

Utilisation des systèmes UNIX

Benjamin Negrevergne

PSL University – Paris Dauphine – Équipes *MILES*



Systèmes d'exploitation (OS)

- Les programmes qui s'exécutent sur un ordinateur
 - Légitimes
 - Peu fiables
 - Utilisation de ressources trop importantes / infini
 - Mal-intentionnés
 - Piratage de données, ouvertures d'accès à des utilisateurs non autorisés
 - Avoir besoin de communiquer/collaborer
 - En compétition les uns avec les autres pour l'accès aux ressources

Systèmes d'exploitation (OS)

- Les programmes qui s'exécutent sur un ordinateur
 - Légitimes
 - Peu fiables
 - Utilisation de ressources trop importantes / infini
 - Mal-intentionnés
 - Piratage de données, ouvertures d'accès à des utilisateurs non autorisés
 - Avoir besoin de communiquer/collaborer
 - En compétition les uns avec les autres pour l'accès aux ressources
- L'OS s'assure du bon déroulement de l'exécution des programmes en contrôlant l'accès aux principales ressources :
 - CPU (i.e. temps de calcul), mémoire
 - Périphériques, canaux de communication
 - Données utilisateurs

Systèmes d'exploitation (OS)

- Les programmes qui s'exécutent sur un ordinateur
 - Légitimes
 - Peu fiables
 - Utilisation de ressources trop importantes / infini
 - Mal-intentionnés
 - Piratage de données, ouvertures d'accès à des utilisateurs non autorisés
 - Avoir besoin de communiquer/collaborer
 - En compétition les uns avec les autres pour l'accès aux ressources
- L'OS s'assure du bon déroulement de l'exécution des programmes en contrôlant l'accès aux principales ressources :
 - CPU (i.e. temps de calcul), mémoire
 - Périphériques, canaux de communication
 - Données utilisateurs

► Unix est le premier système d'exploitation

Objectifs pédagogiques de ce cours

- Familiariser avec l'utilisation de l'environnement Unix
- Donner une perspective de développeur plutôt que d'utilisateur/consommateur
- Préparer au cours de système d'exploitation

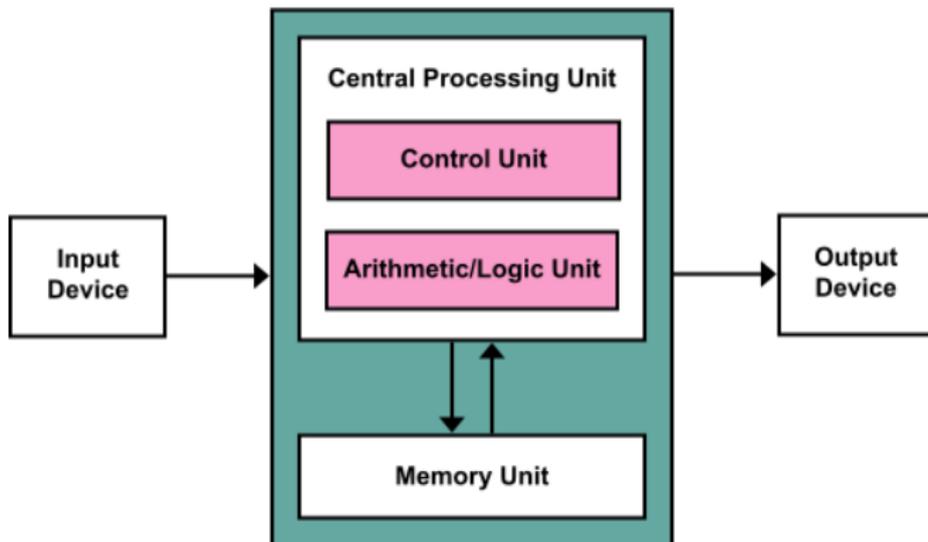
Objectifs pédagogiques de ce cours

- Familiariser avec l'utilisation de l'environnement Unix
- Donner une perspective de développeur plutôt que d'utilisateur/consommateur
- Préparer au cours de système d'exploitation

<https://www.lamsade.dauphine.fr/~bnegrevergne/ens/Unix/>

Architecture de Von Neumann (1945)

- Inventée par John von Neumann (1903-1957)



Composants principaux:

- CPU, capable d'effectuer des opérations (instructions assembleur)
- Mémoire RAM, capable de stocker des variables
- Entrées / Sorties, capables d'interagir avec l'utilisateur

Exécution d'un programme simple

Instructions:

- ADD: Additionner le contenu de deux registres
- LOAD: Charger une valeur de la RAM vers un registre
- STORE: Stocker une valeur dans un registre vers la RAM

Exécution d'un programme simple

Instructions:

- ADD: Additionner le contenu de deux registres
- LOAD: Charger une valeur de la RAM vers un registre
- STORE: Stocker une valeur dans un registre vers la RAM

Programme: séquence d'instructions + des données

```
1 // Programme en C qui additionne deux nombres et qui affiche de résultat
2 int a = 15;
3 int b = 17;
4 int c = a + b;
5 print("%d", c);
```

Exécution d'un programme simple

Instructions:

- ADD: Additionner le contenu de deux registres
- LOAD: Charger une valeur de la RAM vers un registre
- STORE: Stocker une valeur dans un registre vers la RAM

Programme: séquence d'instructions + des données

```
1 // Programme en C qui additionne deux nombres et qui affiche de résultat
2 int a = 15;
3 int b = 17;
4 int c = a + b;
5 print("%d", c);
```

```
1 // Sequence d instructions CPU correspondante (après compilation)
2 1: LOAD @6,R1
3 2: LOAD @7,R2
4 3: ADD R1,R2
5 4: STORE R1 @7
6 5: PRINT @7
7 6: 15
8 7: 17
```

CPUs modernes: $\sim 10^9$ instr./sec

Exécution d'un programme simple

Instructions:

- ADD: Additionner le contenu de deux registres
- LOAD: Charger une valeur de la RAM vers un registre
- STORE: Stocker une valeur dans un registre vers la RAM

Programme: séquence d'instructions + des données

```
1 // Programme en C qui additionne deux nombres et qui affiche de résultat
2 int a = 15;
3 int b = 17;
4 int c = a + b;
5 print("%d", c);
```

```
1 // Sequence d instructions CPU correspondante (après compilation)
2 1: LOAD @6,R1
3 2: LOAD @7,R2
4 3: ADD R1,R2
5 4: STORE R1 @7
6 5: PRINT @7
7 6: 15
8 7: 17
```

CPUs modernes: $\sim 10^9$ instr./sec

► Un CPU (1 cœur) n'exécute qu'un seul programme à la fois

Not so fast

- 1940 - 1950: Un seul utilisateur par ordinateur
- 1960: avènement carte perforées et du traitement par *batches*



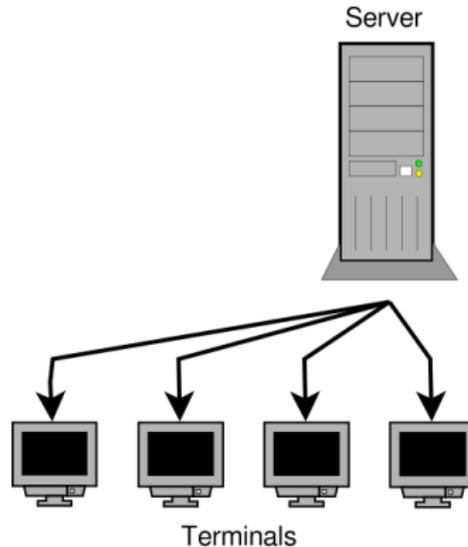
Images: wikiwand.com

$80 \times 10 \text{ (bits)} \times 2000 \text{ (cartes)} \times 165 \text{ (cartons)} \times 100 \text{ (palettes)} \approx 3\text{Go}$

Architecture serveur-terminal



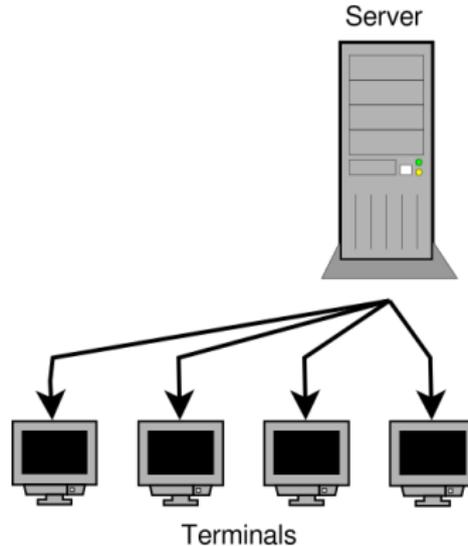
Terminal pour le PDP-7,
Image: @LivingComputers



Architecture serveur-terminal

```
danny@athlon: /home/danny
[danny@athlon ~]$ free
              total        used        free      shared  buffers   cached
Mem:    259532    23488    4852      0         464      6204
-/+ buffers/cache: 157412    9420      274304
Swap:   277632    2708    274924
[danny@athlon ~]$ ls /
bin/  dev/  home/  lib/  opt/  root/  sfs/  usr/
boot/  etc/  initrd/  net/  proc/  sbin/  tmp/  var/
[danny@athlon ~]$ uname -a
Linux athlon 2.6.11-6ck-#1 Tue Mar 22 16:04:32 CET 2005 i686 Athlon(tm) XP 2200+
unknown GNU/Linux
[danny@athlon ~]$
```

Virtual terminal xterm,
Image: Wikipedia



UNIX

► Nécessité d'un **système** capable de gérer plusieurs programmes et plusieurs utilisateurs

1965 – **MULTICS**: Un premier système d'exploitation avec

- Multi-utilisateurs
- Partage du temps CPU (*time sharing*) et des autres ressources

UNIX

► Nécessité d'un **système** capable de gérer plusieurs programmes et plusieurs utilisateurs

1965 – **MULTICS**: Un premier système d'exploitation avec

- Multi-utilisateurs
- Partage du temps CPU (*time sharing*) et des autres ressources

1969 – **UNICS** puis **UNIX**:

- Par *Ken Thompson* et *Dennis Ritchie*
- En assembleur pour PDP-7, puis PDP-11/45

UNIX

► Nécessité d'un **système** capable de gérer plusieurs programmes et plusieurs utilisateurs

1965 – **MULTICS**: Un premier système d'exploitation avec

- Multi-utilisateurs
- Partage du temps CPU (*time sharing*) et des autres ressources

1969 – **UNICS** puis **UNIX**:

- Par *Ken Thompson* et *Dennis Ritchie*
- En assembleur pour PDP-7, puis PDP-11/45

1973 – Réécriture de UNIX dans un langage de haut niveau

- Invention du langage C par *Dennis Ritchie*

Historique UNIX

Unix est une famille de système d'exploitations.

- Dérivés de Unics (premier système d'exploitation)
- Très nombreux forks

Quelques Unix (ou Unix-like)

- BSD (Berkeley Software Distribution) descendant de UNIX
- Free BSD version libre et gratuite de BSD
- GNU/Linux version libre et gratuite dérivée de BSD
- ...

Historique UNIX

Unix est une famille de système d'exploitations.

- Dérivés de Unics (premier système d'exploitation)
- Très nombreux forks

Quelques Unix (ou Unix-like)

- BSD (Berkeley Software Distribution) descendant de UNIX
- Free BSD version libre et gratuite de BSD
- GNU/Linux version libre et gratuite dérivée de BSD
- ...

Historique UNIX

Unix est une famille de système d'exploitations.

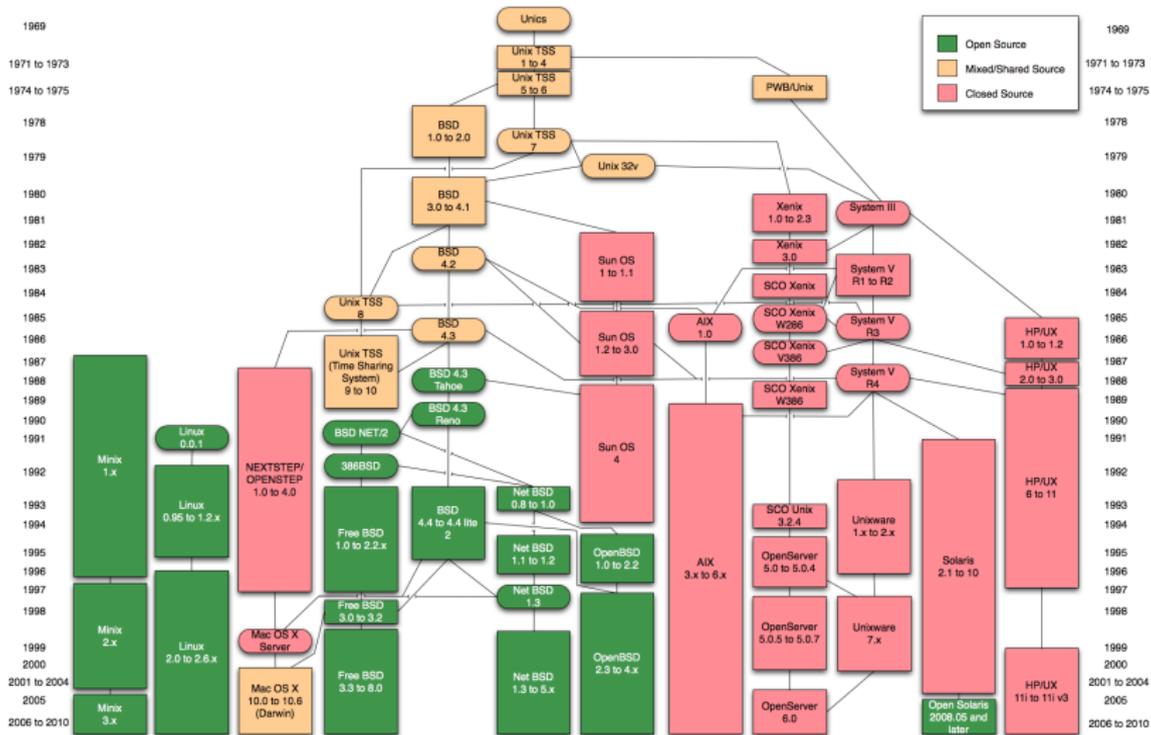
- Dérivés de Unics (premier système d'exploitation)
- Très nombreux forks

Quelques Unix (ou Unix-like)

- BSD (Berkeley Software Distribution) descendant de UNIX
- Free BSD version libre et gratuite de BSD
- GNU/Linux version libre et gratuite dérivée de BSD
- ...

► Importance de UNIX dans l'infrastructure du web!

D'autres UNIX-likes



Dérivé d'UNIX

- Android dérivé de Linux
- OSX (OS des Mac) dérivé de FreeBSD
- IOS dérivé de OSX
- Orbis OS (OS de la PS4), dérivé de Free BSD
- Freebox OS, dérivé de Linux
- MS-DOS/Windows (\approx 1980), n'est pas un dérivé d'Unix.
(Mais Microsoft Xenix en est un!)

Windows vs. Unix

- Windows: l'utilisateur n'est pas un programmeur
 - ▶ Des programmeurs professionnels développent des applications pour les utilisateurs.

 - Unix: utiliser un ordinateur c'est le programmer pour pouvoir automatiser des tâches
 - ▶ Tout utilisateur doit pouvoir facilement programmer lorsqu'il en a besoin.
- ▶ UNIX offre des avantages aux utilisateurs compétents.

Windows vs. Unix

- Windows: l'utilisateur n'est pas un programmeur
 - ▶ Des programmeurs professionnels développent des applications pour les utilisateurs.

 - Unix: utiliser un ordinateur c'est le programmer pour pouvoir automatiser des tâches
 - ▶ Tout utilisateur doit pouvoir facilement programmer lorsqu'il en a besoin.
- ▶ UNIX offre des avantages aux utilisateurs compétents.

Comment faciliter la programmation/l'automatisation de tâches répétitives?

Philosophie UNIX

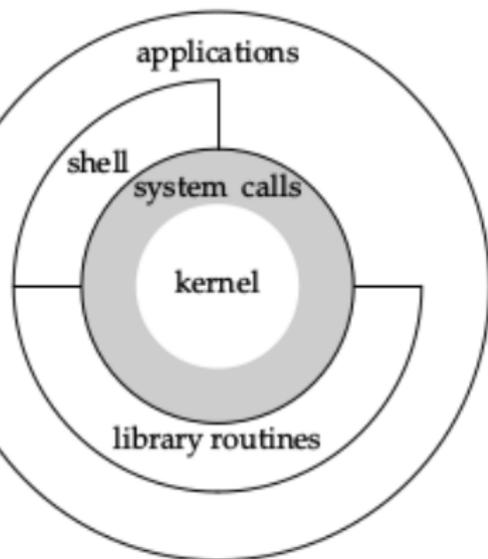
- 1 Écrire des programmes qui font qu'une chose mais qui le font bien
- 2 Écrire des programmes qui collaborent

Exemple avec le langage shell:

- La commande `ls` liste le contenu d'un répertoire (un fichier par ligne)
- La commande `wc -l` compte le nombre de ligne dans un fichier
- La commande `ls | wc -l` compte le nombre de fichiers dans un repertoire

```
1 $ ls
2 # combien de fichiers dans le repertoire courant
3 $ ls | wc -l
4 # top 5 des plus gros fichiers
5 $ ls -la | sort -k 5 -r | head -n 5
6 # ??
7 ls | grep -o '\.*' | sort | uniq -c | sort -r | head -n 5
```

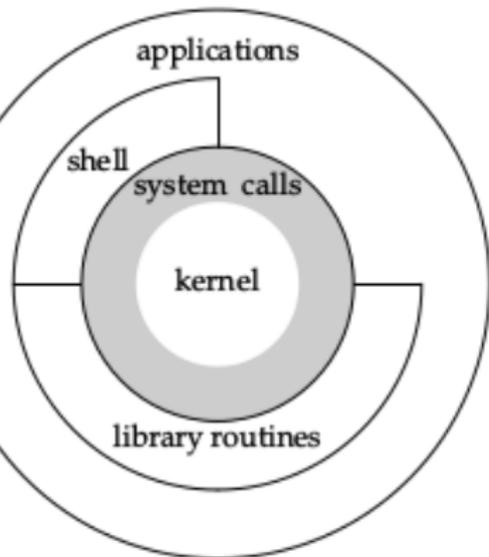
Architecture UNIX



Le noyau: contient les composants logiciels qui permettent de gérer l'accès au matériel

age: <https://github.com/congdv>

Architecture UNIX



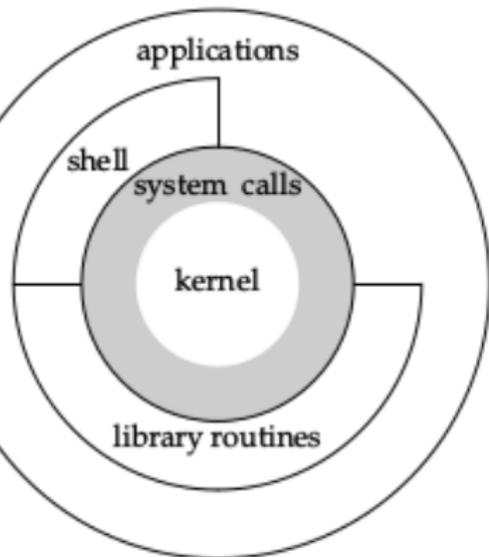
Le noyau: contient les composants logiciels qui permettent de gérer l'accès au matériel

l'API système: permet aux applications de communiquer le matériel via une API Elle permet d(e)

- unifier l'accès aux différents matériels
- contrôler l'exécution d'opérations critiques pour éviter les applications buggées/malveillantes de tout faire planter

Page: <https://github.com/congdv>

Architecture UNIX



Le noyau: contient les composants logiciels qui permettent de gérer l'accès au matériel

l'API système: permet aux applications de communiquer le matériel via une API Elle permet d(e)

- unifier l'accès aux différents matériels
- contrôler l'exécution d'opérations critiques pour éviter les applications buggées/malveillantes de tout faire planter

Le shell, et les applications: fournissent à l'utilisateur des fonctionnalités de haut niveau

age: <https://github.com/congdv>

Planning

10 x 1.5h de cours

- Utilisation/Programmation Shell
- Fonctionnement et utilisation de l'API Système
- Composants des systèmes d'exploitation
(processus, thread, scheduler etc.)

8 x 3h de TP

- Programme détaillé sur le site web.
- 4 séances de TP : API/Unix (Projet Écrire un shell)
- 1 séances de TP : Problèmes noyau (Projet Allocateur Mémoire)

Chargés de TP:

- Florian Sikora
- Moi.

Utilisateurs Unix

- Les utilisateurs sous Unix détiennent des ressources (fichiers, processus, etc.)
- Les utilisateurs sont identifié par un nom d'utilisateur (login)
- Différents utilisateurs peuvent avoir différents privilèges
- L'utilisateur root à tous les privilèges

Lorsqu'un programme s'exécute, **il hérite de tous les privilèges de l'utilisateur qui l'a lancé**, et peut accéder à toutes les ressources détenues par l'utilisateur.

⚠ Ne pas exécuter des programmes d'origine incertaine

⚠ Ne pas exécuter de programme en tant que root si ça n'est pas strictement nécessaire

Connexion à un système Unix

Le système a recours à un mécanisme d'authentification (e.g. mot de passe)

Connexion à un système Unix

Le système a recours à un mécanisme d'authentification (e.g. mot de passe)

Basé sur une fonction h non-inversible

- $e = h(d)$ → facile

Ex: $md5("mdp") = 41dde949f31b2c175cff20718de936b7$

- $d = h^{-1}(e)$ → difficile

Ex: $md5^{-1}(ea6c3834c6fa57258d4a8a174a5e4ab2) = ?$

Connexion à un système Unix

Le système a recours à un mécanisme d'authentification (e.g. mot de passe)

Basé sur une fonction h non-inversible

- $e = h(d)$ → facile

Ex: $md5("mdp") = 41dde949f31b2c175cff20718de936b7$

- $d = h^{-1}(e)$ → difficile

Ex: $md5^{-1}(ea6c3834c6fa57258d4a8a174a5e4ab2) = ?$

Authentification avec un mot de passe p :

- Le mdp crypté $e = h(p)$ est stocké sur le serveur
- L'utilisateur envoie un mdp p' au serveur
- Le serveur calcule $h(p')$
- Le serveur accepte la connexion si $h(p') = e$

Note: le serveur ne connaît pas le mdp non crypté p !

Connexion à un système Unix

Le système a recours à un mécanisme d'authentification (e.g. mot de passe)

Basé sur une fonction h non-inversible

- $e = h(d)$ → facile

Ex: $md5("mdp") = 41dde949f31b2c175cff20718de936b7$

- $d = h^{-1}(e)$ → difficile

Ex: $md5^{-1}(ea6c3834c6fa57258d4a8a174a5e4ab2) = ?$

Authentification avec un mot de passe p :

- Le mdp crypté $e = h(p)$ est stocké sur le serveur
- L'utilisateur envoie un mdp p' au serveur
- Le serveur calcule $h(p')$
- Le serveur accepte la connexion si $h(p') = e$

Note: le serveur ne connaît pas le mdp non crypté p !

⚠ Si le mot de passe est court/simple, il est possible de trouver $h^{-1}(e)$ comparant c avec $h(x)$ pour des x possibles

Ex. attaques à base de dictionnaire

Invite de commandes (Shell)

Prompt: (ne pas recopier)

```
1 $
```

Format des commandes:

```
1 $ nom_commande [options] [arguments]
```

Commande simple

```
1 $ ls  
2 steibergs.png test.c test.dat test.dat
```

Commande avec option (-l : long listing format)

```
1 $ ls -l  
2 -rw-r--r-- 1 bnegreve bnegreve 14674 Jul 2 2019 steibergs.png  
3 -rw-r--r-- 1 bnegreve bnegreve 78 Nov 16 23:48 test.c  
4 -rw-r--r-- 1 bnegreve bnegreve 16 Dec 5 19:35 test.dat  
5 -rw-r--r-- 1 bnegreve bnegreve 16 Dec 5 19:35 test.dat
```

Commande avec argument et option

```
1 $ wc -l test.txt  
2 438
```

Autres exemples

```
1 $ who # qui est connecté sur la machine
2 or pts/11 2020-01-08 14:25 (10.1.3.176)
3 tamby pts/12 2020-01-09 12:52 (10.1.3.208)
4 tamby pts/13 2020-01-09 13:26 (10.1.3.208)
5 tamby pts/20 2020-01-09 14:07 (10.1.3.208)
6 bnegrevergne pts/21 2020-01-27 15:03 (10.1.3.208)
7 $ cat test.txt # affiche le contenu du fichier test.txt dans le terminal
8 salut
9 $ pwd # affiche le nom de repertoire courant
10 /home/bnegrevergne
11 $ mkdir unix # crée un nouveau repertoire dans le repertoire courant
12 $ cd unix # se déplace dans le répertoire unix
13 $ mv a.txt b.txt # renomme ou déplace un fichier
14 $ gzip b.txt # compresse un fichier
15 $ date # affiche la date
16 $ cal # affiche un calendrier
17 $ sort # trie les lignes dans l'ordre alphabétique (ou autre)
```

Obtenir de l'aide

```
1 $ man ls
```

```
1 LS(1)
```

```
    User Commands
```

```
    LS(1)
```

```
2
```

```
3 NAME
```

```
4
```

```
    ls - list directory contents
```

```
5
```

```
6 SYNOPSIS
```

```
7
```

```
    ls [OPTION]... [FILE]...
```

```
8
```

```
9 DESCRIPTION
```

```
10
```

```
    List information about the FILES (the current directory by default). Sort  
    entries alphabetically if none of -cftuvSUX nor --sort is specified.
```

```
11
```

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

```
12
```

```
    -a, --all
```

```
14
```

```
        do not ignore entries starting with .
```

```
15
```

```
    -A, --almost-all
```

```
16
```

```
        do not list implied . and ..
```

```
17
```

```
18
```

man man

```
1      1 Executable programs or shell commands
2      2 System calls (functions provided by the kernel)
3      3 Library calls (functions within program libraries)
4      4 Special files (usually found in /dev)
5      5 File formats and conventions, e.g. /etc/passwd
6      6 Games
7      7 Miscellaneous (including macro packages and conventions), e.g. man(7),
      groff(7)
8      8 System administration commands (usually only for root)
9      9 Kernel routines [Non standard]
```

```
1 $ man 1 write
2     write - send a message to another user
3     ...
4 $ man 2 write
5     write - write to a file descriptor
6     ...
```