Ligne de commande UNIX

Philippe Veber

16 octobre 2023

Principaux concepts

Système d'exploitation

Un **système d'exploitation** est un ensemble de programmes sur lesquels les applications que l'on utilise s'appuient pour :

- obtenir du temps processeur, de la mémoire
- accéder aux périphériques de stockage, d'entrée (clavier, souris, scanner *etc*) ou de sortie (imprimante, carte graphique, carte son *etc*).
- lancer d'autres programmes

UNIX est une famille de systèmes d'exploitation comprenant notamment Linux et MacOS X. Ces systèmes présentent de grandes similarités.

Le terminal

Aussi appelé *shell*, programme permettant d'interagir avec le système d'exploitation, à l'aide de **commandes**.

Travailler en ligne de commande, c'est donc interagir avec le système d'exploitation (p. ex. pour gérer des fichiers) en entrant des commandes (plutôt qu'avec une souris).

Le terminal est un **interpréteur**, i.e. un programme répétant la boucle :

- lire une commande
- exécuter la commande
- afficher le résultat

Lancer un terminal

- sous Linux Debian/Ubuntu: `CTRL` + `ALT` + `T`
- sous MacOSX: Applications > Utilitaires > Terminal

Pourquoi (diable) utiliser un terminal

- interface universelle pour utiliser des programmes d'analyse
- il existe un grand nombre de commandes, et celles-ci peuvent être combinées pour réaliser des traitements très sophistiqués
- permet d'automatiser ("scripter") ces traitements
- fonctionne à l'identique à distance

L'invite

- court message précédant l'emplacement où l'on peut entrer une commande
- apporte généralement des informations utiles
- se termine par un caractère `$` qui signale le début de la ligne de commande
- après avoir tapé sa commande, on l'envoie à l'interpréteur avec `ENTRÉE`
- `CTRL+D` pour terminer

Un premier exemple de commande :

```
$ date
jeu. 06 oct. 2022 10:28:09 CEST
```

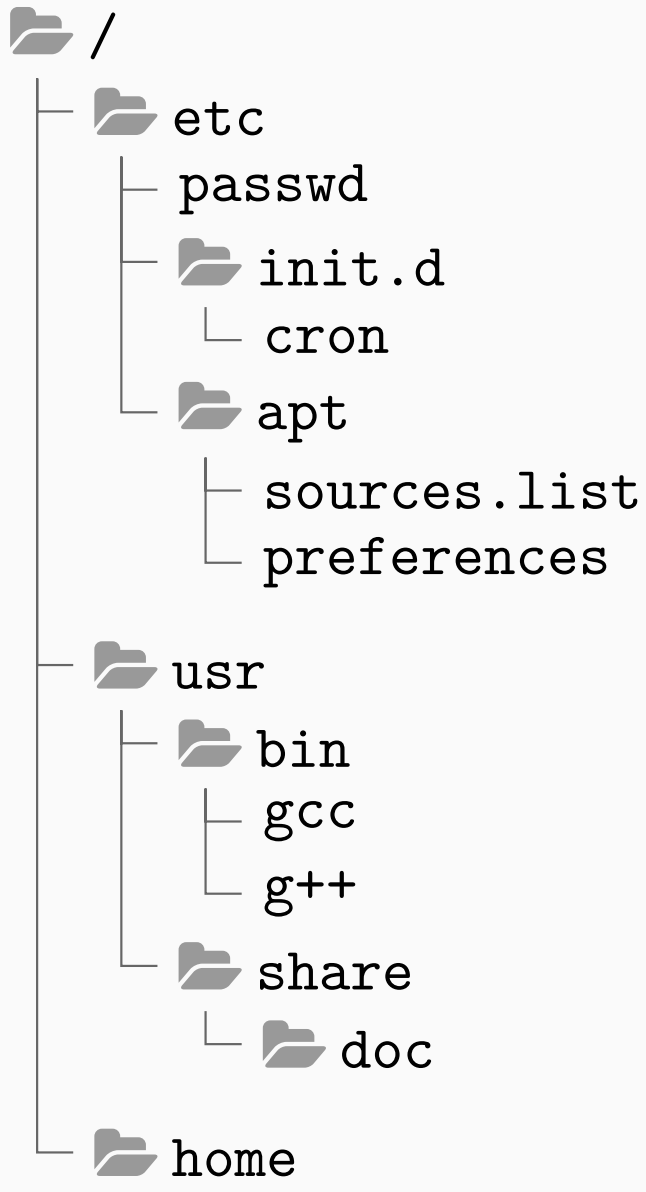
Système de fichiers

Fichier suite d'octets formant un document. Chaque fichier a un nom qui permet de le désigner

Répertoire structure contenant des fichiers ou d'autres répertoires. Il existe un répertoire particulier appelé **répertoire racine** et noté /, à partir duquel on peut retrouver n'importe quel fichier.

Système de fichiers organisation hiérarchique des fichiers accessibles au système (on parle aussi d'arborescence).

Chemin



Notation pour désigner un fichier dans l'arborescence.

Par exemple pour arriver au fichier cron on suit le chemin

$$\text{Racine} \to \text{etc} \to \text{init.d} \to \text{cron}$$

que l'on dénote

$$/\text{etc}/\text{init.d}/\text{cron}$$

Examiner le système de fichiers

La commande `ls` permet de `lister` le contenu d'un répertoire

```
$ ls /
bin  boot  cdrom  dev  etc  home  lib  [...]
```

On voit que la commande `ls` prend un **argument**, à savoir le chemin que l'on veut examiner

Examiner le système de fichiers

La commande `tree` affiche une arborescence

```
$ tree .
.
├── img
│   ├── asymetric_cryptography_step1.png
│   └── asymetric_cryptography_step2.png
├── intro_R.Rmd
├── Makefile
└── intro_unix.md
```

Répertoire courant

À tout moment, le terminal est "positionné" sur un répertoire, appelé **répertoire courant**. Cette information est utilisée par les commandes de différentes manières.

`pwd` (*print working directory*) affiche le répertoire courant :

```
$ pwd
/home/pveber/w/formation-ngs-cnrs
```

Changer le répertoire courant

La commande `cd` (*change directory*) permet de changer le répertoire courant :

```
$ cd /
$ pwd
/
```

On peut appeler `cd` avec l'argument `-` pour retourner à la position précédente.

Exercice : vérifier la fonctionnalité de `cd -`

Le symbole .

Le symbole `.` représente le répertoire courant :

```
$ pwd
/home/pveber/w/formation-ngs-cnrs
$ ls /home/pveber/w/formation-ngs-cnrs
intro_unix.md   intro_unix.pdf   Makefile
$ ls .
intro_unix.md   intro_unix.pdf   Makefile
$ ls
intro_unix.md   intro_unix.pdf   Makefile
```

On voit que si on appelle `ls` sans argument, la commande affiche le contenu du répertoire courant par défaut

Symbole `..`

Le symbole `..` représente le **répertoire parent**, c'est-à-dire le répertoire contenant le répertoire courant.

Par exemple si le répertoire courant est `/A/B`, `..` désigne le répertoire `A`.

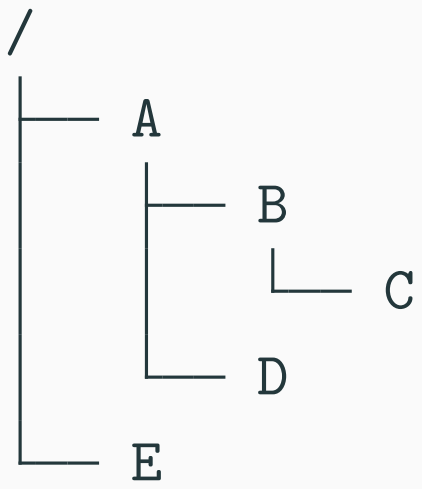
Pour déplacer le répertoire courant vers A, il suffit d'écrire :

```
$ pwd
/A/B
$ cd ..
$ pwd
/A
```

Chemins relatifs

Un chemin commençant par `.` ou `..` est appelé **relatif** et **absolu** s'il commence par `/`

Exercice: déterminer le chemin relatif depuis B de A, C, D et E



Auto-completion

On peut demander au shell de **compléter** le chemin que l'on est en train de taper avec la touche **tabulation**.

Lorsqu'on tape sur tabulation

- s'il n'y a qu'un seul chemin possible pour compléter ce que vous avez déjà tapé, le shell complète
- sinon il ne fait rien mais affichera toutes les possibilités si vous tapez à nouveau sur tabulation

Auto-completion

Avantages :

- plus rapide
- plus sûr (pas de risque de typo)
- permet de détecter un fichier absent **avant** de lancer la commande

La complétion peut aussi fonctionner avec les options d'une commande

Exercice: Lister le contenu du répertoire `/usr/lib/xorg/modules/drivers/` en tapant le moins de lettres possible. Combien en faut-il ?

Exercice: Affichez toutes les options de `ls` avec la complétion.

Option d'une commande

On peut modifier le comportement d'une commande à l'aide d'**options** :

```
$ ls -l
total 184
-rw-rw-r-- 1 pveber pveber   4366 oct.   6 07:24 intro_unix.md
-rw-rw-r-- 1 pveber pveber 170774 oct.   6 07:24 intro_unix.pdf
-rw-rw-r-- 1 pveber pveber     57 oct.   6 07:01 Makefile
```

Ici l'option `-l` indique à `ls` de fournir plus de détails sur les fichiers listés.

Commandes pour la gestion des fichiers et répertoires

Créer un répertoire

On utilise la commande `mkdir`. Dans l'exemple suivant les # introduisent des commentaires, c'est-à-dire du texte non interprété par le terminal.

```
$ mkdir site              # création du répertoire
$ ls
ligne-de-commande.md  site  uikit.html
$ ls site                 # le répertoire créé est vide initialement
$ touch site/index.html   # on crée un fichier vide dans le nouveau
$ ls site                 # répertoire
index.html
```

Au passage on a vu la commande `touch` qui permet entre autres de créer un fichier vide.

Exercice

En utilisant `mkdir` et **sans utiliser** `cd`, reproduire l'arborescence suivante :

```
./exercice
├── A
│   ├── B
│   │   └── C
│   └── D
└── E
```

Aller dans le répertoire B, puis déplacez-vous successivement vers A, C, D et E en une seule commande `cd`. Vous pouvez facilement revenir à B à chaque fois en utilisant la commande `cd -`

Copier un fichier ou un répertoire

On utilise la commande `cp` qui prend deux arguments

```
$ ls
ligne-de-commande.md  site  uikit.html
$ cp site/index.html .
$ ls
index.html  ligne-de-commande.md  site  uikit.html
```

On voit sur cet exemple l'utilité de la notation `.` : elle nous permet de dire "copie le fichier dans le répertoire où je me trouve". Si on souhaite copier un répertoire il faut ajouter l'option `-r`

```
$ cp -r mon_repertoire /ma/clef/usb
```

Déplacer/renommer un fichier ou un répertoire

Dans la mesure où le nom d'un fichier est aussi son chemin dans l'arborescence, il n'y a pas de différence en UNIX entre renommer et déplacer un fichier. La commande pour cette opération s'appelle `mv`:

```
$ ls
 index.html   ligne-de-commande.md   site   uikit.html
$ mv index.html essai_index.html
$ ls
 essai_index.html   ligne-de-commande.md   site   uikit.html
```

Effacer un fichier ou un répertoire

On utilise la commande `rm` pour effacer un fichier. Pour effacer un répertoire *vide*, on peut utiliser la commande `rmdir`. Pour effacer un répertoire non-vide, il faut utiliser `rm` avec l'option `-r` (pour *récursif*).

```
$ rm essai_index.html
$ rmdir site
rmdir: impossible de supprimer 'site': Le dossier n est pas vide
$ rm -r site
$ ls
ligne-de-commande.md  uikit.html
```

ATTENTION Un fichier supprimé avec `rm` est réellement supprimé, pas simplement "déplacé à la corbeille", il faut donc être prudent · e quand on s'en sert, tout particulièrement avec l'option `-r`.

Accès aux pages de manuel

La commande `man` permet d'accéder aux pages de manuel de nombreux programmes. Ces pages d'aide ne sont en général pas très didactiques, mais elles fournissent une référence très précise sur le comportement des commandes.

```
$ man ls
```

Pour quitter la page de manuel, tapez `q`.

Récapitulatif des commandes vues

Commande	Effet
pwd	imprime le répertoire courant dans le terminal
ls *path*	lister les fichiers d'un chemin
cd *path*	changer le répertoire courant
mkdir *path*	créer un répertoire
touch *path*	créer un fichier vide
cp [-r] *src dst*	copier un fichier ou un répertoire
mv *src dst*	renommer ou déplacer
rm [-r] *chemin*	effacer un fichier ou un répertoire
man *commande*	afficher la documentation sur une commande

Inspecter le contenu d'un fichier

Format d'un fichier (1/2)

Prescription sur la structure du contenu d'un fichier, permettant de décoder l'information qui s'y trouve.

On parle de format :

- texte quand il repose sur le codage des caractères alpha-numériques ASCII (CSV, FASTA, HTML)
- binaires pour des codages spécialisés (JPG, ZIP)

Format d'un fichier (2/2)

Le format d'un fichier est souvent suggéré dans son nom, par la présence d'une **extension** :

- partie du nom du fichier à droite du point le plus à droite (doc.2.**pdf**)
- attention l'extension est déclarative et non vérifiée

On peut utiliser la commande `file` qui inspecte le contenu d'un fichier pour déterminer le format :

```
$ file intro_unix.pdf
intro_unix.pdf: PDF document, version 1.5
```

Voir le début d'un fichier

```
$ head /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
[...]
$ head -n 1 /etc/passwd
root:x:0:0:root:/root:/bin/bash
```

Voir la fin d'un fichier

On utilise la commande `tail`

```
$ tail /etc/passwd
pulse:x:124:131:PulseAudio daemon,,,:/run/pulse:/usr/sbin/nologi
gnome-initial-setup:x:125:65534::/run/gnome-initial-setup/:/bin/
hplip:x:126:7:HPLIP system user,,,:/run/hplip:/bin/false
[...]
```

Visualiser le contenu d'un fichier

- on peut utiliser la commande `less`
- visualisation interactive (utiliser les flèches)
- `q` pour quitter
- beaucoup de fonctionnalités utiles (recherche, coloration syntaxique)

Retour sur les commandes

Syntaxe des commandes simples

`NOM-COMMANDE [OPTIONS] [ARGUMENT1 [ARGUMENT2 [...]]]`

Quelques exemples :

```
$ cp -i a.txt b.txt              # demande confirmation avant d'écraser un éventuel fichier
$ rm --recursive dir             # forme longue de rm -r
$ rm -r --interactive=always tmp # efface récursivement en demandant confirmation à chaque
                                 # nouveau sous-répertoire
```

- option en forme courte avec un tiret vs forme longue avec 2 tirets
- les options peuvent venir avec une valeur, il y a alors deux syntaxes soit `--option=value` ou `--option value` (plus courant).

Wildcard

On peut désigner un ensemble de fichiers à l'aide du caractère `*` appelé *wildcard* (ou joker).

`*` représente toute séquence de caractères (même vide)

Exemples :

```
$ ls *
$ ls *.pdf
$ ls *_*.jpg
$ ls abc_*.a*
```

Exercice: donner des exemples de noms correspondant à chacun des cas ci-dessus

Les commandes sont des fichiers exécutables

```
$ file /bin/ls
/bin/ls: ELF 64-bit LSB pie executable [...]
$ /bin/ls *.pdf
[...]
```

- quand on appelle une commande, on lance un fichier exécutable
- il existe un mécanisme pour retrouver le chemin du fichier exécutable à partir de son nom

Redirection

Si une commande produit un affichage sur le terminal, on peut orienter celui-ci vers un fichier. On appelle cela une **redirection**.

Exemple :

```
$ ls *.fastq > liste_fichiers_fastq
```

Cette commande crée un nouveau fichier (et écrase au passage tout fichier existant à ce nom)

Organisation simple d'un projet d'analyse de données

Éditer un fichier

Dans une analyse de données on manipule différents types de documents :

- des programmes ou scripts
- des données (tableaux, séquences, *etc*)
- de la documentation
- des graphiques

Une partie de ces documents est écrite dans des formats **texte**, ie comme du texte brut, sans formatage typographique.

Éditer un fichier

Pour créer ces fichiers, on peut utiliser un **éditeur de texte**.

Exercice : essayer `gedit` en lançant la commande

```
$ gedit &
```

Entrez un texte, sauvegardez-le dans un fichier et vérifier le contenu du fichier à l'aide de commandes appropriées dans le terminal.

Notion de scripts

Pour garder trace des commandes utilisées dans une analyse, on les conserve dans un (ou plusieurs) fichiers au format texte appelés **scripts**.

Un script de commandes du terminal doit avoir (convention) l'extension `.sh`

L'autre intérêt du script est que l'on peut "rejouer" facilement l'analyse

La structure du répertoire d'analyse

Il **FAUT** organiser vos fichiers de manière à **SÉPARER** ce que vous avez écrit et ce qui est généré par vos scripts.

En pratique on crée un répertoire appelé par exemple `res` qui est le seul endroit où vos commandes pourront générer des fichiers.

La structure du répertoire d'analyse

Voici une trame pour débuter

1. un fichier `script.sh` dans lequel consigner les commandes effectuées par votre analyse
2. `res`, **le seul** répertoire où vos programmes écriront des résultats.
3. `data`, un répertoire contenant les données brutes
4. `README`, un fichier texte contenant une description résumée du projet, un guide d'utilisation rapide et des informations utiles (auteurs/laboratoires impliqués, article associé à l'analyse, références biblio importantes ...)
5. `LICENSE`, le texte d'une licence pour le code (Cecill-B) par défaut, mais il faut prendre le temps de se renseigner avant de choisir.

Travailler à distance

ssh

La commande `ssh` permet de lancer un nouveau terminal s'exécutant sur une autre machine accessible sur le réseau. Il faut disposer au préalable d'un compte sur la machine distante.

Il faut fournir :

- l'adresse de la machine distante
- un login
- un moyen pour s'authentifier

On utilise ensuite ce terminal exactement comme un terminal sur sa machine locale.

L'authentification

Elle est possible via deux méthodes :

- mot de passe
- **clefs cryptographiques**

Dans les deux cas, elle permet au serveur distant de vous authentifier, mais pas d'authentifier le serveur distant pour vous.

L'authentification par clef est plus sûre et plus pratique, mais demande une mise en place

Identification par clef publique : le principe

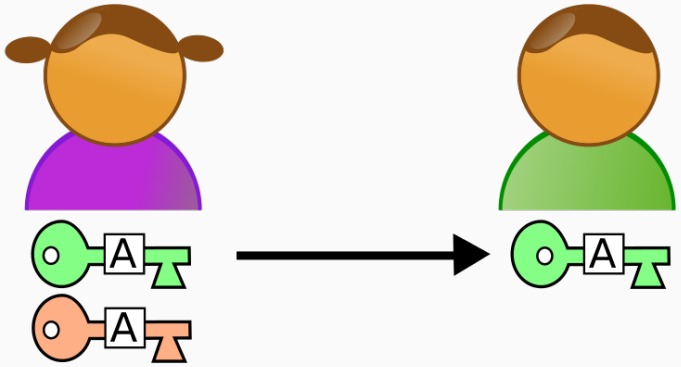


Figure 1: Création des clefs et partage de la clef publique

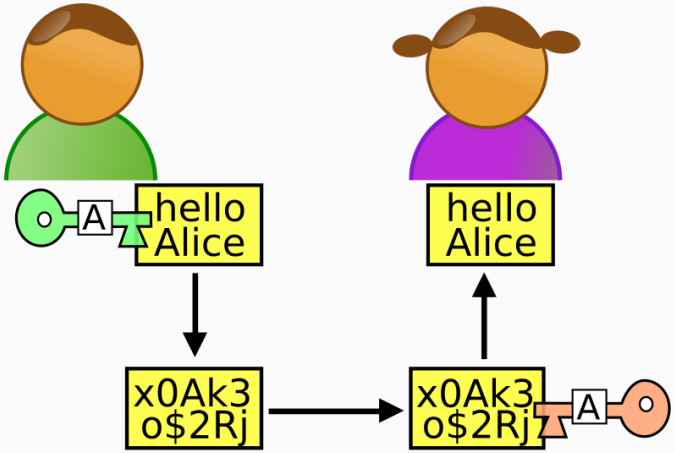


Figure 2: Authentification du propriétaire de la clef publique

[wikipedia]

Création des clefs SSH

Lancer

`ssh-keygen`

Par défaut les clefs ssh ne sont pas protégées par mot de passe (ou *passphrase*) mais il est préférable d'en ajouter.

Cette commande crée :

- la clef privée dans `~/.ssh/id_rsa`
- la clef publique dans `~/.ssh/id_rsa.pub`

Copie des clefs ssh sur le serveur distant

- ouvrir le fichier `~/.ssh/id_rsa.pub` avec `less` et copier la ligne (`CTRL+SHIFT+C`)
- se connecter sur le serveur distant par mot de passe

```
ssh <login>@<adresse>
```

- ajouter la clef (avec un éditeur texte) dans le fichier `~/.ssh/authorized_keys`