'effet.

```
etienne@fixe:~$ date
ven. 12 févr. 2021 08:14:49 EAT
etienne@fixe:~$
```

Lorsqu'un argument est une lettre précédée d'un tiret, il est appelé une option.

Par exemple, l'option `-u` de la commande `date` permet d'afficher l'heure universelle aussi connue sous le nom de UTC. Une option d'une lettre est peu explicite, et elles ont souvent un équivalent plus long qui sera préfixé par deux tirets.

Par exemple, `date --utc` a la même signification que `date -u`.



## Bilan

Lorsqu'une invite de commande s'affiche, cela signifie que vous pouvez entrer une ligne de commande. Une fois validée, elle est interprétée par le shell et vous devez attendre que la commande termine son exécution pour que l'invite de commande s'affiche à nouveau. Nous avons également vu la structure d'une ligne de commande. Elle est composée d'un nom, suivi d'options et d'arguments. Notez cependant que la commande a une syntaxe qui lui est propre. Il est indispensable d'obtenir de l'aide pour savoir l'utiliser. C'est d'ailleurs l'objet de la prochaine partie.



## Q.C.M. 1

1. Parmi les affirmations suivantes concernant la ligne de commande shell, laquelle est vraie :

1. La ligne de commande sert à lancer l'exécution d'une commande mais pas à spécifier des arguments.
2. La ligne de commande permet notamment d'allumer l'ordinateur.
3. La ligne de commande sert à lancer l'exécution d'une commande avec éventuellement des arguments.
4. Il est indispensable d'avoir une souris pour saisir une ligne de commande.

2. On considère que la commande `cal -m apr` est constituée de 3 mots. Quel est le nom de la commande ?

1. cal
2. -m
3. apr

3. Les propositions ci-dessous décrivent des propriétés de l'invite de commande. Laquelle est fausse ?

1. Elle permet de savoir si le terminal est prêt à exécuter une nouvelle commande
2. Elle permet de savoir si on est connecté à l'Internet
3. Elle permet de connaître le nom de l'utilisateur du terminal
4. Elle permet de connaître le nom de la machine sur laquelle s'exécute le terminal

## Trouver de l'aide

Lorsque vous souhaitez réaliser une tâche, vous ne savez pas nécessairement quelle est la commande qui vous sera utile.

Et même si une fée vous souffle à l'oreille le nom de la bonne commande, il y a peu de chances pour que vous sachiez comment l'utiliser.

Rassurez-vous, tout n'est pas perdu. De l'aide existe.

Les outils du Bash sont là pour vous sauver et vous n'aurez même pas à recourir à votre moteur de recherche internet. Notons toutefois que cette aide est souvent en anglais. L'aide est disponible dans plusieurs endroits.

Le mieux placé pour vous expliquer comment utiliser un programme, c'est encore celui qui l'a développé. Ainsi, l'énorme majorité des commandes disposent d'une aide interne.



Video 2

- **L'erreur de syntaxe** : Si vous ne savez pas comment y accéder, ce n'est pas grave, il vous suffit de faire une erreur de syntaxe.

```
etienne@fixe:~$ sed
Utilisation: sed [OPTION]... {script-seulement-si-pas-d'autre-script}
[fichier-d'entrée]...

-n, --quiet, --silent      supprimer l'écriture automatique de l'espace des motifs
--debug
```

Prenons par exemple le cas de la commande `sed`, dont nous ignorons tout. Lorsque l'on tape la commande `sed` avec une erreur de syntaxe, `sed` comprend tout de suite que notre commande n'a aucun sens et nous propose un bref résumé de la syntaxe à utiliser.

- **L'option `-h` ou `--help`** :

Il est souvent possible d'invoquer l'aide interne du programme directement avec l'option `-h` ou `--help`. Par exemple, la commande `cut`.

```
etienne@fixe:~$ cut --help
Utilisation : cut OPTION... [FICHIER]...
Afficher les extraits sélectionnés des lignes de chaque FICHIER vers la sortie standard.

Sans FICHIER ou quand FICHIER est -, lire l'entrée standard.

Les arguments obligatoires pour les options longues le sont aussi pour les
```

On va afficher une aide similaire avec `--help` et `-h`.

Notons que la documentation est directement affichée dans le terminal. Il n'est donc pas aisé de naviguer entre les différentes parties ou même de rechercher du texte à l'intérieur. De plus, l'aide affichée est souvent peu détaillée.

- **Le manuel**

Il existe sur le système d'exploitation des manuels qui constituent une aide plus structurée et formalisée. Une commande appelée `man` permet de les consulter. Lorsque l'on cherche à accéder à la documentation d'une commande, il suffit d'appeler la commande `man` avec pour argument le nom du programme dont on cherche le manuel.

Cette commande accepte des options. Celle qui permet de spécifier la langue de l'aide est `-L fr`.

Enfin, `man` est une commande comme une autre.

Il existe donc un manuel pour son utilisation.

Étudions la structure du manuel avec par exemple l'aide de la commande `date`.

```
DATE(1) User Commands
NAME
date - print or set the system date and time
SYNOPSIS
date [OPTION]... [+FORMAT]
date [-u|--utc|--universal] [MMDDhhmm[[CC]YY][.ss]]
DESCRIPTION
Display the current time in the given FORMAT, or set the system date.

Mandatory arguments to long options are mandatory for short options too.

-d, --date=STRING
display time described by STRING, not 'now'
```

Une page de manuel contient plusieurs parties dont les plus importantes sont le nom, le synopsis, la description, les options et les exemples.

La partie synopsis propose un résumé du format de la commande. Il utilise les conventions suivantes. Le texte en gras est à taper exactement comme indiqué. Les arguments qui sont entre crochets sont facultatifs. De même, toute expression entre crochets est facultative et peut être répétée plusieurs fois.

Ainsi le synopsis de la commande `date` nous indique qu'elle peut être lancée sans argument mais qu'elle peut aussi contenir des options et/ou un format d'affichage. La section description détaille la signification des différentes options et des arguments attendus. La section exemples permet quant à elle de valider notre compréhension de la partie description. Grâce au manuel, on sait maintenant que pour afficher une date comme étant le nombre de secondes écoulées depuis 1970, il suffit d'ajouter `+` pour définir le format suivi de `%s` pour spécifier les secondes.

Pour **quitter** le manuel, appuyez sur la touche `q`.

## Comment trouver une commande ?

Il peut arriver que vous ayez besoin d'une commande qui effectue une certaine tâche, mais que vous n'en connaissiez pas le nom.

L'outil `apropos` vous sera alors d'une grande aide.

C'est une commande qui permet de lister les manuels dont la partie description contient les mots passés en arguments. Imaginons par exemple que vous ayez besoin d'un convertisseur d'encodage de fichier. Le mot-clé `encodage` permettra à `apropos` de trouver l'utilitaire recherché.

En lisant le résultat, on en déduit rapidement que la commande `iconv` est celle qui répond à nos besoins. La consultation du `man` de `iconv` vous permettra de savoir comment utiliser cette commande.



### Bilan

Il est inutile de faire des recherches sur internet pour utiliser votre shell Bash. Nous avons vu l'aide qui est intégrée directement dans les programmes et qui est affichée à travers l'option `-h` ou lors d'une faute de syntaxe. Nous avons également vu l'aide basée sur les fichiers de manuel.

Celle-ci peut être en français et consultée à travers la commande `man`.

Nous avons également vu `apropos` qui permet de trouver les commandes associées à une tâche à effectuer. Félicitations, vous êtes maintenant autonome pour trouver les commandes à utiliser.



## Q.C.M. 2

- Parmi les types d'aide suivants, lequel n'est pas toujours utilisable depuis un shell ?
  - Aide en ligne sur Internet rédigée au format PDF
  - Aide interne d'une commande
  - Aide interne d'une commande suite à une erreur d'utilisation
  - Aide du manuel (man)
- Pour l'utilisation du shell, dans quel cas une recherche sur Internet sera-t-elle indispensable ?
  - Lorsqu'on ne sait pas utiliser une commande
  - Lorsqu'on sait ce qu'on souhaite faire mais qu'on ne sait pas quelle commande utiliser
  - Lorsqu'on fait une erreur de syntaxe lors de l'utilisation d'une commande
  - Aucune des propositions ci-dessus
- Quelle option de commande est généralement utilisée pour accéder à l'aide interne de la commande ?
  - L'option `-a` ou `-aide`
  - Passer le nom de la commande en argument de l'utilitaire `aide`.
  - L'option `-h` ou `--help`
  - L'option `-m` ou `--man`
  - L'option `-i` ou `--info`
- Parmi les lignes de commande suivantes, laquelle permet de lister les commandes en rapport avec la notion d'encodage ?
 

1. <code>man encodage</code>	4. <code>emacs encodage</code>
2. <code>apropos encodage</code>	
3. <code>encodage --help</code>	5. <code>encodage apropos</code>

## Naviguer dans les dossiers

Sous Unix, toutes les ressources de votre ordinateur sont représentées par des fichiers. Les communications, les périphériques et les données passent par le système de fichiers.

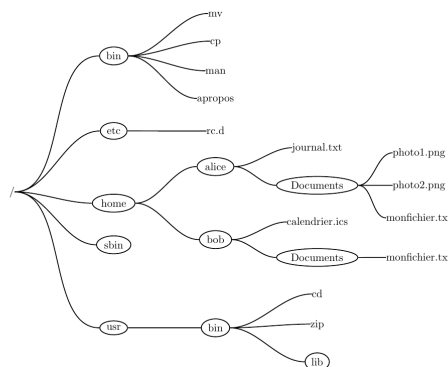
Le **système de fichiers** est la partie la plus visible d'un système d'exploitation. Il se charge de gérer le stockage et la manipulation de fichiers (lecture et écriture).

### Arborescence

La structure de stockage des fichiers est organisée hiérarchiquement selon une arborescence. Une arborescence est un **graphe** en forme d'arbre composé d'éléments pour lesquels les feuilles aux extrémités des branches sont les fichiers et les branches sont des répertoires.

Une branche peut avoir plusieurs feuilles mais une feuille n'est attachée qu'à une seule branche. Une branche en plus des feuilles peut avoir plusieurs sous-branches.

Dans cette partie du cours, on prendra pour exemple l'arborescence suivante :



Video 3

L'arborescence démarre d'un répertoire particulier nommé **racine** ou **root** en anglais, noté `/`. Ensuite nous allons trouver plusieurs répertoires :

- des répertoires systèmes comme `bin` ou `usr` qui contiennent les commandes que vous allez utiliser ;
- des répertoires importants au bon fonctionnement de votre système d'exploitation comme **etc** car il contient les fichiers de configuration de votre système d'exploitation ;
- des répertoires spécifiques comme **tmp** qui sert de stockage temporaire (au redémarrage de la machine, son contenu est supprimé) ;
- le répertoire des utilisateurs **home** qui comporte le répertoire personnel de chaque utilisateur de la machine, aussi appelé home directory en anglais. Ainsi l'utilisateur alice possède-t-il son propre répertoire appelé alice et situé dans `/home`.

## Chemins d'accès dans l'arborescence

Le chemin d'accès est l'expression pour localiser un fichier ou un répertoire dans l'arborescence. Il est constitué d'une liste de noms identifiant les branches du chemin. Il existe deux types de chemins. Le chemin dit absolu a l'avantage de désigner un élément de l'arborescence sans ambiguïté mais il peut être long à écrire. Un chemin relatif est en général beaucoup plus court à écrire mais il peut y avoir une ambiguïté sur sa signification, c'est-à-dire sur le noeud de l'arbre auquel il fait référence.

### •• Relatif vs absolu

---

- **Chemin absolu** : Tout chemin qui commence par `/` est un chemin absolu. C'est à dire qu'on suit l'arborescence à partir de la dossier racine du système.
- **Chemin relatif** : À l'inverse, les autres chemins sont relatifs. Un chemin relatif a comme point de départ le répertoire courant.

**Chemin absolu** Un chemin absolu permet d'identifier sans ambiguïté un élément de l'arborescence, qu'il s'agisse d'un répertoire ou d'un fichier. Pour construire un chemin absolu, on commence par l'élément ciblé que l'on préfixe par `/` puis on y ajoute le nom du répertoire parent, c'est-à-dire la branche qui le relie vers la racine. On réitère l'opération jusqu'à atteindre la racine.

### Exemple

Par exemple, la construction du chemin du fichier **photo1.png** avec l'arborescence de notre cours est le suivant :

- Nom du fichier : **photo1.png**,
  - première étape : **Documents/photo1.png**,
  - deuxième étape : **alice/Documents/photo1.png**,
  - troisième étape : **home/alice/Documents/photo1.png**,
  - dernière étape : **/home/alice/Documents/photo1.png**.
- 

Le chemin d'accès `/home/alice/Documents/photo1.png` est dit absolu car il identifie sans ambiguïté le fichier **photo1.png** dans l'arborescence.

### Chemin relatif

Les chemins qui ne commencent pas par la racine sont dit relatifs. Leur signification dépend d'un répertoire de départ appelé répertoire courant.

Lorsque l'utilisateur ouvre une session, le répertoire courant est son répertoire personnel. Au cours de la session, il peut utiliser la commande `cd` pour changer de répertoire courant. Les chemins relatifs peuvent être ambigus. Prenons par exemple le cas du système de fichier proposé par la figure 1.3.2 dans lequel il peut y avoir deux fichiers qui correspondent au chemin relatif `Documents/monfichier.txt`. Si le répertoire courant est le répertoire personnel de **alice**, alors il s'agira de `/home/alice/Documents/monfichier.txt` tandis qu'il s'agira de `/home/bob/Documents/monfichier.txt` si le répertoire courant est le répertoire personnel de **bob**.

### Nommage de fichier

Le nom d'un répertoire ou d'un fichier ne doit pas comporter plus de 255 caractères. Il est de bonne pratique de n'utiliser que des caractères alphabétiques (sans accent) et des chiffres. Il faut préférer utiliser le caractère souligné '\_' à l'espace comme séparateur de mot dans un nom. À savoir aussi, les minuscules et les majuscules sont distinguées.

- `.` fait référence au répertoire courant.
- `..` fait référence au répertoire parent du répertoire courant.
- `~` fait référence au répertoire personnel de l'utilisateur.

*Exemple :*

Si le répertoire courant est le répertoire `Documents` de l'utilisateur **alice**, alors les chemins relatifs suivants font tous référence au même fichier désigné par le chemin absolu `/home/alice/Documents/monfichier.txt` :

- `monfichier.txt`
- `./monfichier.txt`
- `../Documents/monfichier.txt`
- `~/Documents/monfichier.txt`

## Les commandes de navigation

### Se situer

Pour savoir où l'on se situe, c'est-à-dire savoir quel est le répertoire courant, on utilise la commande `pwd` (Print Working Directory) affiche le chemin d'accès absolu du répertoire courant. Elle indique la position dans l'arborescence :

```
1 $ pwd
2 /home/alice
```

Cette commande est informative. Elle ne modifie pas le répertoire courant, ni le système de fichiers.

### Changer de dossier

Pour changer de répertoire courant et donc se déplacer dans le système de fichiers, vous utiliserez la commande `cd` (Change Directory). La commande `cd` change le répertoire courant par le répertoire dont le chemin est passé en argument, qu'il soit relatif ou absolu :

```
1 $ pwd
2 /home/alice
3 $ cd Documents
4 $ pwd
5 /home/alice/Documents
```

## Lister les fichiers et dossier d'un répertoire

Pour lister les fichiers et dossiers d'un répertoire la commande **ls** (List Segments).

```
1 $ pwd
2 /home/alice/
3 $ ls Documents
4 photo1.png photo2.png photo3.png monfichier.txt
```

Si la commande **ls** n'est pas suivi d'un nom de dossier, elle liste les fichiers et dossiers du répertoire courant.

Parmi les options de l'instruction **ls**, on utilise fréquemment :

- **-l** : permet d'avoir davantage d'informations sur les fichiers et dossiers
- **-a** : permet d'afficher les dossiers et fichiers cachés

## Création et suppression de fichiers

Création d'un fichier

Pour créer un fichier on utilise l'instruction **touch** suivi du nom du fichier à créer.

```
1 $ touch nouveau_fichier.txt
2 $ ls
3 Document journal.txt nouveau_fichier.txt
```

### •• Suppression d'un fichier

Pour supprimer un fichier on utilise l'instruction **rm** suivi du nom du fichier à supprimer.

```
1 $ rm journal.txt
2 $ ls
3 Document nouveau_fichier.txt
```

## Création et suppression de répertoires

### •• Création d'un répertoire

La commande **mkdir** (MaKe DIRectory) crée un répertoire et va servir à organiser votre arborescence pour y ranger vos fichiers.

```
1 $ mkdir monRepertoire
2 $ ls
```

Il est possible de créer plusieurs répertoires en même temps :

```
1 $ mkdir monRepertoire1 monRepertoire2
```

### •• Suppression d'un répertoire

La suppression du répertoire se fait avec la commande **rmdir** (ReMove DIRectory) suivie du nom du (ou des) répertoire(s) à supprimer. Cette commande fonctionne si le ou les répertoires à supprimer sont vides. Pour supprimer un répertoire non vide, il faut utiliser la commande **rm** présentée dans le paragraphe suivant.

```
1 $ rmdir monRepertoire2
2 $ ls -l
```





Notez que la commande `rm` peut également supprimer des répertoires : l'option `-r` permet de supprimer récursivement l'arborescence contenue dans le répertoire. Utilisez également l'option `-f` pour



La commande `rm -rf Documents` permet de supprimer l'intégralité du répertoire `Documents` en une commande sans possibilité de revenir en arrière.



## Bilan

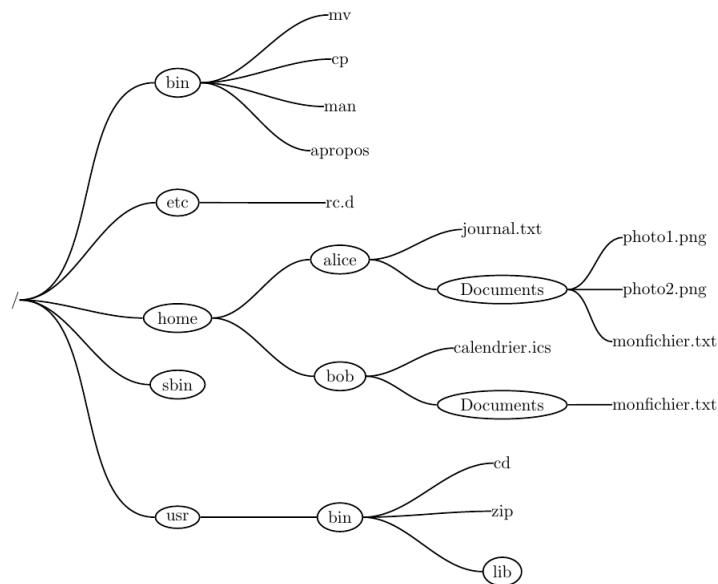
Nous avons vu l'arborescence du système de fichiers. Nous avons identifié des répertoires particuliers comme : la racine, le répertoire personnel, le répertoire courant.

L'accès à un fichier ou un répertoire est décrit par un chemin d'accès. Celui-ci est dit relatif si le point de départ est le répertoire courant, ou absolu si le point de départ est la racine.

Nous avons aussi vu comment créer et supprimer des fichiers et répertoires à l'aide de la ligne de commande.

### Exercice 1

On considère l'arborescence de fichier suivante :



l'utilisateur `alice` se situe dans son répertoire personnel (`/home/alice`).

Quelles instructions lui permettent de se rendre dans le dossier `Documents` et de créer un répertoire nommé `voyage` puis dans ce dossier de créer un fichier `octobre.txt`.

.....

.....

.....

.....

## Copier, déplacer et renommer des fichier

---

### Convention d'ordre des arguments

Lors de la copie ou du déplacement, à retenir on commence par donner le nom de l'ancien (fichier ou dossier) puis le nom du nouveau. Ce qui signifie que le premier argument désigne l'ancien élément, le second désigne le nouveau élément.

### •• Couper / coller

---

Pour déplacer une fichier, (l'action de couper / coller) on utilise la commande `mv` (MoVe)

#### Exemple

Pour déplacer le `fichier1` dans le dossier `Documents`


```
1 $ mv fichier1.txt Documents
```

La commande `mv` permet aussi de renommer une fichier

#### Exemple

Ici on renome le `fichier1.txt` en `fichier2.txt`

```
1 $ mv fichier1.txt fichier2.txt
```

 Si le nom du fichier de destination existe déjà, celui-ci sera écrasé par le fichier source. Il faut être très attentif lors de l'utilisation de la commande `mv` car il est facile de se tromper et ainsi de perdre des informations.

### Copier / coller

#### Définition

Copier `cp`

La commande `cp` (abréviation de CoPy) permet de copier des fichiers :

#### Exemple

La commande ci-dessous permet de copier le repertoire `monRepertoire` dans `monAutreRepertoire`.

```
1 $ cp -R monRepertoire monAutreRepertoire
```

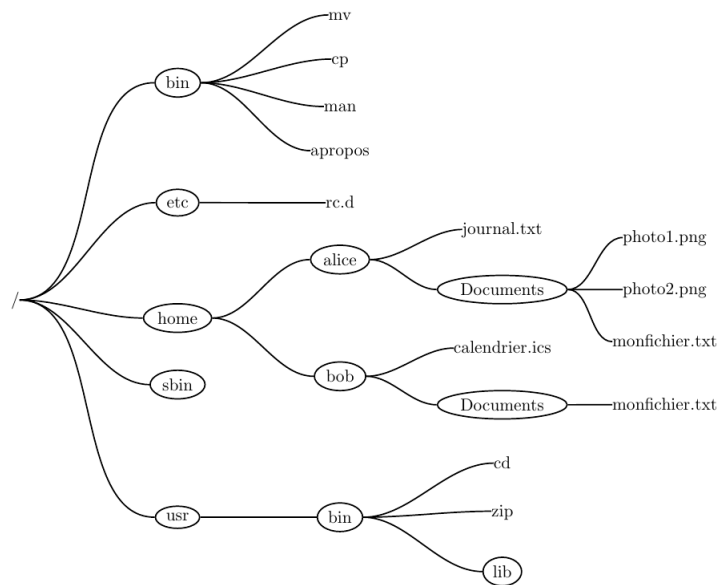


### Bilan

Vous avez vu les principales commandes pour modifier et naviguer dans l'arborescence du système de fichiers. Cette compétence est indispensable à la poursuite du MOOC. Vous trouverez ci-dessous un tableau pour vous rappeler les commandes indispensables :

<code>cd autre_dossier</code>	changer le répertoire courant vers <code>autre_dossier</code> .
<code>mv fichier1 fichier2</code>	renommer ou déplacer le fichier1 vers fichier2.
<code>cp fichier1 fichier2</code>	copier le fichier1 vers fichier2.
<code>touch nom_fichier</code>	créer un fichier nommé <code>nom_fichier</code> .
<code>rm fichier</code>	supprimer le <code>fichier</code> .
<code>mkdir nouveau_dossier</code>	créer un répertoire nommé <code>nouveau_dossier</code> .

**Exercice 2** Copier, déplacer et renommer  
On considère l'arborescence suivante :



L'utilisateur a placé le répertoire courant dans le répertoire personnel d'Alice. Pour chacune des commandes ci-dessous, indiquer les fichiers qui existent encore, ceux qui ont été dupliqués ou renommés. Donner le ou les chemins (s'ils ont été copiés) permettant d'accéder à leur contenu.

```

1  ^^mv journal.txt journal2.txt
2  ^^mv journal.txt /home/bob
3  ^^mv Documents/photo1.png .
4  ^^mv Documents/photo1.png ./Documents/photo3.png
5  ^^Icp Documents/photo1.png ./Documents/photo3.png
6  ^^Icp Documents/photo1.png ./Documents/photo2.png
7  ^^Imv Documents/photo1.png ./Documents/photo2.png

```

.....

.....

.....

.....

.....

.....

.....

.....