Initiation à l'environnement Unix : TP 1

septembre 2024 — Pierre Rousselin

Exercice 1 : Premières commandes et raccourcis claviers

— Une *commande shell* est une suite de mots séparés par des blancs (espaces et/ou tabulations).

— Le nombre de blancs entre les mots n'a aucune importance.

— Le premier mot de la commande s'appelle le *nom de la commande*.

— Les éventuels mots suivants s'appellent les *arguments de la commande*.

1. Tester chacune des commandes suivantes dans un terminal. Décrire en une phrase son utilité, indiquer le nom de la commande, son nombre d'arguments et ses arguments.

a) `date`
b) `cal`
c) `cal 3 2022`
d) `who`
e) `logname`
f) `hostname`
g) `uname`
h) `uname -m -r`
i) `uname -mrs`
j) `echo Hello, world!`
k) `echo  Hello,  world!`

2. Appuyer sur ↑ ou taper Ctrl+P (maintenir la touche Ctrl et appyer sur la touche P) plusieurs fois, jusqu'à l'affichage de la commande `who`. Maintenant appuyer sur ↓ ou taper Ctrl+N jusqu'à afficher la commande `uname -m -r` puis taper sur ↵. Noter à quoi servent ces raccourcis et les apprendre.

3. Appuyer sur Ctrl+L. Noter à quoi sert ce raccourci et l'apprendre.

4. Sans la taper, faire apparaître la commande `cal 3 2022`, *sans l'exécuter* (c'est-à-dire sans appuyer sur entrée).

5. Taper Ctrl+U, noter à quoi sert ce raccourci et l'apprendre.

6. Effacer la ligne de commande courante avec un raccourci clavier puis retaper Ctrl+D. Que s'est-il passé ?

7. Rouvrir un terminal et taper Ctrl+P plusieurs fois. Commenter.

8. Fermer le terminal en utilisant des raccourcis claviers.

--- * ---

Correction de l'exercice 1 :

1.a) `date` : nom de commande `date`, pas d'argument. Sert à afficher la date et l'heure.
b) `cal` : nom de commande `cal`, pas d'argument. Affiche un calendrier.
c) `cal 3 2022` : nom de commande `cal`, deux arguments qui sont les chaînes `"3"` et `"2022"`.
d) `who` : affiche les utilisateurs connectés.
e) `logname` : donne le *login* avec lequel l'utilisateur est connecté
f) `hostname` (non standard) : donne le nom de la machine sur laquelle l'utilisateur est connecté.
g) `uname` : « unix name »
h) `uname -m -r` : 2 arguments, qui sont interprétées par le programme `uname` (et non par le shell) comme des options. Le `m` est pour *machine* (architecture du processeur) et le `r` est pour *release* (numéro de version du noyau).

i) `uname -mrs` : 1 seul argument qui pour le programme `uname` représente 3 options regroupées (le `s` donne le nom du noyau, comme `uname` sans argument).

j) `echo Hello, world!` : 2 arguments. La commande `echo` affiche ses différents arguments puis un saut de ligne.

k) `echo Hello, world` : même chose (il est très important que les étudiants le comprennent).

2. Navigation dans l'historique. Mnémonique : next et previous.

3. Effacer l'écran (en fait, faire remonter la ligne du prompt en haut de l'écran). Mnémonique : Ctrl+L comme le début de *clear*.

4. Le raccourci `C-u` efface la ligne de commande en cours d'édition. Mnémonique : *undo*.

5. Sur une ligne vide, Ctrl+D ferme le terminal (en fait on envoie « fin de fichier »).

6. L'historique est conservé.

7. Ctrl+U puis Ctrl+D.

--- * ---

Exercice 2 : Promenade dans l'arborescence des fichiers

1. Ouvrir un nouveau terminal et entrer la commande suivante, en respectant scrupuleusement sa syntaxe :

```
PS1='$ '
```

2. Entrer la commande `pwd` (pour « *print working directory* », c'est-à-dire, afficher le nom du répertoire courant) et noter ce qui est imprimé à l'écran : c'est le chemin absolu de votre *home*, répertoire personnel.

3. Entrer successivement les commandes `cd ..` (avec un espace entre `cd` et `..`) et `pwd`, jusqu'à ce que le résultat ne change plus. Commenter.

4. Entrer la commande `cd` (sans argument), puis `pwd`. Commenter.

5. Entrer la commande `cd /`, puis `pwd` et `ls`.

6. Entrer la commande `cd /usr/include`. Utiliser la commande `ls`. À quoi semble servir ce répertoire ?

7. La commande `cat` (pour « *concatenate* ») permet d'afficher un ou plusieurs fichiers donnés en argument (à la suite) dans le terminal.

Essayez les commandes suivantes :

```
$ cat stdio.h stdlib.h
$ cat stdio.h
```

La commande `wc` (pour « *word count* ») affiche (dans cet ordre) le nombre de lignes, de mots et de caractères (en fait, d'octets) des fichiers donnés en argument, puis, s'il y en a plusieurs, les sommes de ces nombres pour tous les fichiers. Essayer les commandes

```
$ wc stdio.h stdlib.h
```

8. Entrer les commandes `cd ..`, `pwd` puis `ls`.

9. Entrer les commande `cd share/man`, puis `pwd` et `ls`. Pouvez-vous deviner ce que désignent certains des résultats affichés ?

10. Entrer la commande `ls /bin`. Certains noms vous sont-ils familiers ?

11. Le caractère `~` (qui se lit « tilde ») est saisi au clavier avec la combinaison de touches Alt Gr+2. Entrer la commande `echo ~`, puis la commande `cd ~`. Qu'a fait le shell au caractère `~` ?

12. Représenter les répertoire et fichiers `/`, `/bin`, `/usr`, `/usr/include`, `stdlib.h`, `stdio.h` et `/usr/share/man` sous la forme d'une arborescence.

--- * ---

Correction de l'exercice 2 :

1. Ce changement de prompt est nécessaire pédagogiquement, sans ça les étudiants pensent que `cd` écrit sur le terminal, ce qui n'est pas le cas.
2. C'est long sur les machines de l'institut (des tonnes d'utilisateurs).
3. Essayer de comprendre la notion de répertoire parent. Comprendre que la racine est son propre parent.
4. Retour au répertoire personnel.
5. Chemin absolu, ls sans arguments liste les fichiers du répertoire courant. Parmi ceux-là, repérer le répertoire qui apparaît dans le chemin vers le répertoire personnel des étudiants.
6. Le chemin `/usr/lib` est absolu. La plupart des fichiers ont l'extension `.h` (les autres sont des sous-répertoires).
7. Les commandes `cat` et `wc` sont très importantes pour les scripts et il s'agit de mentionner des fichiers non répertoires.
8. Le répertoire `usr` pour « *Unix System Resources* » contient l'essentiel des programmes (`/usr/bin`), librairies (`/usr/lib`), ressources textuelles (`/usr/share`), ...) du système.
9. `/usr/share/man` contient les pages de manuel. Les répertoires `fr`, `it`, ... correspondent à la langue (cela dépend du système).
10. Le répertoire `/bin` contient les exécutables indispensables à une utilisation minimale du système. Entre autres, on trouve `cat`, `date`, `ls`, ...
11. Ici il faut vraiment insister sur le fait que c'est bien le shell qui interprète le caractère `~` et le remplace par le chemin absolu vers le répertoire personnel. C'est le développement du tilde (*tilde expansion*), premier développement que voient les étudiants.
12.

--- * ---

Exercice 3 : *Find your path*

Find your path est un jeu pédagogique pour apprendre les chemins relatifs et absolus sous Unix. Il a été développé par Thierry Excoffier à l'université Claude Bernard Lyon 1.

Allez jouer à *Find your path* : ouvrez un navigateur web et entrez l'url

`http://demo710.univ-lyon1.fr/FYP/`.

Ne jouez pas trop longtemps non plus, il reste beaucoup à apprendre. Inutile d'aller au-delà du niveau 9 pour l'instant (nous n'avons pas vu les liens).

--- * ---

Exercice 4 : Le *GameShell*

Encore un autre jeu, cette fois développé par Pierre Hyvernat (université Savoie Mont-Blanc) et Rodolphe Lepigre : le GameShell [1].

Nous utiliserons une archive hébergée localement.

1. Entrer les commandes suivantes :

   ```
   $ cd
   $ wget https://www.math.univ-paris13.fr/~rousselin/GameShellLocal/gameshell.sh
   ```

   La première commande permet de s'assurer qu'on est dans son répertoire personnel, la seconde télécharge un fichier sur le web.
2. Lancer le GameShell :

   ```
   $ bash gameshell.sh
   ```

1. Le code source est disponible sur `https://github.com/phyver/GameShell`.

Pour reprendre une partie en cours, on peut utiliser la commande

```
$ ./gameshell-save.sh
```

3. Faire les 3 premières « missions » du GameShell.

--- * ---

Exercice 5 : Créer, copier, déplacer, supprimer

Pendant tout l'exercice, on représentera les répertoires et fichiers mentionnés sous la forme d'une arborescence.

1. Assurez-vous que vous êtes bien dans votre répertoire personnel et listez son contenu.
2. Entrer la commande `mkdir exo_arbo` (pour « *make directory* », c'est-à-dire créer un répertoire). Lister le contenu du répertoire personnel et du répertoire `exo_arbo`.
3. Entrer la commande `mkdir abeilles exo_arbo/tp1 ~/arbres`. Qu'a-t-elle fait ? Parmi ses arguments, lesquels sont des chemins absolus et lesquels sont des chemins relatifs ? (indice : voir le résultat de `echo ~/arbres`).
4. Que fait la commande suivante ?

   ```
   $ mkdir -p vivant/plante/fleur exo_arbo/tp1/exos/ex1/
   ```

5. Le shell `bash` (qui est votre shell par défaut) a une fonctionnalité qui permet de gagner énormément de temps et d'éviter les fautes de frappe : la complétion automatique. Elle se fait avec la touche ⇤⇥ (la touche à gauche de la touche A). Saisir les caractères suivants (la touche tabulation est représentée ci-dessous par `<tab>`) et voir le résultat dans le terminal :

   ```
   $ mkd<tab> vi<tab><tab><tab>roses
   ```

6. Lorsque plusieurs choix sont possibles, la tabulation ne provoque pas de complétion, mais appuyer deux fois de suite sur cette touche liste les choix possibles : essayer avec

   ```
   $ ls a<tab><tab>
   ```

7. La commande `rmdir` (pour « *remove directory* ») permet de supprimer des répertoires.
   a) Tester la commande suivante, et indiquer quel est le message d'erreur. Expliquer.

      ```
      $ rmdir vivant exo_arbo/tp1/exos/ex1
      ```

   b) Avec la bonne commande, supprimer le sous-répertoire `tp1` du répertoire `exo_arbo`.
8. La commande `touch` permet (entre autres choses) de créer des fichiers (normaux) vides. Observer le résultat de la commande (exécutée depuis votre répertoire personnel) :

   ```
   $ touch ~/arbres/hello.c abeilles/truc.txt bidule
   ```

   en tapant

   ```
   $ ls ~/arbres abeilles/ .
   ```

   Remarque : `.` désigne le répertoire courant.
9. La commande `mv` pour « *move* », permet de déplacer ou de renommer des fichiers. Observer avec `ls` le résultat de chacune des commandes suivantes :

   ```
   $ mv arbres/hello.c arbres/bonjour.c
   $ mv abeilles arbres vivant/
   $ mv bidule vivant
   $ mv vivant vie
   ```

   Compléter la phrase suivante : « Si le dernier argument de `mv` est un répertoire existant, alors , sinon `mv` doit avoir arguments et son premier est . »
10. La commande `cp` pour « *copy* », permet de copier des fichiers et des répertoires. Observer le résultat des commandes suivantes :

```
$ cp vie/arbres/bonjour.c salut.c
$ mkdir copies
$ cp salut.c vie/abeilles/truc.txt copies
$ cp vie/bidule exo_arbo copies
$ cp -R vie/bidule exo_arbo copies
$ cp vie copie_vie
$ cp -R vie copie_vie
```

Décrire le fonctionnement de la commande `cp`, selon que son dernier argument est un répertoire existant ou non et que l'option `-R` est présente ou non.

11. Enfin, la commande `rm` (pour « *remove* ») permet de supprimer fichiers et répertoires. Observer le résultat des commandes suivantes :

```
$ rm vie/bidule
$ rm copies
$ rm -r copies
$ rm -R copie_vie
$ rm -i vie/arbres/bonjour.c vie/abeilles/truc.txt
```

12. Supprimer tous les fichiers et répertoires créés pendant cet exercice.

--- * ---

Correction de l'exercice 5 :

1. `cd; ls`

2. `mkdir exo_arbo; ls; ls exo_arbo`

3. Insister sur le fait que `mkdir`, comme énormément d'autres commandes peut avoir plus d'un argument. À cause du développement du tilde par le shell, le chemin `~/arbres` est absolu. Les autres sont relatifs.

4. L'option `-p` pour « parents » crée également les répertoires parents s'ils n'existent pas. Il peut être intéressant de voir que sans cette option, la commande échoue.

5. Remarquer que la complétion est valable à la fois pour les noms de commandes et pour les noms de chemins.

6. Au passage : la complétion se fait toujours au moins jusqu'à l'endroit où les noms de chemins ou de commande coïncident.

7. La commande `rmdir` ne permet pas de supprimer les fichiers non vides.

```
$ rmdir exo_arbo/tp1/exos; rmdir exo_arbo/tp1; rmdir exo_arbo
```

8. En fait, `touch` sert avant tout à mettre à jour les dates associées aux fichiers existants.

9. Si le dernier argument de `mv` est un répertoire existant, alors ses premiers arguments sont déplacés dans ce répertoire, sinon `mv` doit avoir 2 arguments et son premier est renommé en son second.

10. Sans l'option `-R` (pour « récursif »), la commande `cp` ne permet pas de copier les répertoires. Puis, si le dernier argument de `cp` est un répertoire existant, ses premiers sont copiés dans ce répertoire (avec le même nom de base). Sinon, `cp` doit avoir deux arguments (éventuellement en plus de `-R`) et le premier est copié avec comme nom de la copie le second.

11. La commande `rm` ne permet pas de supprimer les répertoires, sauf si `-r` ou `-R` (qui sont exactement équivalentes) sont présentes. L'option `-i` pour (« interactif ») permet de demander confirmation pour chaque fichier et/ou répertoire. Insister sur le fait qu'il n'y a pas de corbeille ! Les fichiers sont bels et bien perdus (sauf opération complexe consistant à chercher les données anciennement associées au fichier).

--- * ---

Exercice 6 : GameShell, suite

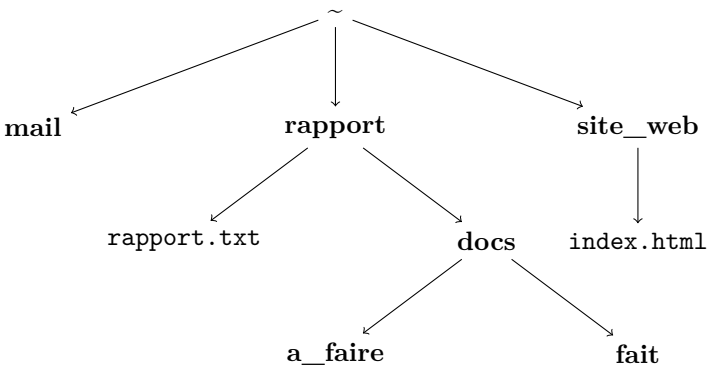
Accomplir les missions 4 à 6 du GameShell.

--- * ---

La commande `touch` crée un fichier vide par argument (un chemin) qui lui est fourni, si ceux-ci ne désignent pas des fichiers déjà existants.

Exercice 7 : Arborescence

Ci-dessous est représentée une arborescence. Le ~ représente le répertoire personnel de l'utilisateur. Les répertoires sont en **gras**, les fichiers normaux sont en `mono-châsse`.



À partir du répertoire personnel faîtes les actions suivantes (il existe plusieurs solutions possibles) :

1. Créer cette arborescence (répertoires et fichiers normaux). Utiliser un éditeur de texte (par exemple `nano` ou `gnome-text-editor`) pour créer les fichiers normaux et y entrer du contenu (peu importe lequel).
2. Aller directement dans `~/rapport/docs/a_faire`
3. De là, passer dans `~/rapport/docs/fait` et y copier le fichier `rapport.txt`. Rappel : le répertoire courant peut être désigné par `.` (un point).
4. Renommer cette copie `rapport_copie.txt`
5. Revenir dans `~/rapport`
6. Sans changer de répertoire, regarder avec la commande `cat` le contenu de `index.html`
7. Sans changer de répertoire, lister le contenu du répertoire **site_web**.
8. Revenir dans votre répertoire personnel et supprimer les répertoires `rapport`, `site_web` et `mail` ainsi que tout ce qu'ils contiennent.

--- * ---

Correction de l'exercice 7 : On commence par créer les répertoires et les fichiers. Plusieurs solutions. La touche `M` pour « Méta » est en pratique la touche `Alt` ou la touche `Esc`.

```
$ cd # pour s'assurer qu'on est dans home
$ mkdir mail rapport site_web
$ mkdir rapport/docs
$ cd rapport/docs # penser au raccourci M-.
$ mkdir a_faire fait
$ cd .. # on est dans rapport
$ touch rapport.txt
$ touch ~/site_web/index.html
```

Autre possibilité pour les répertoires en utilisant l'expansion d'accolades et l'option `-p` de `mkdir` qui crée, si besoin, les répertoires intermédiaires :

```
$ cd # pour s'assurer qu'on est dans home
$ mkdir -p mail rapport/docs/{a_faire,fait} site\_web
$ touch rapport/rapport.txt site_web/index.html
```

Les « brace expansions » sont une spécificité de bash et ne sont pas dans la norme POSIX, donc interdits dans les scripts. Petite remarque au passage : si le programme `tree` est installé, c'est un bon moment pour l'utiliser et vérifier que tout fonctionne.

Ensuite pour le reste du sujet :

```
$ cd # pour s'assurer qu'on est dans home
$ cd rapport/docs/a_faire # penser à TAB souvent
$ cd ../fait
$ cp ~/rapport/rapport.txt .
$ cat ~/site_web/index.html
$ ls ~/site_web
$ cd
$ rm -r site_web rapport mail
```

On peut rappeler aussi que `cd -` permet de retourner dans l'ancien répertoire de travail (`OLDPWD`).

--- * ---

Exercice 8 : La commande `type`

Les commandes shell sont rangées en 4 catégories :

- les *commandes internes* (ou primitives) du shell sont celles qui sont exécutées par le shell lui-même, sans utiliser d'autre programme ;
- les *commandes externes* sont celles qui font appel à un autre programme directement ;
- les *alias* sont des raccourcis, souvent créés par l'utilisateur pour d'autres commandes ;
- enfin, les *fonctions shell* sont des suites d'instructions écrites en shell.

1. Voir la sortie de la commande

```
$ type rm
```

Est-ce une commande interne ou externe ? Si elle est externe, dans quel fichier est son programme ?

2. Voir la sortie de la commande

```
$ type pwd cd
```

Est-ce que ce sont des primitives ? D'après vous, pourquoi ?

3. Donner le type des commandes `ls`, `cp`, `rm`, `mkdir`, `echo`, `cat`, `wc`, `date`, `cal` et `type`.

4. Quel semble être le répertoire qui contient la plupart des programmes exécutables sur cette machine ?

--- * ---

Correction de l'exercice 8 :

1. Normalement, `rm` devrait être une commande externe qui est le plus souvent situé à `/bin/rm` ou `/usr/bin/rm` (dépend du système et du shell).

2. `pwd` et `cd` sont de toute façon des primitives car il s'agit de donner des informations sur l'état du shell lui-même ou changer cet état.

3. La commande `type` est toujours une primitive du shell (doit connaître l'état du shell pour fonctionner). Encore une fois, cela dépend des shell, mais il est fort probable que `echo` soit une primitive du shell (bien que `/usr/bin/echo` puisse exister !) pour des raisons de performance, tandis que les autres commandes mentionnées sont probablement des commandes externes. Les commandes « hachées » sont simplement des commandes externes dont le shell a sauvegardé le chemin (il ne la cherche plus dans les répertoires du `PATH` (on en reparlera un peu plus tard).

4. Dépend des systèmes. Sur les machines de l'institut, la plupart des utilitaires sont dans `/usr/bin` et `/bin` est un lien symbolique vers `/usr/bin`.

--- * ---

La commande `man` fournit de l'aide *pour les commandes externes*. Pour les primitives de `bash`, on peut utiliser la commande `help`.

Exercice 9 : `man`, la commande la plus importante de toutes

1. Entrer la commande `man ls`. À quoi servent les options `-l` et `-a` ? Taper sur la touche `q` pour sortir de l'aide et les essayer.
2. À l'aide du manuel, dire à quoi sert l'option `-f` de la commande `rm` et comment on peut supprimer un fichier dont le nom commence par un tiret (comme par exemple `-f`).
3. À l'aide du manuel, décrire l'utilité de l'option `-k` de la commande `man`. La tester pour lister les navigateurs web installés (et documentés) sur le système avec
   ```
   $ man -k 'web browser'
   ```
4. À l'aide de la commande `help`, obtenir de l'aide sur les commandes `echo` et `type` intégrées au shell `bash`.
5. Voir la page de manuel de `touch`. À quoi sert ce programme, si ce n'est à créer des fichiers vides ?
6. Revoir la page de manuel de `man`. Dans quelle section sont les programmes et les commandes du shell ? Dans quelle section sont documentées les bibliothèques (comme la bibliothèque standard du C) ? Expliquer la différence entre les deux commandes suivantes :
   ```
   $ man 1 printf
   $ man 3 printf
   ```
7. Dans la page de manuel de `mv`, observer les deux premières lignes de la partie « SYNOPSIS ». Que signifient les crochets ? les points de suspension ? Si besoin, se reporter au manuel de `man`.

--- * ---

Correction de l'exercice 9 : Juste quelques remarques : rassurer les étudiants sur le fait qu'il est rare de comprendre entièrement une page de manuel. Certains passages sont très spécialisés. Pour d'autres, ils les comprendront plus tard dans le semestre.

--- * ---

Les caractères jokers pour créer des *motifs shell* sont seulement :

`*` : correspond à toute chaîne de caractère (éventuellement vide) sauf les chaînes commençant par le caractère `.` dans le cas où `*` est en début de chaîne ;

`?` : correspond à un caractère quelconque ;

`[ ]` : correspond à un (et un seul) caractère à l'intérieur des crochets. On peut utiliser des intervalles, comme dans `[a-z]` qui correspond à une seule lettre minuscule ou dans `[0-5]` qui correspond à un seul chiffre entre 0 et 5. On peut inverser la recherche en faisant commencer l'intervalle par `!` : par exemple `[!0-9]` correspond à un caractère qui est tout sauf un chiffre.

Plus d'information dans `man bash` à la rubrique « Développement des chemins » (*pathname expansion*).

Exercice 10 : Les caractères jokers et le développement des chemins

1. Créez le répertoire `tp_joker` dans votre répertoire personnel. Déplacez-vous dans ce répertoire. Créez les fichiers (vides) suivants : `annee1 Annee2 annee4 annee45 annee41 annee510 annee_saucisse annee_banane bonbon`

2. Essayer de prévoir le résultat des commandes suivantes, puis les tester :

```
$ echo *
$ echo *_*
$ echo [ab]*
$ echo [!ab]*
$ echo c*
$ echo ??????
```

3. Afficher le nom de tous les fichiers dont le nom :

a) se termine par `5`
b) commence par `annee4` ;
c) commence par `annee4` et a 7 caractères ;
d) commence par `annee` et dont le sixième caractère n'est pas un chiffre ;
e) commence par `annee` et dont le septième caractère n'est pas un chiffre ;
f) contient la chaîne `ana` ;
g) commence par `a` ou `A` ;
h) a pour avant-dernier caractère `4` ou `1` ;

4. Lister les fichiers dont le nom commence par `std` et se termine par `.h` dans le répertoire `/usr/include`.

5. Lister les fichiers dont le nom commence par une lettre `w` majuscule ou minuscule et se termine par `.h` et se trouvant dans un répertoire arrière-petit-enfant de la racine.

--- * ---

Correction de l'exercice 10 :

1. Essayer de prévoir le résultat des commandes suivantes, puis les tester :

```
$ echo * # affiche les noms des fichiers non cachés
$ echo *_* # affiche les noms des fichiers dont le nom contient un souligné
$ echo [ab]* # affiche les noms de fichier commençant par a ou b
$ echo [!ab]* # affiche les noms de fichier ne commençant pas par a ou b
$ echo c* # affiche c* car aucun fichier ne correspond au motif...
$ echo ?????? # affiche les noms de fichier de 6 caractères
```

Insister sur le fait que c'est le shell (et non la commande `echo` ou `ls`) qui développe les noms de chemins et que si aucun fichier ne correspond au motif, le shell le laisse inchangé.

2. a) `echo *5`
b) `echo annee4*`
c) `echo annee4?`
d) `echo annee[0-9]*!`
e) `echo annee?[0-9]*!`
f) `echo *ana*`
g) `echo [aA]*`
h) `echo *[41]?`

3. `echo /usr/include/std*.h`

4. `echo /*/*/*/[wW]*.h`

--- * ---

Exercice 11 : GameShell encore et toujours

Accomplir les missions 7 à 11 du GameShell. Bien sûr vous avez le droit de continuer à avancer en autonomie dans ce jeu.

--- * ---

Exercice 12 : La guerre des éditeurs

Les éditeurs de texte permettent de créer, modifier et enregistrer les fichiers qui contiennent du code source (par exemple, en C ou en shell, des fichiers de configuration, des documents .tex, etc). Ils ont des fonctionnalités plus ou moins complexes (recherche de texte, remplacement de texte, etc). Les informaticiens passent le plus clair de leur temps « dans leur éditeur de texte. » Ce qui fait qu'ils peuvent nouer une relation assez passionnée avec ce programme.

Nous avons parlé dans les exercices précédents de deux éditeurs de texte :

— `gnome-text-editor` : à l'avantage d'être très simple à prendre en main, mais il n'est pas très puissant et surtout, il ne tient pas dans un terminal, ce qui est rédhibitoire pour, par exemple, administrer une machine distante qui n'a pas d'environnement graphique.

— `nano` : fonctionne dans un terminal, mais est encore trop simple.

Parmi tous les éditeurs de texte disponibles (sûrement des centaines...), il y en a deux qui ont certainement le plus déchaîné les passions : emacs (créé par Richard Stallman) et vi (créé par Bill Joy). Aujourd'hui, plus grand monde n'utilise vi, mais l'éditeur Vim (pour *vi improved*, par Brian Moolenaar) est son successeur le plus populaire.

Ces deux éditeurs sont incroyablement puissants et très différents. Ils sont plutôt délicats à prendre en main au début et une vie entière ne suffit pas à les maîtriser entièrement, mais le jeu en vaut la chandelle : lorsqu'on sait les utiliser, on gagne énormément de temps et éditer des fichiers devient vraiment plus agréable.

— Vous pouvez aller lire `https://fr.wikipedia.org/wiki/Guerre_d%27%C3%A9diteurs` aussi, pour les anglophones, `https://xkcd.com/378/`

— Il faut choisir... emacs ou Vim ? On va donner une chance aux deux (en vérité, il serait bien de connaître au moins un peu les deux).

  — Pour vim, taper la commande `vimtutor` dans le terminal et suivre les instructions (ça dure environ 20 minutes). Pour lancer vim dans une fenêtre plutôt que dans le terminal, utiliser la commande `gvim`.

  — Pour emacs, taper la commande `emacs`, puis le raccourci `C-h t` (contrôle et h en même temps, puis t). Pour lancer emacs dans le terminal, plutôt que dans une fenêtre graphique, utiliser `emacs -nw`

`--- * ---`