Introduction Système

Une introduction au système d'exploitation Linux

Guillaume Santini

guillaume.santini@iutv.univ-paris13.fr
IUT de Villetaneuse

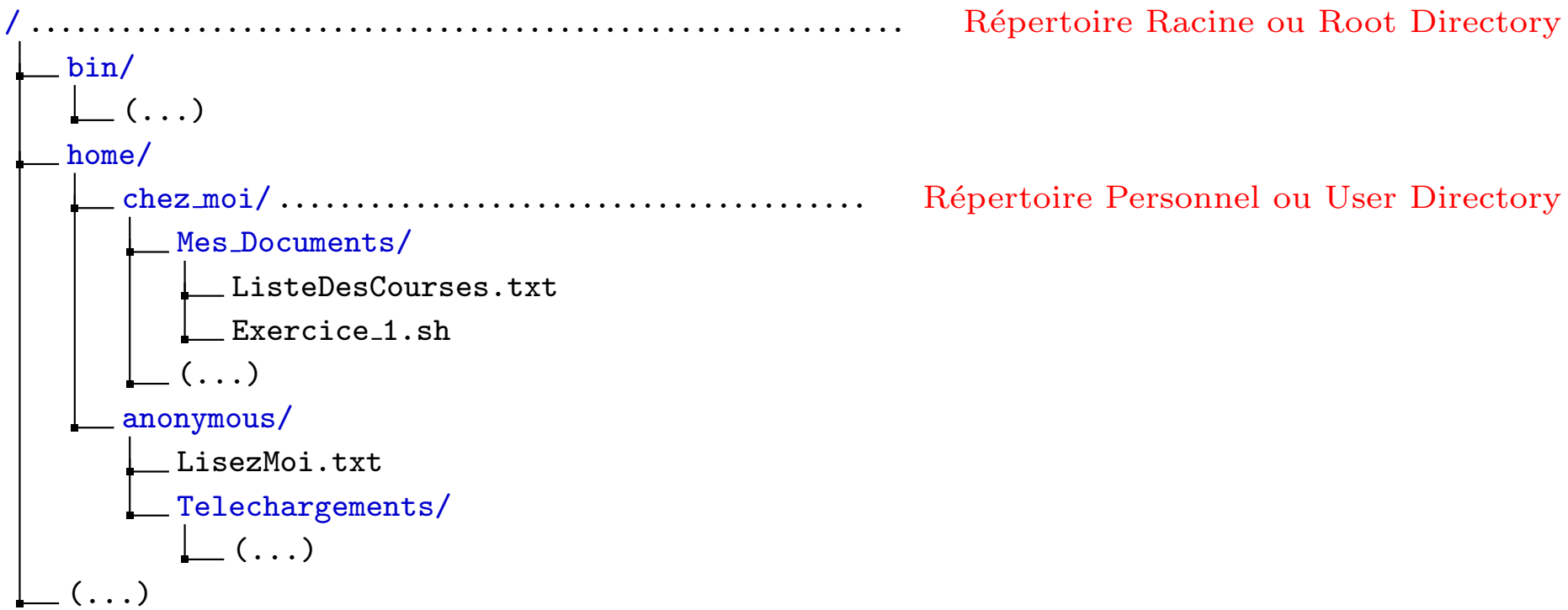
2 janvier 2012

Partie #2

Plan

L'organisation arborescente

Exemple d'arborescence Linux



Les répertoires importants

- Le Repertoire Racine (*Root directory*) contient tous les répertoires et fichiers accessibles depuis le système.
- Le Répertoire Personnel (*User Directory* ou *Home Directory*) est le répertoire dans lequel l'utilisateur peut faire ce qu'il veut (écrire, modifier, supprimer, installer ...).

La notion de Chemin

Le chemin défini un nom unique

- Deux fichiers ou répertoires ne peuvent pas porter le même nom si ils sont dans un même répertoire.
- Les noms des fichiers et répertoires différencient les caractères Majuscules et minuscule. Les fichiers Essai.txt et essai.txt peuvent donc être dans le même répertoire.

Exemples de chemins absolus

```
/ ..................................................  Un chemin absolu part de la racine /
└── home/ ..........................................  /home/
    └── chez_moi/ ..................................  /home/chez_moi/
        ├── Etoiles/ ...............................  /home/chez_moi/Etoiles/
        │   ├── SOLEIL.jpg .........................  /home/chez_moi/Etoiles/SOLEIL.jpg
        │   └── Soleil.jpg .........................  /home/chez_moi/Etoiles/Soleil.jpg
        └── Systeme_Solaire/ .......................  /home/chez_moi/Systeme_Solaire/
            └── SOLEIL.jpg .........................  /home/chez_moi/Systeme_Solaire/SOLEIL.jpg
```

Syntaxe d'un chemin absolu

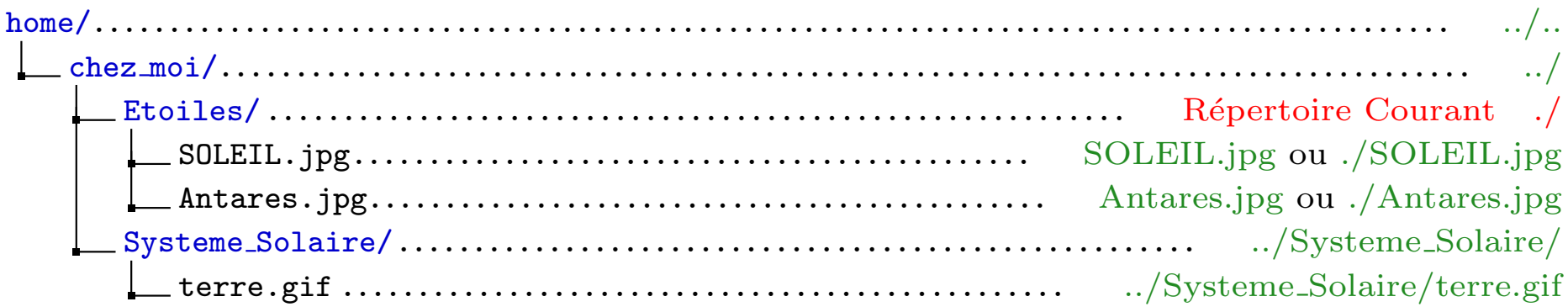
Le chemin *absolu* d'un fichier ou d'un répertoire est unique. Il donne la liste des répertoires et sous-répertoires en partant de la racine / (la référence *absolue* de l'arborescence) jusqu'à la cible.

Répertoire courant et Chemins relatifs

Le répertoire courant

- Le répertoire courant est un répertoire de référence d'où sont lancées les commandes.
- Par défaut, le répertoire courant est le répertoire personnel de l'utilisateur,
- Naviguer dans l'arborescence équivaut à modifier le répertoire courant.

Exemples de chemins relatifs



Syntaxe d'un Chemin Relatif

- Le chemin *relatif* d'un fichier ou d'un répertoire donne la liste des répertoires et sous-répertoires en partant du répertoire courant (la référence *relative* dans l'arborescence) jusqu'à la cible.
- Il est relatif, car lorsque le répertoire courant change, le chemin relatif change.

Notation Spéciales

Les chemins des répertoires de référence

Répertoire	Notation
Répertoire Racine	/
Répertoire Personnel	~

Répertoire	Notation
Répertoire Courant	.
Répertoire Parent	..

Remarques

- La notation ~ correspond à un chemin absolu. Elle est remplacée lors d'une évaluation par le chemin absolu du répertoire personnel de l'utilisateur.

Exemple de chemins valides pointant le fichier cible

```
/ ..........................  Répertoire Racine
└── home/
    └── chez_moi/ .....  Répertoire Personnel
        └── Etoiles/ ....  Répertoire Courant
            └── Soleil.jpg ......  Fichier cible
```

Chemins Absolus
/home/chez_moi/Etoiles/Soleil.jpg
~/Etoiles/Soleil.jpg
/home/chez_moi/../chez_moi/Etoile/Soleil.jpg
/home/chez_moi/../../home/chez_moi/Etoile/Soleil.jpg
Chemins Relatifs
Soleil.jpg
../Etoile/Soleil.jpg
../../chez_moi/Etoile/Soleil.jpg

Tout est Fichier

Gestion des fichiers

Lors de la création du système de fichier une table des i-nodes est créée. Celle-ci fixe le nombre maximum de fichiers.

Fichiers

- Chaque fichier est décrit comme un i-node.
- L'i-node contient un certain nombre de *métadonnées* concernant le fichier :
  - adresse sur le disque et taille du fichier en nombre d'octets,
  - identification du propriétaire (UID et GID) et permissions (lecture, écriture et exécussion),
  - dates de dernière modification et de dernier accès,
  - ...
- Le nom du fichier n'est pas stocké dans son i-node !

Répertoire

Un répertoire est un fichier spécial listant les références des fichiers qu'il contient sous forme de couples (nom_du_fichier, i-node).

Fichiers spéciaux

Les fichiers de périphériques sont des fichiers spéciaux mis en place par le système pour assurer le lien avec un périphérique.

Conventions

Noms et Chemins

- Par convention, le nom d'un fichier ou d'un répertoire est identifié avec son chemin (sauf mention contraire explicite).
- Par convention, un chemin peut être absolu, relatif. Il peut utiliser les notations spéciales.
- Par convention la notion de fichier sera comprise dans son sens large. Par exemple, le chemin d'un fichier devra être interprété sans distinction comme le chemin vers un fichier ordinaire ou comme le chemin vers un répertoire (sauf mention contraire explicite).

Commandes, options, paramètres

Commande c'est le nom d'un programme qui exécute une action.

Options ce sont des paramètres optionnels. Ils peuvent être omis. L'ajout d'options modifie le comportement de la commande (le résultat). Les options sont encadrées par les caractères `< options >`.

Paramètres ce sont des arguments que la commande évalue.

Sources et Cible

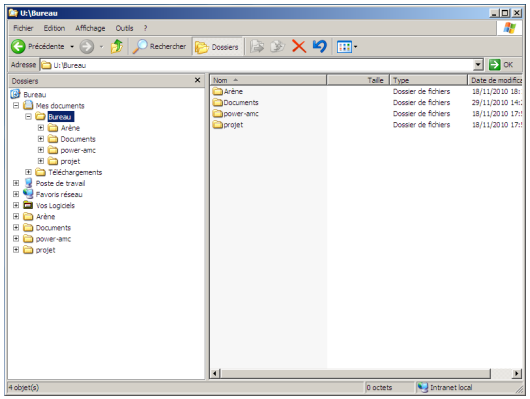
Source c'est un fichier ou un répertoire utilisé en entrée d'une commande,

Cible c'est un fichier ou un répertoire utilisé en sortie d'une commande.

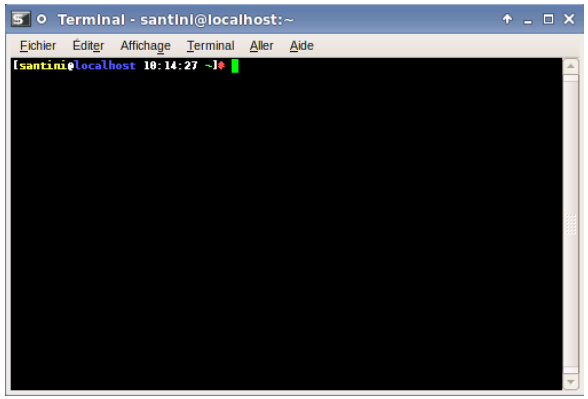
Manipulation de l'arborescence en ligne de commande

Alternatives pour naviguer dans l'arborescence et manipuler les fichiers

Interface Graphique



Ligne de Commande



Principales commandes

Commande	Fonction principale
`pwd`	Afficher le nom du répertoire courant
`ls`	Afficher le contenu d'un répertoire
`cd`	Changer de répertoire courant
`mkdir`	Créer un répertoire
`rm`	Supprimer fichier(s) ou répertoire(s)
`cp`	Copier fichier(s) ou répertoire(s)
`mv`	Déplacer/Renommer fichier(s) ou répertoire(s)

pwd

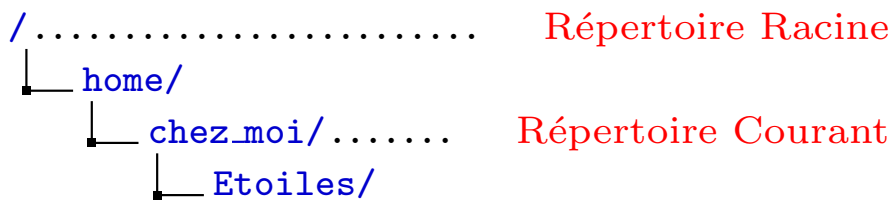
Syntaxe

```
pwd
```

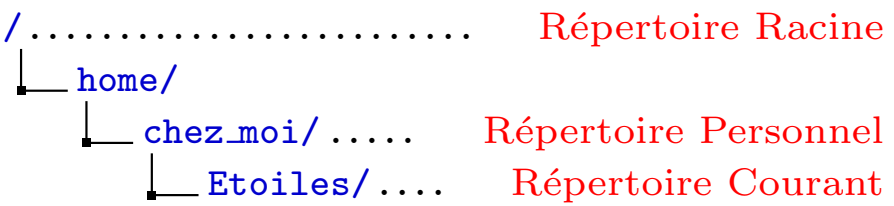
Description

- Affiche le nom du répertoire courant.

Exemple d'utilisation:




```
[ login@localhost ~ ] pwd
/home/chez_moi
```




```
[ login@localhost ~/Etoiles ] pwd
/home/chez_moi/Etoiles/
```

ls

Syntaxe

```
ls  <source>
```

Description

- Affiche le contenu d'un répertoire.
- Par défaut si aucune source n'est indiquée, la commande affiche le contenu du répertoire courant.

Exemple d'utilisation:

```
/ ..... Répertoire Racine
└── home/ ..... Répertoire Courant
    └── chez_moi/
        ├── astronomie.txt
        └── Etoiles/
```

```
[ login@localhost /home/ ] ls
chez_moi/
```

```
[ login@localhost /home/ ] ls chez_moi/
Etoiles/ astronomie.txt
```

ls(bis)

Syntaxe

```
ls -a <source>
```

Description

- Affiche le contenu d'un répertoire y compris les fichiers et répertoires cachés.
- Les fichiers et répertoires cachés ont un nom dont le premier caractère est un point.
- Les fichiers et répertoires cachés sont utilisés par le système ou certaines applications.

Exemple d'utilisation:

```
chez_moi/....    Rép. Courant
├── ./ssh/
│   ├── id_rsa
│   ├── id_rsa.pub
│   └── known_hosts
├── .bashrc
├── astronomie.txt
└── Etoiles/
    └── soleil.jpg
```

Sans option -a

```
[ login@localhost ~ ] ls
astronomie.txt
Etoiles/

[ login@localhost ~ ] ■
```

Avec option -a

```
[ login@localhost ~ ] ls -a
.
..
.ssh/
.bashrc
astronomie.txt
Etoiles/

[ login@localhost ~ ] ■
```

cd

Syntaxe

```
cd  <cible>
```

Description

- Change le répertoire courant (permet de naviguer dans l'arborescence).
- Si le chemin du répertoire cible est omit, le répertoire courant redevient par défaut le répertoire personnel.

Exemple d'utilisation:

```
/ ..................................................... Répertoire Racine
└── home/ ............................................. Répertoire Courant Initial
    └── chez_moi/ ..................................... Répertoire Courant Final #1
        ├── astronomie.txt
        └── Etoiles/ .................................. Répertoire Courant Final #2
```

Commande #1 :

```
[ login@localhost /home ] cd

[ login@localhost ~ ] ■
```

Commande #2 :

```
[ login@localhost /home ] cd chez_moi/Etoile

[ login@localhost ~/Etoile ] ■
```

mkdir

Syntaxe

```
mkdir  chemin <chemin_2 ...>
```

Description

- Création d'un ou de plusieurs répertoires aux endroits spécifiés par les chemins.
- Si le chemin est occupé par un fichier ou un répertoire, il y a un message d'erreur.

Exemple d'utilisation:

```
chez_moi/ ..... Répertoire Courant
├── astronomie.txt
├── Systeme_Solaire/ ..... Création Commande #1
├── Etoiles/
│   ├── Rouges/ ..... Création Commande #2
│   └── Bleues/ ..... Création Commande #3
└── Galaxies/ ..... Création Commande #3
```

Commande #1 :

```
[ login@localhost ~ ] mkdir Systeme_Solaire
```

Commande #2 :

```
[ login@localhost ~ ] mkdir Etoiles/Rouges
```

Commande #3 :

```
[ login@localhost ~ ] mkdir Galaxies Etoiles/Bleues
```

rm

Syntaxe

```
rm  chemin <chemin_2 ...>
```

Description

- La commande supprime le fichier pointé par le(s) chemin(s).
- Si le chemin pointe sur un répertoire, la commande affiche un message d'erreur.

Exemple d'utilisation:

```
chez_moi/ ..... Répertoire Courant
├── astronomie.txt ..... Supprimé par la Commande #1
├── Etoiles/
│   └── soleil.jpg ..... Supprimé par la Commande #2
└── aldebaran.gif ..... Supprimé par la Commande #2
```

Commande #1 :

```
[ login@localhost ~ ] rm astronomie.txt
```

Commande #2 :

```
[ login@localhost ~ ] rm aldebaran.gif Etoiles/soleil.jpg
```

rm(bis)

Syntaxe

```
rm -r chemin <chemin_2 ...>
```

Description

- L'option -r (**R**écursif) permet de supprimer un répertoire et tout son contenu.

Exemple d'utilisation:

```
chez_moi/ ..................................................... Répertoire Courant
├── astronomie.txt
├── Etoiles/ ..................................... Supprimé par la Commande #1
│   ├── soleil.jpg ............................... Supprimé par la Commande #1
│   └── Galaxie/ ................................. Supprimé par la Commande #1
│       └── Andromede.pdf ........................ Supprimé par la Commande #1
└── aldebaran.gif
```

Commande #1 :

```
[ login@localhost ~ ] rm -r Etoiles
```

cp

Syntaxe

```
cp  source cible
```

Description

- Copie le fichier source vers la cible.
- La source doit être un fichier ordinaire (pas un répertoire),
- Si la source est un répertoire la commande produit un message d'erreur.
- Si la cible :
  - est le chemin d'un répertoire existant, le fichier sera copié dans ce répertoire et conservera son nom,
  - ne correspond pas à un répertoire existant, le fichier sera copié avec le nom `cible`.

Exemple d'utilisation:

```
chez_moi/ ..... Répertoire Courant
├── astronomie.txt ..... Fichier Source Commande #1
├── Etoiles/ ..... Répertoire Cible Commande #1
│   └── astronomie.txt ..... Copié/Créé par la Commande #1
└── cv.pdf
```

Commande #1 :

```
[ login@localhost ~ ] cp astronomie.txt Etoiles
```

cp(bis)

Syntaxe

```
cp  source <source_2 ...> cible
```

Description

- Copie plusieurs fichiers sources vers la cible.
- Les sources doivent être des fichiers ordinaires, et la cible un répertoire.

Exemple d'utilisation:

```
chez_moi/ ..................................................... Répertoire Courant
├── cv.pdf ..................................................... Fichier Source Commande #2
├── motivations.pdf ............................................ Fichier Source Commande #2
└── Candidature/ ............................................... Répertoire Cible Commande #2
    ├── cv.pdf ................................................. Copié/Créé par la Commande #2
    └── motivations.pdf ........................................ Copié/Créé par la Commande #2
```

Commande #2 :

```
[ login@localhost ~ ] cp cv.pdf motivations.pdf Candidature
```

cp(ter)

Syntaxe

```
cp -r source <source_2 ...> cible
```

Description

- L'option -r (**R**écursif) permet de copier un répertoire et son contenu si il apparait dans le(s) source(s).

Exemple d'utilisation:

```
chez_moi/ ..................................................... Répertoire Courant
├── astronomie.txt
├── Galaxie/ ..................................... Fichier Source Commande #3
│   └── Andromede.pdf
├── Etoiles/ ..................................... Répertoire Cible #3
│   ├── soleil.jpg
│   └── Galaxie/ ..................... Copié/Créé par la Commande #3
│       └── Andromede.pdf ..... Copié/Créé par la Commande #3
└── aldebaran.gif
```

Commande #3 :

```
[ login@localhost ~ ] cp -r Galaxies Etoiles
```

mv

Syntaxe

```
mv  source cible
```

Description

Déplace/Renomme un fichier ou répertoire.

- modifie le chemin d'accès à la `source` qui devient le chemin `cible`.
- Le chemin `source` disparait et le chemin `cible` est créé.
- Le fichier ou répertoire pointé reste le même.
- La cible doit être un chemin non occupé ou un répertoire.

Exemple d'utilisation: Renommer un fichier

État Initial de l'arborescence :

```
chez_moi/.............  Répertoire Courant
└── AstroNomIe.TXT.....  Fichier Source
```

État Final de l'arborescence :

```
chez_moi/.............  Répertoire Courant
└── astronomie.txt .....  Fichier Renommé
```

```
[ login@localhost ~ ] mv AstroNomIe.TXT astronomie.txt
```

mv(bis)

Exemple d'utilisation: Déplacer un Répertoire

État Initial de l'arborescence :

```
chez_moi/ ..... Répertoire Courant
├── astronomie.txt ..... Fichier Source
└── Etoiles/ ..... Répertoire Cible
```

État Final de l'arborescence :

```
chez_moi/ ..... Répertoire Courant
└── Etoiles/ ..... Répertoire Cible
    └── astronomie.txt ..... Fichier Déplacé
```

```
[ login@localhost ~ ] mv astronomie.txt Etoiles
```

Exemple d'utilisation: Renommer un Répertoire

État Initial de l'arborescence :

```
chez_moi/ ..... Répertoire Courant
└── Etoiles/ ..... Répertoire Source
    └── astronomie.txt
```

État Final de l'arborescence :

```
chez_moi/ ..... Répertoire Courant
└── Relativite/ ..... Répertoire Renommé
    └── astronomie.txt
```

```
[ login@localhost ~ ] mv Etoiles Relativite
```

mv(ter)

Exemple d'utilisation:

État Initial de l'arborescence :

```
chez_moi/.............  Répertoire Courant
├── astronomie.txt.....  Fichier Source
├── relativite.pdf.....  Fichier Source
└── Etoiles/...........  Répertoire Cible
```

État Final de l'arborescence :

```
chez_moi/.............  Répertoire Courant
└── Etoiles/...........  Répertoire Cible
    ├── astronomie.txt.  Fichier Déplacé
    └── relativite.pdf.  Fichier Déplacé
```

```
[ login@localhost ~ ] mv astronomie.txt relativité.pdf Etoiles
```

Exemple d'utilisation:

État Initial de l'arborescence :

```
chez_moi/.............  Répertoire Courant
├── relativite.pdf.....  Fichier Source
├── Etoiles/...........  Répertoire Source
│   └── astronomie.txt
└── Espace/............  Répertoire Cible
```

État Final de l'arborescence :

```
chez_moi/.............  Répertoire Courant
└── Espace/............  Répertoire Cible
    ├── relativite.pdf.  Fichier Déplacé
    └── Etoiles/.......  Répertoire Déplacé
        └── astronomie.txt  Fichier Déplacé
```

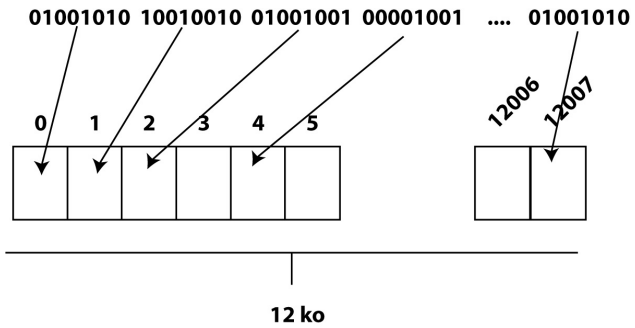
```
[ login@localhost ~ ] mv relativité.pdf Etoiles Espace
```

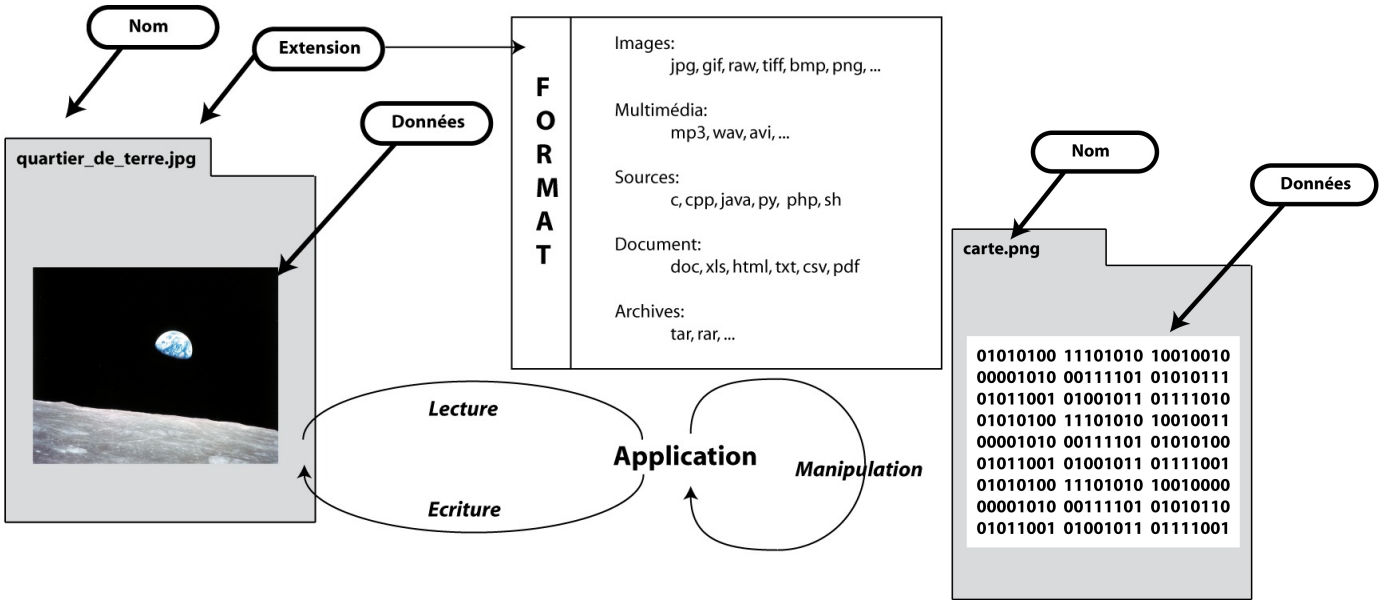
Plan

Fichier binaire et fichier texte

Les données numériques

Tout fichier enregistré sur un support numérique est encodé sous forme binaire.





Accès aux données

Lors de son utilisation un fichier est *lu* par un programme. Pour cela il doit décoder les informations binaires et les traiter.

Fichier binaire et fichier texte

Deux grands types de fichiers : Binaire Vs Numérique

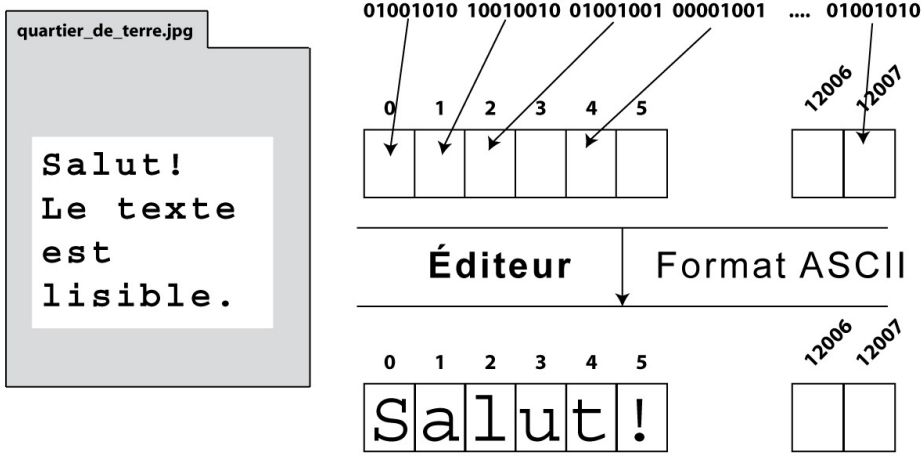
De façon générale un fichier binaire ne peut être *"lu"* que par un programme informatique, alors qu'un fichier texte peut être *"lu"* par être humain.

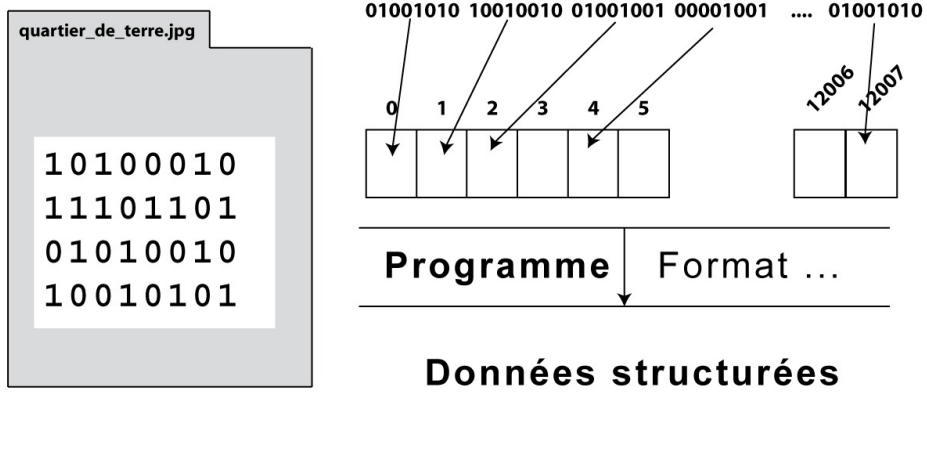
Les fichiers textes

C'est un fichier qui peut être *"lu"* par un éditeur de texte brut. Les données sont encodées au format ASCII (une norme). Chaque octet correspond à un caractère (256 possibles).

Les fichiers binaires

Ce n'est pas un fichier texte . . .Il peut contenir des instructions machines, des données compressées, des données binaires brutes nécessitant un programme pour être lues.





Fichiers sources → Exécutable → Processus

Les sources : Une *"recette de cuisine"*

- Exprime un ensemble de tâches à réaliser pour accomplir le programme (le plat cuisiné).
- Utilise un langage de programmation.
- C'est un fichier texte.

L'exécutable

- Exprime les mêmes tâches dans un langage machine.
- Ce fichier ne fonctionne que sur des ordinateurs qui ont la même architecture.
- C'est un fichier binaire.

Les processus

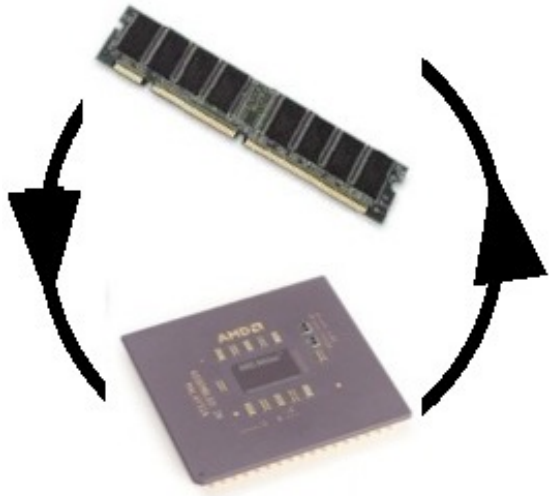
- L'évaluation des instructions machines engendre des processus.
- Ces processus sont exécutés par le matériel.
- Les instructions machine doivent donc être adaptées au matériel.

dessine.c

```
(...)
float r, x, y;
r=3.0;
x=0.0;
y=7.1;
cercle([0.,0.],r)
segment([0.,0.],[x,y])
```

dessine

```
10100101 11101001
10001001 00100101
00101010 00100010
01111011 10110101
01000010 00110011
00101101 11010100
(...)
```



Identification des processus par le système d'exploitation

Système multi-utilisateur

- Plusieurs utilisateurs partagent les mêmes ressources matériel (RAM, CPU, disques, ...),
- Chaque utilisateur lance des processus liés à ses activités sur la machine et il utilise les résultats de ces processus.

Système multi-tâches

- Plusieurs programmes en cours d'exécution partagent les mêmes ressources matériel (mémoire vive, CPU, disques, ...). Ils peuvent provenir d'un seul ou de plusieurs utilisateurs,
- Chaque programmes lance des processus et il utilise les résultats de ces processus.

Il faut partager les ressources ! ! !

- Chaque programme doit être exécuté éventuellement *"en même temps"*. Il faut donc gérer le partage des ressources de calcul (accès à la mémoire vive, au CPU),
- Chaque programme ou utilisateur doit pouvoir retrouver les résultats de ses calculs. Il faut donc pouvoir identifier qui a lancé les processus et qui doit récupérer les résultats.

La gestion des processus est réalisée par le système d'exploitation. C'est une de ses tâches principales. Pour cela il a besoin de pouvoir identifier chaque processus.

PID et PPID

PID - Process IDentifier

- C'est un numéro unique attribué à chaque processus lors de son lancement.
- Il permet d'identifier de façon unique chaque processus.
- La liste des processus en cours d'exécution est accessible en ligne de commande par les commandes `ps` et `top`.

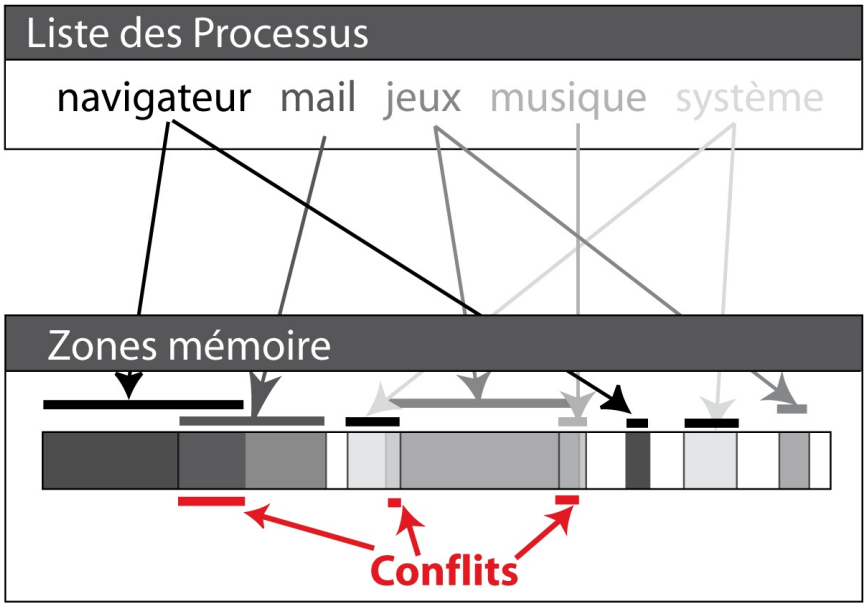
PPID - Parent Process IDentifier

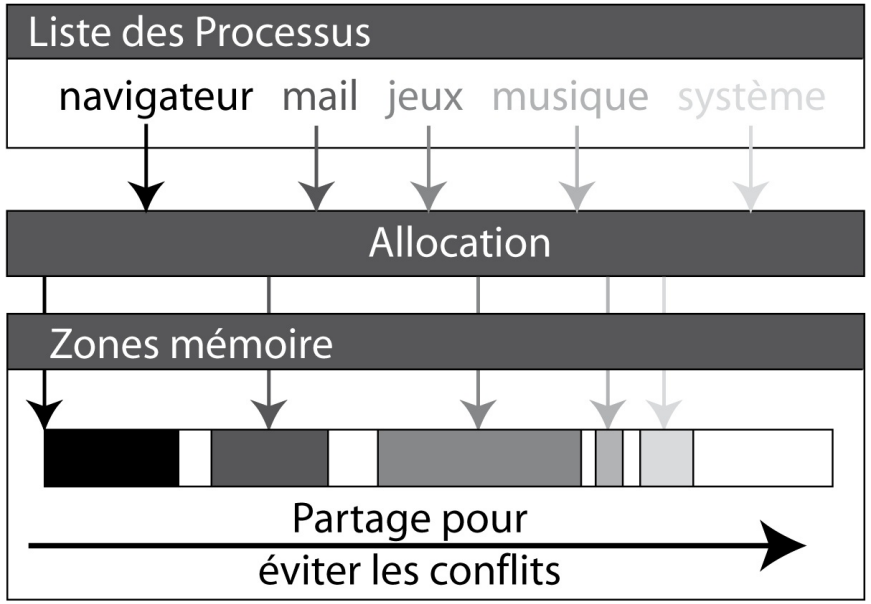
- Le premier processus lancé porte le numéro de PID 1. Les processus suivants sont des processus issus de ce processus parent.
- Chaque processus est lancé par un processus parent *via* l'appel système `fork`.
- Le PPID est le PID du processus Parent.

Utilités

- L'utilisateur peut suivre un processus, le suspendre temporairement, le relancer ou le tuer (interruption définitive).
- Le système s'en sert pour lui affecter des ressources matériel.

Gestion de la mémoire vive





Chaque processus a besoin de mémoire

Pour stocker et travailler sur :

- les données,
- les instructions,
- les résultats.

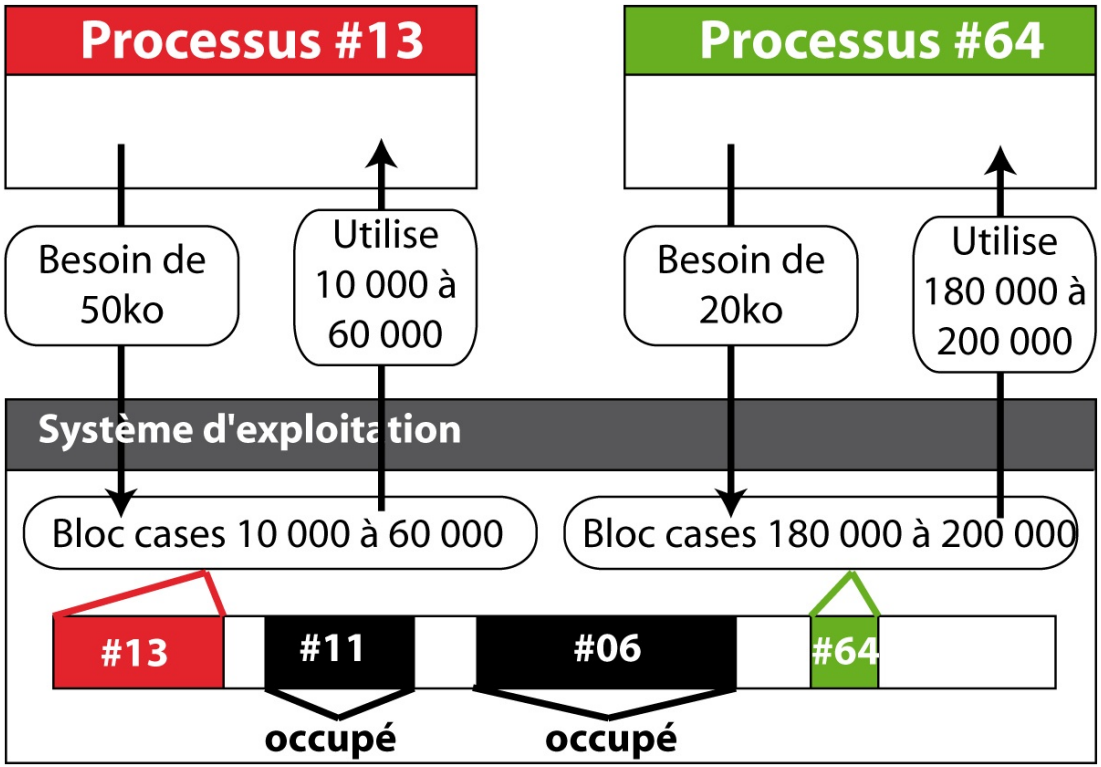
Il faut assurer l'intégrité des données !

Allocation de zone mémoire

L'allocation permet :

- d'attribuer à chaque processus un espace de travail en mémoire,
- le système contraint le programme à écrire dans sa zone mémoire et ainsi,
- évite qu'un programme modifie les données d'un autre programme.

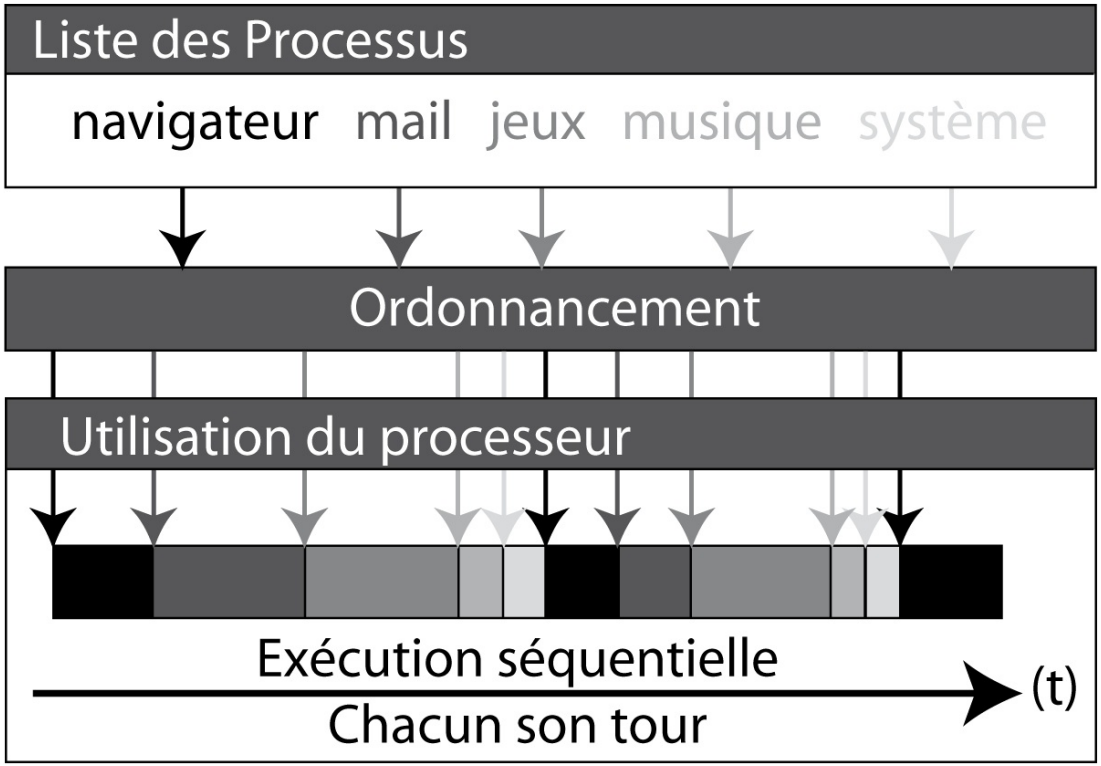
Gestion de la mémoire vive



Principes généraux de l'allocation

- L'OS maintient une table des zones mémoires allouées à chaque processus. Ces zones sont réservées et ne peuvent être utilisées que par le processus parent.
- Lorsqu'il a besoin de mémoire, un processus demande à l'OS quelle zone il peut utiliser,
- L'OS lui attribue, en fonction de l'espace libre, un certain nombre de blocs mémoire.
- Les blocs mémoire attribués sont alors réservés.

Gestion de l'accès au CPU



Le planificateur gère le temps CPU attribué à chaque processus

- Le CPU ne traite qu'un seul processus à la fois,
- Le planificateur permet l'alternance d'accès au CPU en attribuant une priorité à chaque processus.
- L'illusion d'exécution simultanée de plusieurs processus est donnée par une alternance rapide d'attribution de temps de calcul à chaque processus.

ps

Syntaxe

ps <-*eu*>

Description

- Affiche les processus en cours d'exécution.
- L'option <-e> indique que tous les processus doivent être affichés,
- L'option <-u> restreint l'affichage aux processus de l'utilisateur.

Exemple d'utilisation:

```
[ login@localhost ~ ] ps -eu
  Warning : bad ps syntax, perhaps a bogus '-'? See http ://procps.sf.net
   USER  PID %CPU %MEM  VSZ  RSS   TTY STAT  START  TIME COMMAND
santini 5905  0.0  0.2 4824 1656 pts/1   Ss 09 :27 0 :00 -bash LC_ALL=fr_FR.UTF
santini 5962  0.0  0.1 3884  896 pts/1   R+ 09 :48 0 :00 ps -eu MANPATH=/etc/jav

[ login@localhost ~ ] ■
```

top

Syntaxe

```
top
```

Description

- Permet de suivre dynamiquement (temps réel) les ressources matériel utilisées par chaque processus.
- Ouvre un interface dans la ligne de commande qui peut être quittée en pressant la touche Q
- Donne pour chaque processus en autres choses, le PID, le nom du propriétaire, la date de lancement du processus, les %CPU et %MEM utilisés.

Exemple d'utilisation:

```
  Tasks : 85 total, 1 running, 84 sleeping, 0 stopped, 0 zombie
  Cpu(s) : 5.7%us, 0.0%sy, 0.0%ni, 93.6%id, 0.0%wa, 0.7%hi, 0.0%si, 0.0%st
  Mem : 772068k total, 231864k used, 540204k free, 2412k buffers
  Swap : 995992k total, 0k used, 995992k free, 161316k cached

 PID    USER PR NI  VIRT  RES  SHR S %CPU %MEM    TIME+             COMMAND
5116    root 20  0 33832  22m 6576 S  5.7  3.0 0 :19.49                   X
5879 santini 20  0 16060 7344 6116 S  0.3  1.0 0 :01.06    xfce4-netload-p
   1    root 20  0  1664  568  496 S  0.0  0.1 0 :02.95                init
   2    root 20  0     0    0    0 S  0.0  0.0 0 :00.00           kthreadd
   3    root RT  0     0    0    0 S  0.0  0.0 0 :00.00        migration/0
```

Processus en ligne de commande

Occupation de la ligne de commande

- Lorsque l'on tape une commande, la ligne de commande est bloquée (plus de prompt) jusqu'à la fin de l'exécution.
- La ligne de commande est à nouveau disponible ensuite.

```
[ login@localhost ~ ] sleep 20
(il faut attendre 20 secondes avant l'apparition du
nouveau prompt)
...
...
[ login@localhost ~ ] ■
```

```
[ login@localhost ~ ] gedit
(Il faut quitter l'application ou tuer le processus gedit
pour avoir un nouveau prompt)
...
...
```

Libération de la ligne de commande

Deux façons possibles de lancer une instruction en tâche de fond :

Lancement en tâche de fond

- Les commandes qui prennent beaucoup de temps peuvent être lancées en tâche de fond pour libérer la ligne de commande du shell.
- Pour lancer directement la commande en tâche de fond il suffit de faire suivre la commande du caractère &. On retrouve immédiatement un nouveau prompt.

```
[ login@localhost ~ ] gedit &

[ login@localhost ~ ] ■
```

Relégation en tâche de fond

- Si une tâche déjà lancée occupe la ligne de commande, il est possible de suspendre son exécution en pressant la combinaison de touches Ctrl + z . La tâche est alors interrompue et on retrouve un nouveau prompt.
- Il est possible de relancer le processus en tâche de fond au moyen de la commande bg.

```
[ login@localhost ~ ] gedit
^Z
[1]+ Stopped gedit

[ login@localhost ~ ] bg
[1]+ gedit &

[ login@localhost ~ ] ■
```