Initiation à l'environnement Unix : TP 1

septembre 2022 — 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 la flèche du haut ou taper `C-p` (la touche `Ctrl` en même temps que la touche `p`) plusieurs fois, jusqu'à l'affichage de la commande `who`. Maintenant appuyer sur la flèche du bas ou taper `C-n` jusqu'à afficher la commande `uname -m -r` puis taper sur entrée. Noter à quoi servent ces raccourcis et les apprendre.

3. Appuyer sur `C-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 `C-u`, noter à quoi sert ce raccourci et l'apprendre.

6. Faire apparaître de nouveau, toujours sans la taper ni l'exécuter, la commande `uname`. Puis taper `C-d`. Que s'est-il passé ?

7. Effacer la ligne de commande courante avec un raccourci clavier puis retaper `C-d`. Que s'est-il passé ?

8. Rouvrir un terminal et taper `C-p` plusieurs fois. Commenter.

9. Fermer le terminal en utilisant des raccourcis claviers.

`--- * ---`

Exercice 2 : Promenade dans le système de 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. La commande `wc` (pour « *word count* ») affiche (dans cet ordre) le nombre de lignes, de mots et de caractères des fichiers donnés en argument, puis, s'il y en a plusieurs, les sommes de ces nombres pour tous les fichiers.
   À l'aide des commandes `cat` et `wc` suivies de l'argument qui convient, afficher le contenu et le nombre de lignes du fichier `stdio.h`. Donner le nombre total de lignes des fichiers `stdlib.h` et `stdio.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 répertoires et fichiers mentionnés dans l'exercice sous la forme d'une arborescence (c'est-à-dire comme un arbre généalogique).

--- * ---

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

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
   ```
   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 tp_shell` (pour « *make directory* », c'est-à-dire créer un répertoire). Lister le contenu du répertoire personnel et du répertoire `tp_shell`.
3. Entrer la commande `mkdir abeilles tp_shell/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 tp_shell/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 tabulation (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 tp_shell/tp1/exos/ex1
      ```
   b) Avec la bonne commande, supprimer le sous-répertoire `tp1` du répertoire `tp_shell`.
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 tp_shell copies
$ cp -R vie/bidule tp_shell 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.

--- * ---

Exercice 6 : GameShell, suite

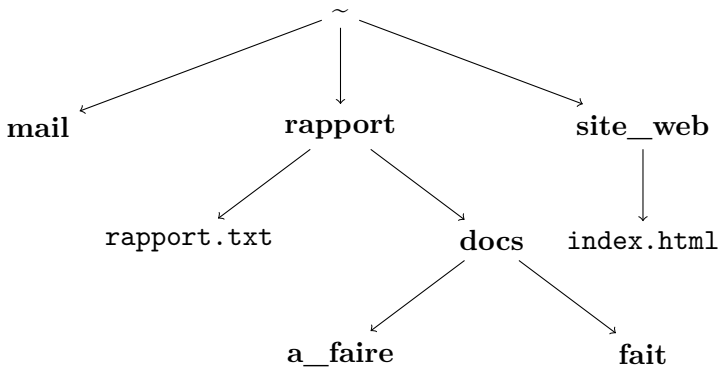
Accomplir les missions 4 à 6 du GameShell.

--- * ---

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 `gedit`) 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 ~ et supprimer toute cette arborescence.

`--- * ---`

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 ?

--- * ---

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 tirer (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.
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`.

--- * ---

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) contient la chaîne `ana` ;

f) commence par `a` ou `A` ;

g) 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.

--- * ---

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 :

— `gedit` : 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`

`--- * ---`