Introduction aux systèmes informatiques

Laure Petrucci

IUT R&T Villetaneuse

17 septembre 2010

Plan du cours

1. Historique et généralités
2. Systèmes de fichiers
3. Commandes UNIX
4. Entrées/sorties et processus
5. Environnement utilisateur et scripts shell

1 Historique et généralités
- Différentes générations
- Composants logiciels

2 Systèmes de fichiers

3 Commandes UNIX

4 Entrées/sorties et processus

5 Environnement utilisateur et scripts shell

1ère génération (1950–1960) : *exploitation porte ouverte*

- Matériel : tubes (peu fiable, lent, encombrant)
- Programmes écrits directement en langage machine

ENIAC (1946)

2ème génération (1958–1968) : *traitement par lots*

- Transistors, cicuits imprimés
- Premiers périphériques, cartes perforées, imprimantes, bandes
- Premiers systèmes d'exploitation

UNIVAC (1954)

3ème génération (1960–1970) : *multi-programmation* puis *traitement partagé*

- Processeurs d'entrées/sorties
- Multi-programmation : plusieurs activités progressent en parallèle
- Temps partagé : interactivité

4ème génération (1970–...) : *réseaux* et *systèmes répartis*

- 1969 : premier microprocesseur
- Réseaux

Logiciels exécutés

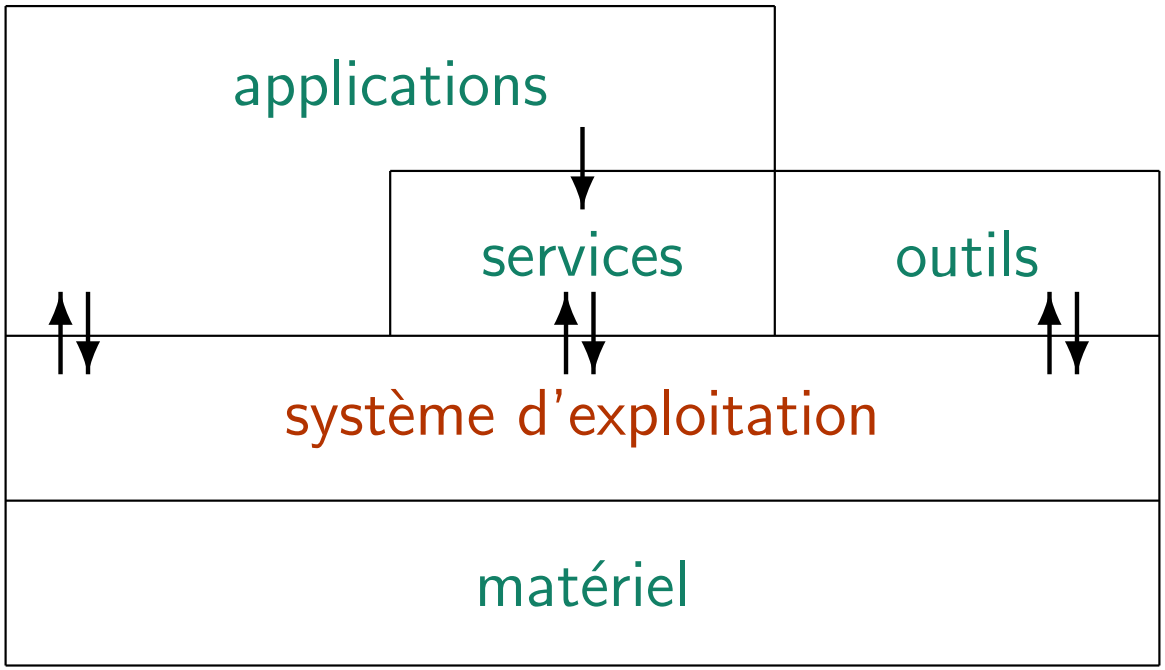
Applications : Outils achetés ou développés pour des besoins spécifiques

Logiciels de base : Services adaptés à la résolution de problèmes usuels :

- Outils : éditeurs de texte, compilateurs, ...
- Services : gestionnaires de données, fenêtres, communications, ...
- Système d'exploitation : chargement et lancement des programmes, gestion du processeur, des périphériques, ...

Exemples de systèmes d'exploitation : WINDOWS, UNIX

Interactions entre composants



Historique d'UNIX

- créé aux laboratoires Bell (USA) en 1969.
- but : gestion d'un mini-ordinateur pour une petite équipe de programmeurs.
- intéresse rapidement universités puis constructeurs.
- nombreuses versions : Linux, AIX, HPIX, SPIX, ...
- de nos jours :
  - respect de la norme POSIX $\Rightarrow$ compatibilité.
  - interface graphique $\Rightarrow$ convivialité.
- système multi-tâches, multi-utilisateurs

1 Historique et généralités

2 Systèmes de fichiers
- Fichiers et répertoires
- Structure arborescente
- Adressage des fichiers
- Commandes de manipulation des fichiers
- Méta-caractères
- Montage de disques
- Protection des fichiers
- Représentation des fichiers

3 Commandes UNIX

4 Entrées/sorties et processus

5 Environnement utilisateur et scripts shell

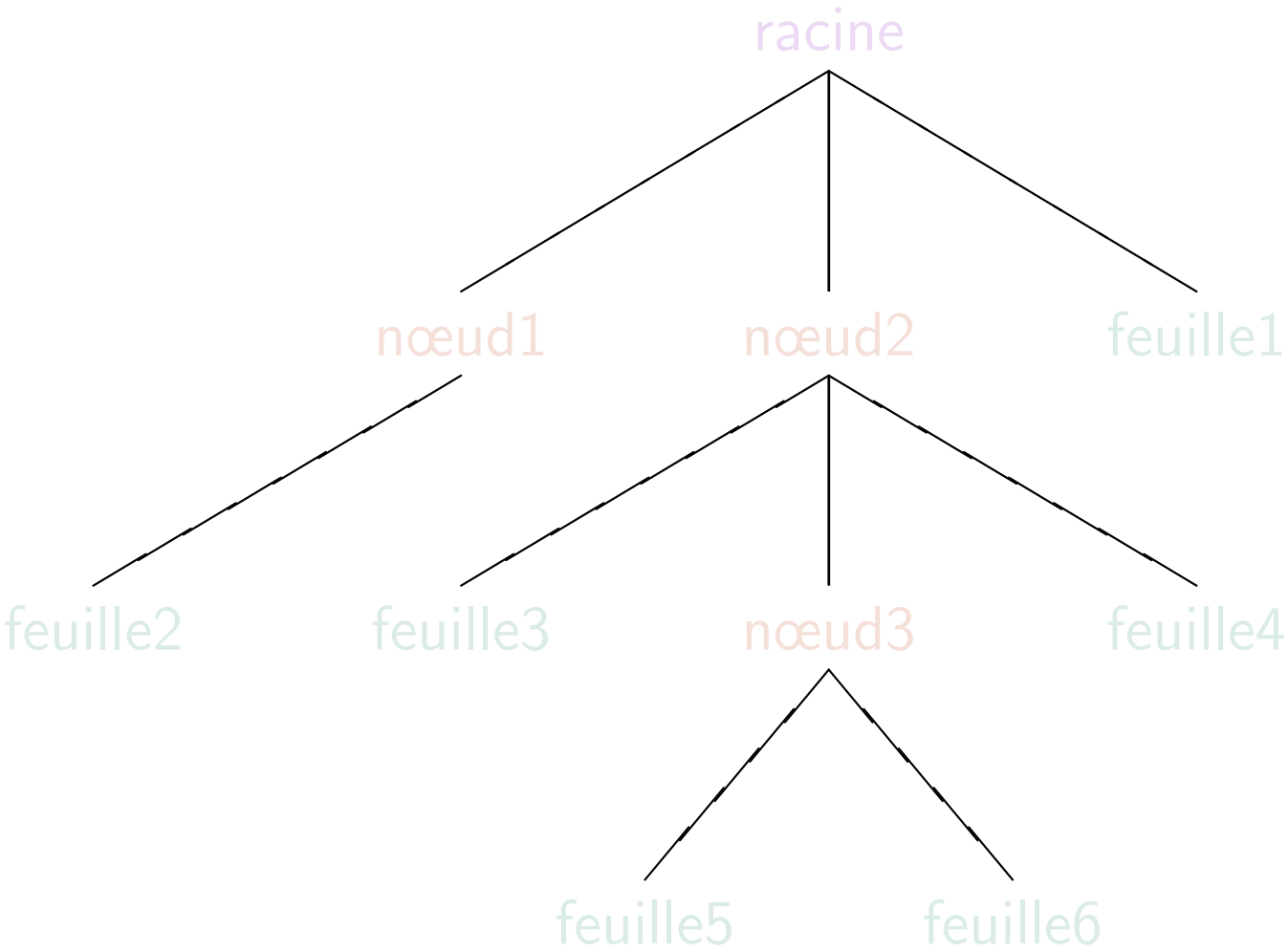
Fichiers et répertoires

Les documents sur lesquels on travaille sont stockés dans des fichiers.

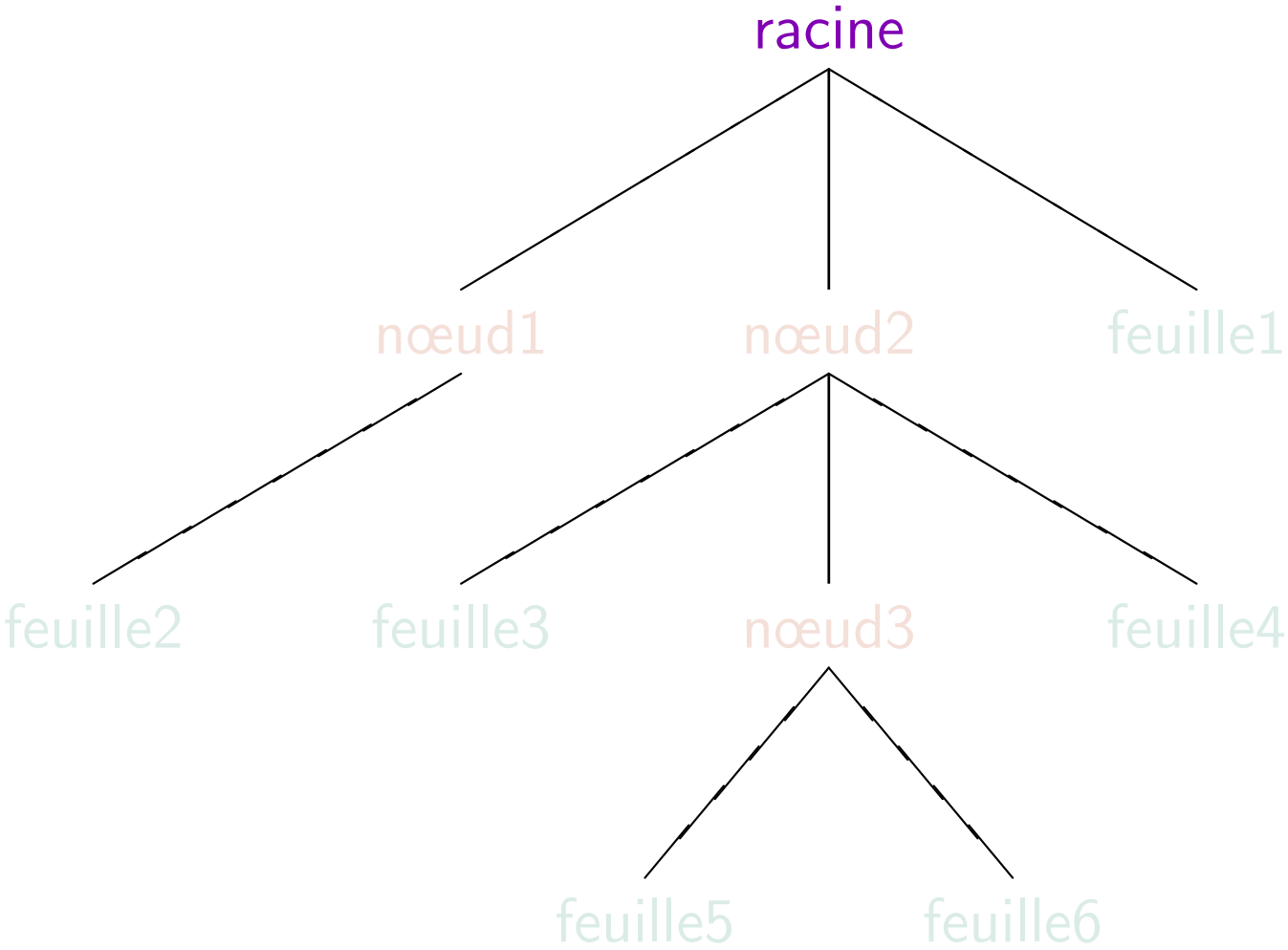
Les fichiers peuvent être regroupés dans des dossiers (également appelés répertoires ou catalogues).

L'ensemble des fichiers est stocké sur le disque selon une structure arborescente.

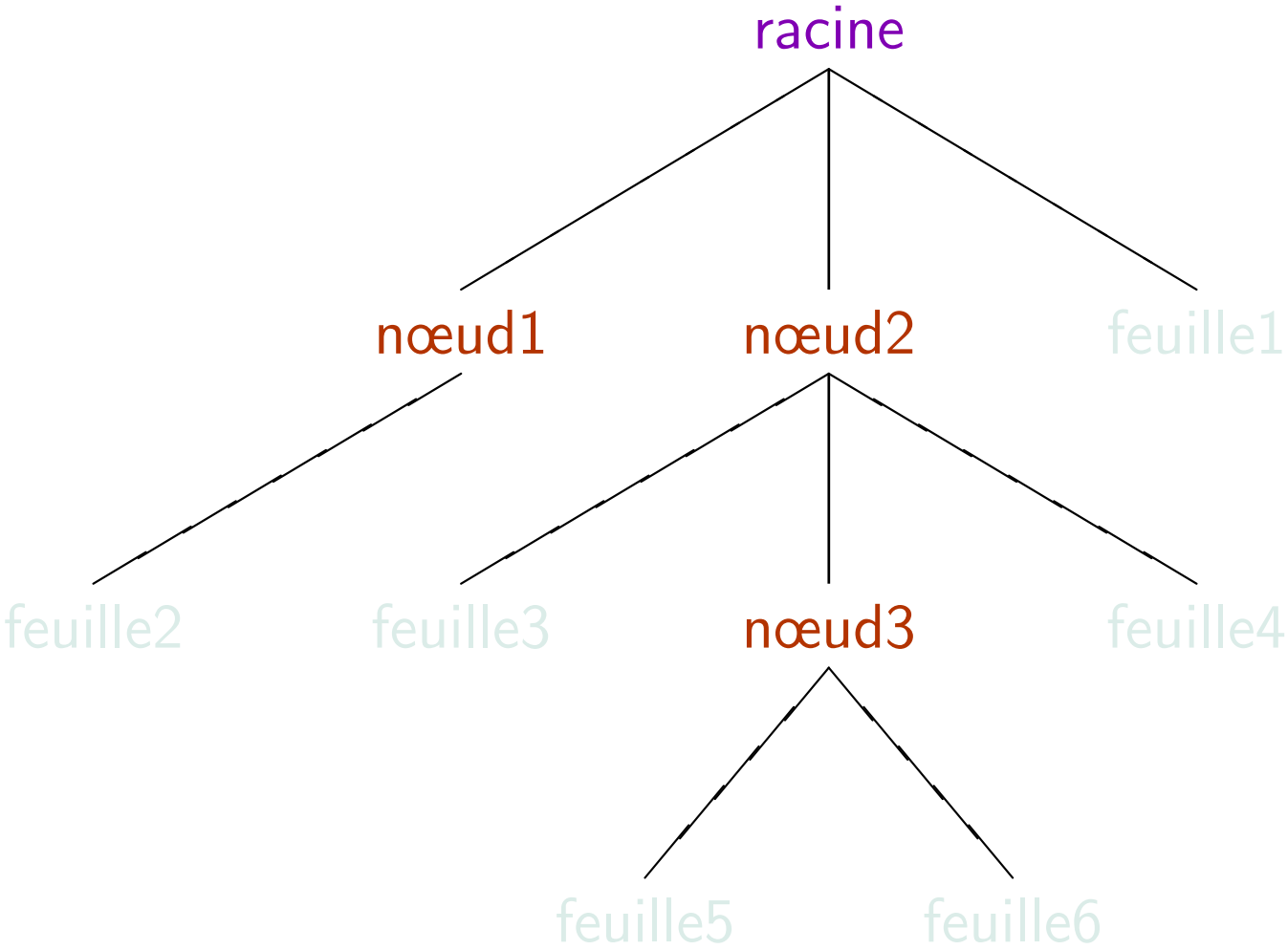
Structure arborescente



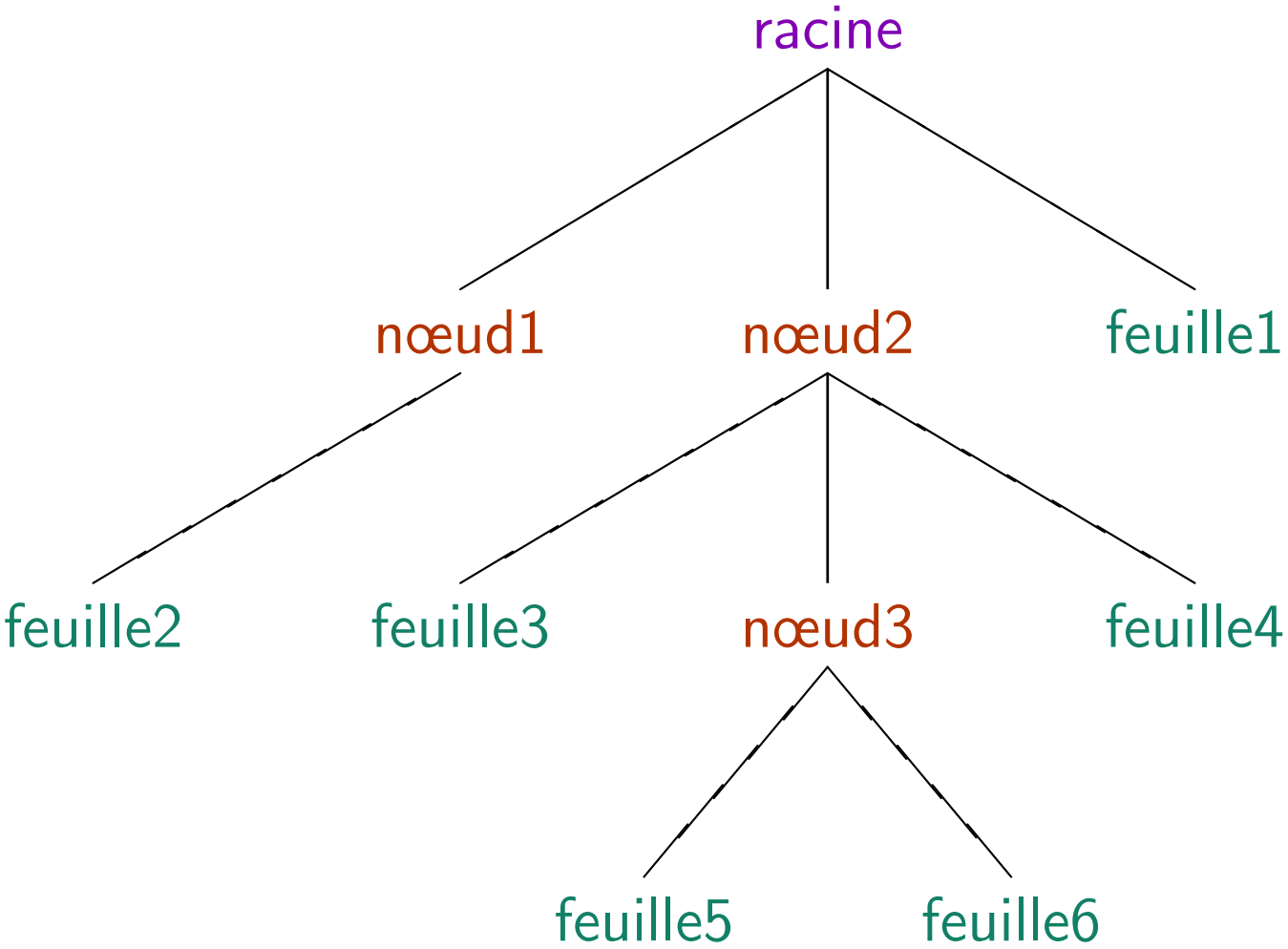
Structure arborescente



Structure arborescente



Structure arborescente



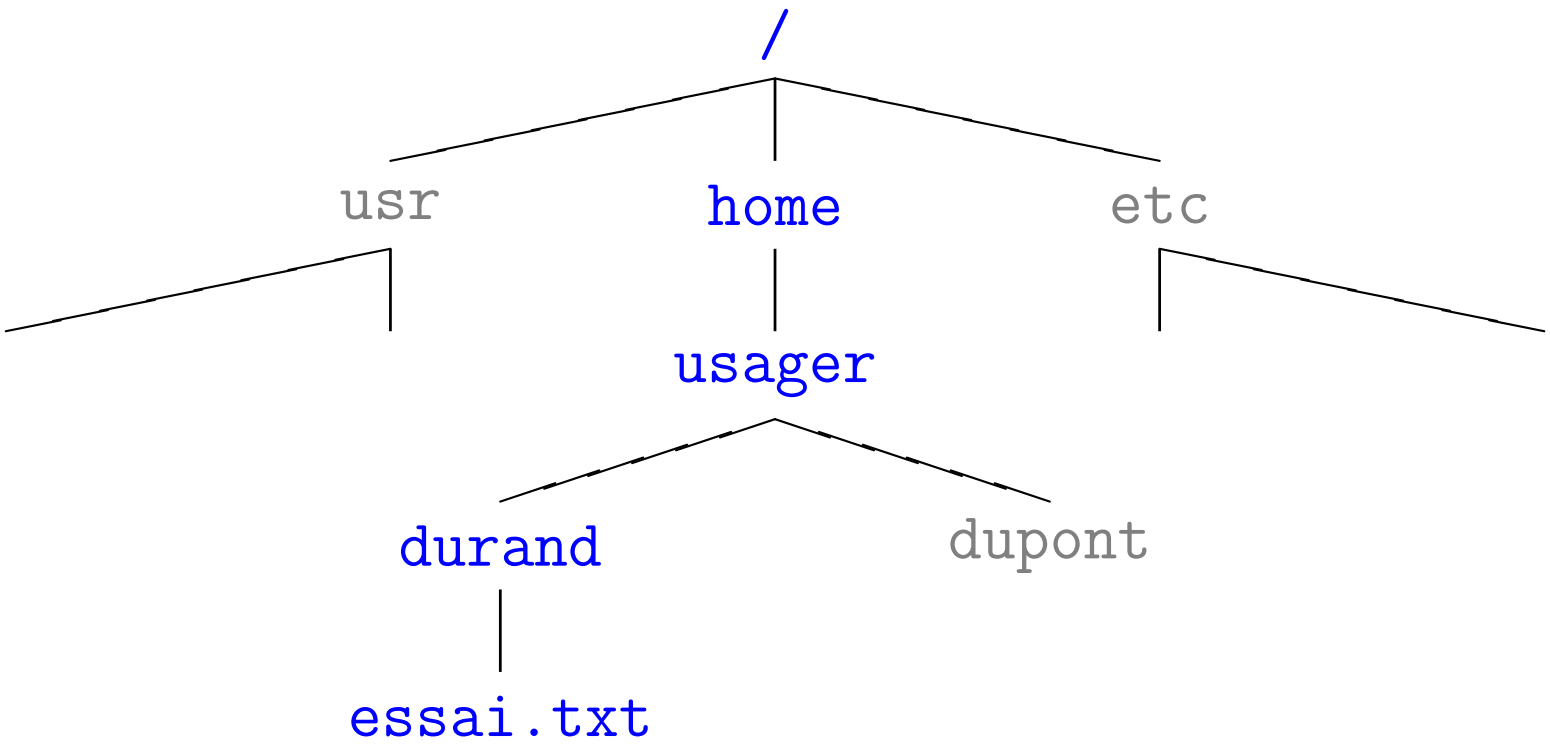
Noms des fichiers

Les noms des fichiers comportent souvent une extension : le nom se termine par un `.` suivi de quelques lettres (par exemple `.txt`).

Sous WINDOWS, l'extension indique au système le type du fichier : texte, fichier lisible par un logiciel particulier, ...

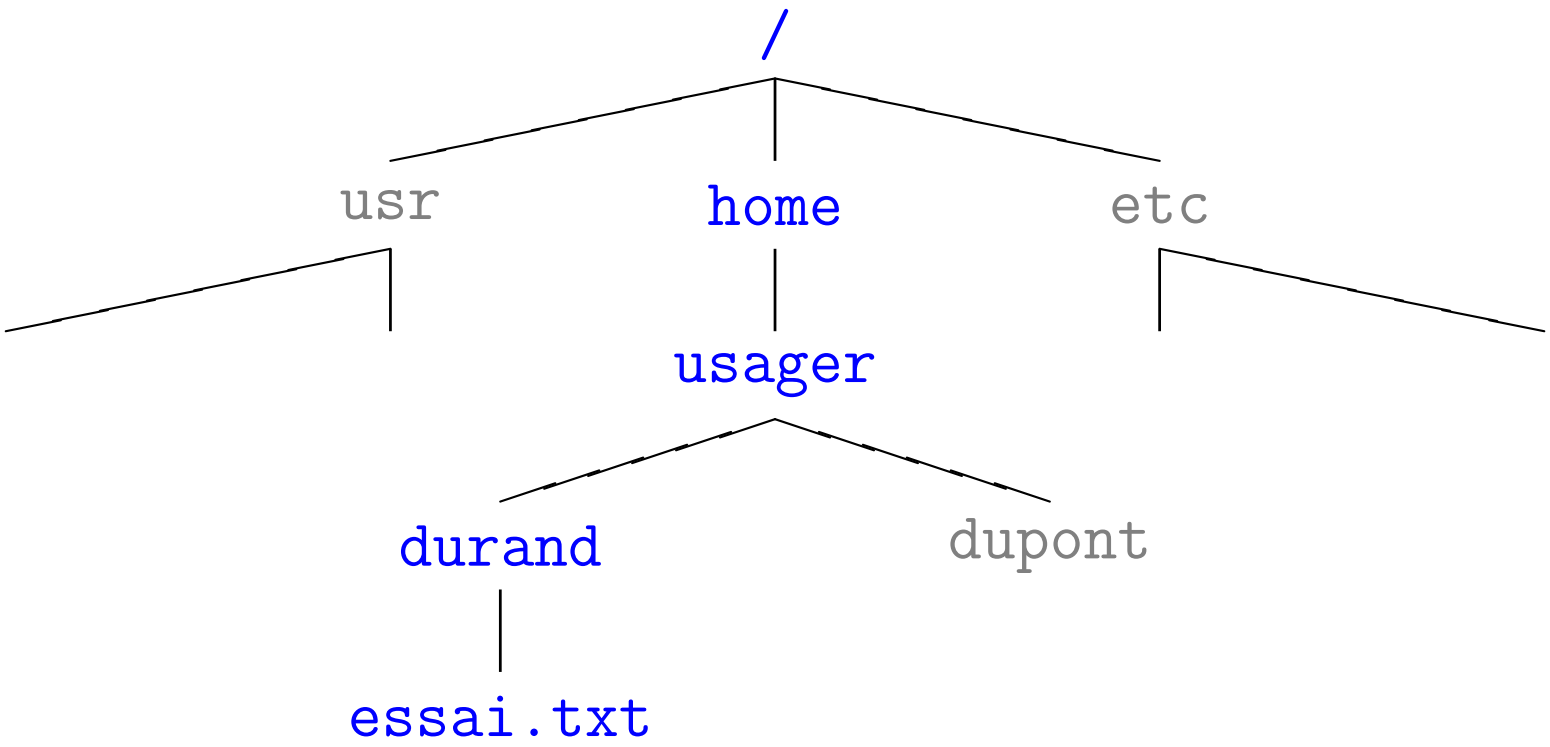
Sous UNIX, l'extension sert principalement à l'utilisateur pour se souvenir du type de fichier. Certains outils permettent d'associer une extension à une application particulière.

Référence absolue



/home/usager/durand/essai.txt

Référence absolue



/home/usager/durand/essai.txt

Référence relative

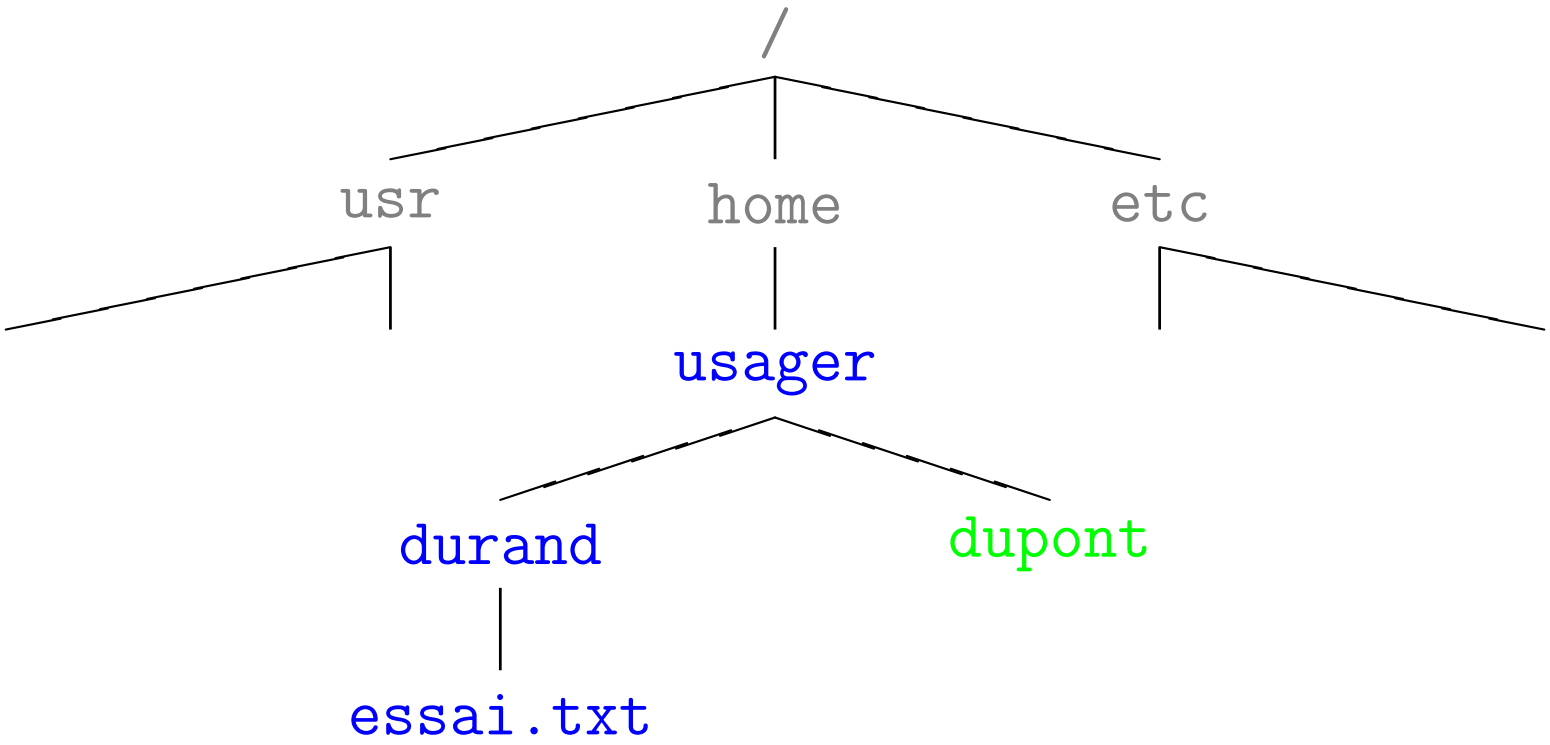
Répertoire privé (*home directory*)

Répertoire de travail (*working directory*)

Répertoire courant référencé par .

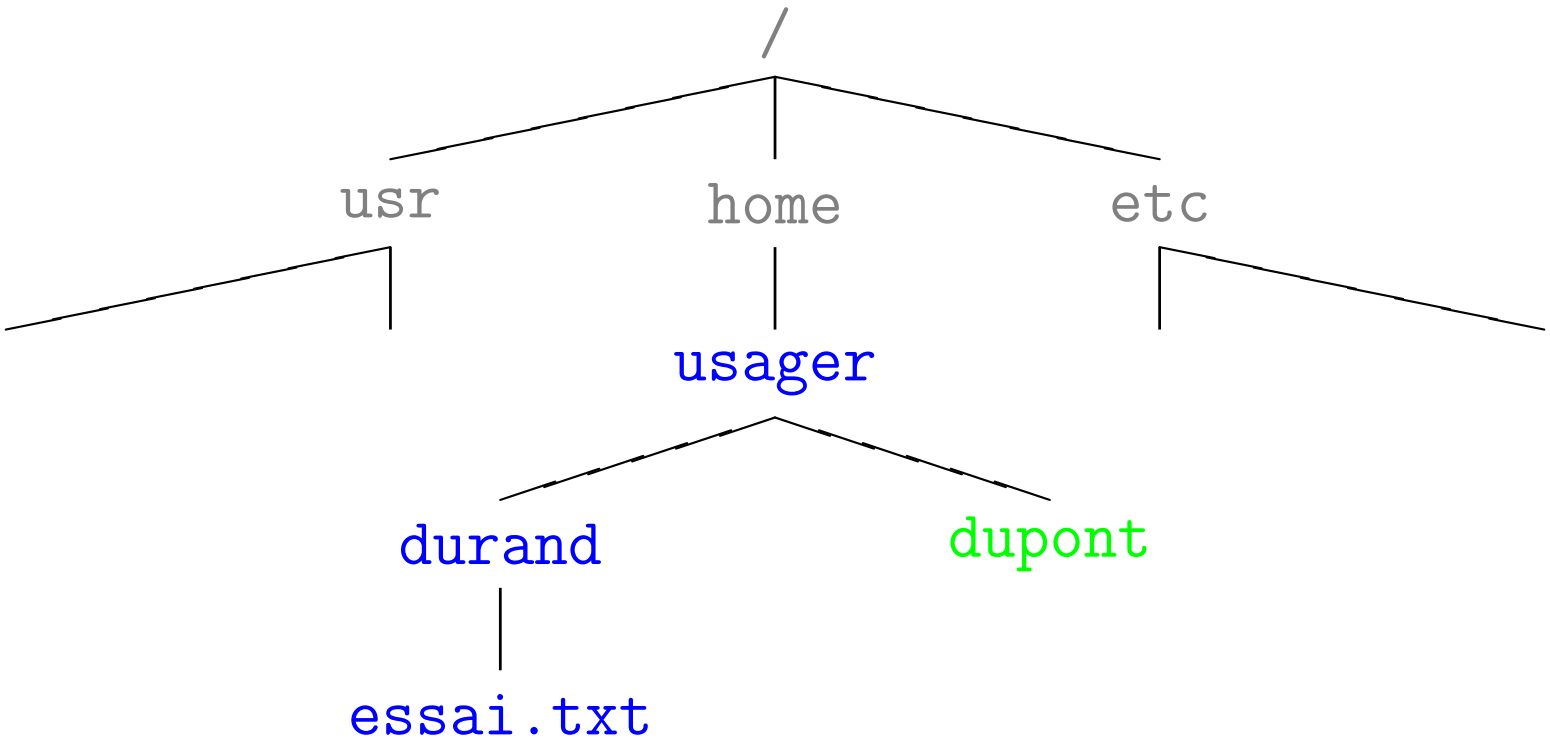
Répertoire père référencé par ..

Référence relative



../durand/essai.txt

Référence relative



../durand/essai.txt

Commandes de base

`pwd` (*print working directory*)

`pwd` affiche la référence absolue du répertoire de travail.

`cd` (*change directory*)

`cd` *référence_répertoire*

Le répertoire de travail devient celui dont la référence est *référence_répertoire* (référence absolue ou relative).

`ls` (*list*)

`ls` *liste_références*

affiche, pour tout fichier ordinaire dans *liste_références*, son nom, et pour tout répertoire dans *liste_références*, son nom et son contenu.

`ls -l` *liste_références*

affiche également des informations sur le fichier telles que sa taille, la date de dernière modification, ...

Commandes de base

`pwd` (*print working directory*)

`pwd` affiche la référence absolue du répertoire de travail.

`cd` (*change directory*)

`cd` *référence_répertoire*

Le répertoire de travail devient celui dont la référence est *référence_répertoire* (référence absolue ou relative).

`ls` (*list*)

`ls` *liste_références*

affiche, pour tout fichier ordinaire dans *liste_références*, son nom, et pour tout répertoire dans *liste_références*, son nom et son contenu.

`ls -l` *liste_références*

affiche également des informations sur le fichier telles que sa taille, la date de dernière modification, ...

Commandes de base

`pwd` (*print working directory*)

`pwd` affiche la référence absolue du répertoire de travail.

`cd` (*change directory*)

`cd` *référence_répertoire*

Le répertoire de travail devient celui dont la référence est *référence_répertoire* (référence absolue ou relative).

`ls` (*list*)

`ls` *liste_références*

affiche, pour tout fichier ordinaire dans *liste_références*, son nom, et pour tout répertoire dans *liste_références*, son nom et son contenu.

`ls -l` *liste_références*

affiche également des informations sur le fichier telles que sa taille, la date de dernière modification, ...

Démo

Copie, renommage et suppression

cp (*copy*)

`cp` *ancien nouveau*

copie le fichier de référence *ancien* dans un fichier de référence *nouveau*.

mv (*move*)

`mv` *ancien_nom nouveau_nom*

attribue au fichier de référence *ancien_nom* la référence *nouveau_nom*.

rm (*remove*)

`rm` *liste_références*

supprime toutes les références dans *liste_références*.

Copie, renommage et suppression

cp (*copy*)

`cp` *ancien nouveau*

copie le fichier de référence *ancien* dans un fichier de référence *nouveau*.

mv (*move*)

`mv` *ancien_nom nouveau_nom*

attribue au fichier de référence *ancien_nom* la référence *nouveau_nom*.

rm (*remove*)

`rm` *liste_références*

supprime toutes les références dans *liste_références*.

Copie, renommage et suppression

`cp` (*copy*)

`cp` *ancien nouveau*

copie le fichier de référence *ancien* dans un fichier de référence *nouveau*.

`mv` (*move*)

`mv` *ancien_nom nouveau_nom*

attribue au fichier de référence *ancien_nom* la référence *nouveau_nom*.

`rm` (*remove*)

`rm` *liste_références*

supprime toutes les références dans *liste_références*.

Démo

Création et suppression de répertoire

`mkdir` (*make directory*)

`mkdir` *référence_répertoire*

crée un répertoire dont la référence est *référence_répertoire* (référence absolue ou relative).

`rmdir` (*remove directory*)

`rmdir` *liste_références*

supprime tous les répertoires **vides** dont la référence est dans *liste_références*.

Création et suppression de répertoire

`mkdir` (*make directory*)

`mkdir` *référence_répertoire*

crée un répertoire dont la référence est *référence_répertoire* (référence absolue ou relative).

`rmdir` (*remove directory*)

`rmdir` *liste_références*

supprime tous les répertoires **vides** dont la référence est dans *liste_références*.

Démo

Méta-caractères

`*` : toute chaîne de caractères ne commençant pas par un `.`

`?` : un caractère quelconque

`[...]` : un caractère quelconque parmi ceux entre crochets.

`[c-j]` : un caractère quelconque dans la plage allant du caractère `c` au caractère `j`.

`{mot1,mot2,mot3}` : une chaîne de caractères égale à `mot1`, ou `mot2`, ou `mot3`.

Montage de disques

Arborescence unique : disque logique

Disques physiques : sous-arborescences

Monter un disque : inclure l'arborescence du disque physique dans l'arborescence générale.

`mount` *disque point_de_montage*

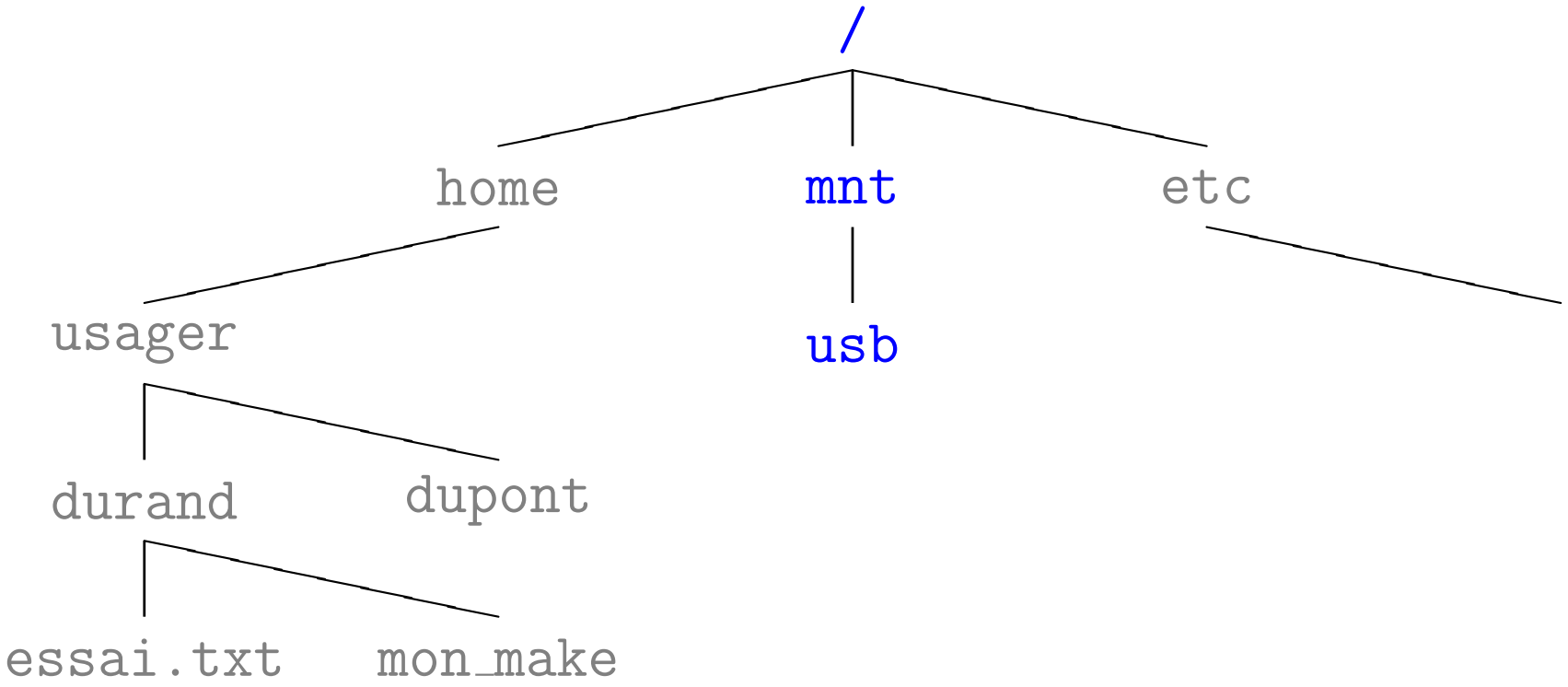
monte le *disque* à l'endroit référencé par *point_de_montage* dans l'arborescence générale.

Démonter un disque : retirer son arborescence de l'arborescence générale.

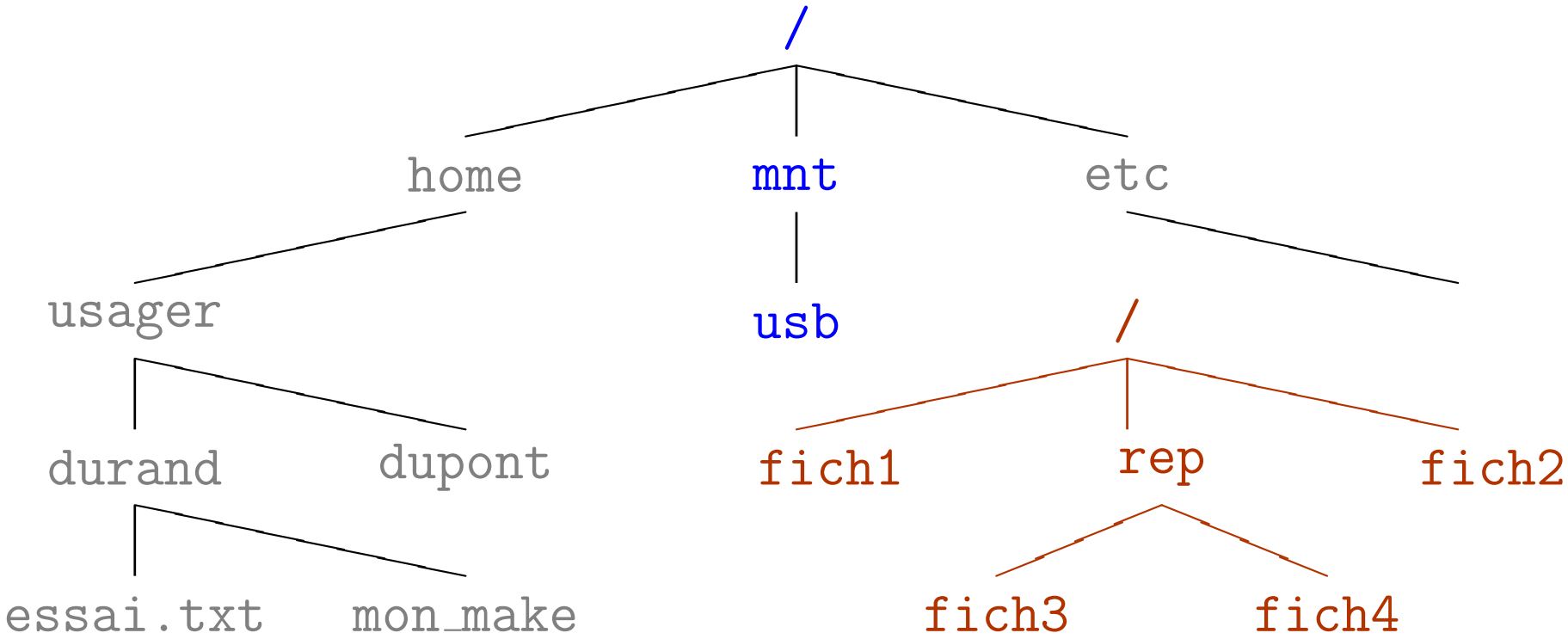
`umount` *disque*

démonte le *disque*.

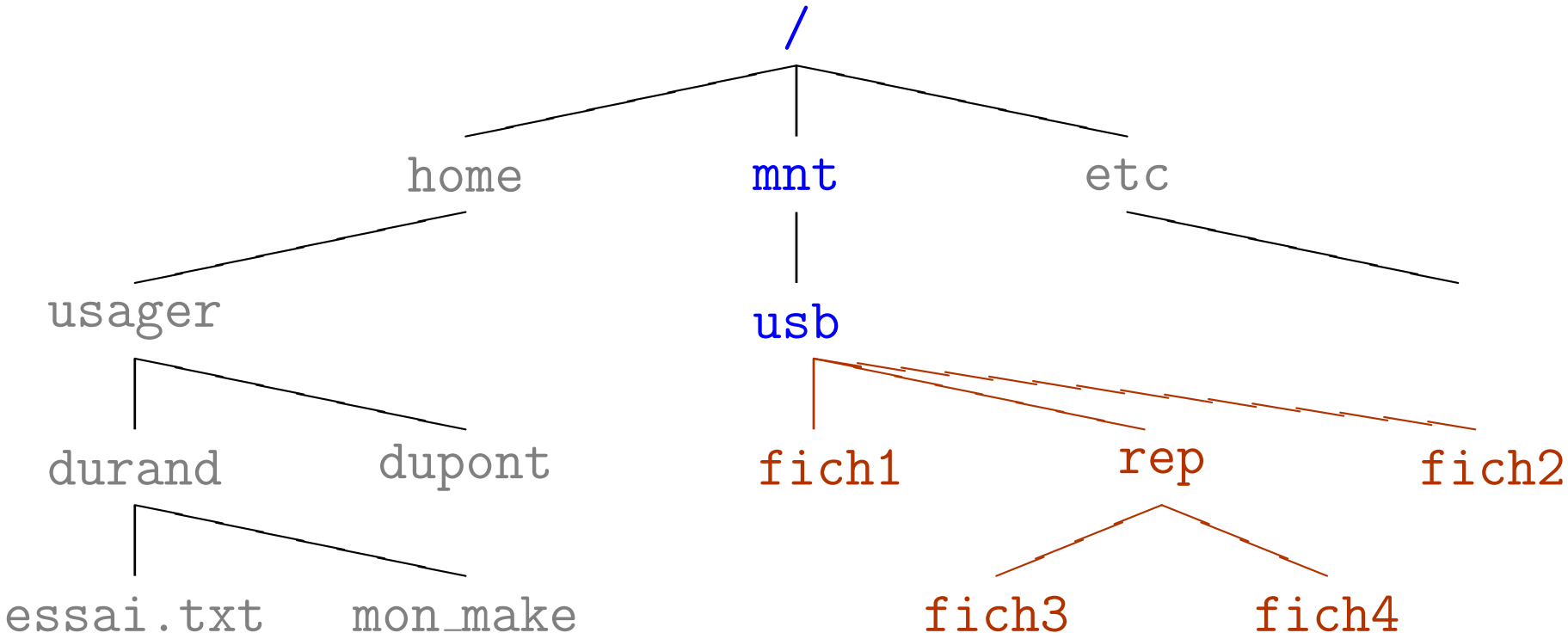
Montage de clé USB



Montage de clé USB



Montage de clé USB



Droits d'accès aux fichiers

Tout utilisateur :

- possède un numéro d'utilisateur (`uid` – user identification)
- appartient à au moins un groupe (`gid` – group identification).

Droits d'accès aux fichiers

Il y a 3 types d'utilisateurs de fichiers :

- propriétaire (user) : l'utilisateur propriétaire du fichier.
- groupe (group) : les utilisateurs appartenant au même groupe que le fichier.
- autres (other) : tous les autres utilisateurs.

À chaque fichier sont associés différents droits :

- lecture (read) : possibilité de lire le fichier ou de regarder le contenu du répertoire.
- écriture (write) : possibilité d'écrire le fichier ou d'écrire des fichiers dans le répertoire.
- exécution (execute) : possibilité d'exécuter le fichier ou de traverser le répertoire.

Droits d'accès aux fichiers

Il y a 3 types d'utilisateurs de fichiers :

- propriétaire (user) : l'utilisateur propriétaire du fichier.
- groupe (group) : les utilisateurs appartenant au même groupe que le fichier.
- autres (other) : tous les autres utilisateurs.

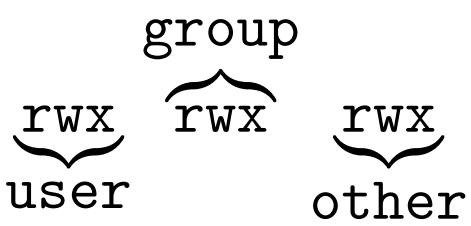
À chaque fichier sont associés différents droits :

- lecture (read) : possibilité de lire le fichier ou de regarder le contenu du répertoire.
- écriture (write) : possibilité d'écrire le fichier ou d'écrire des fichiers dans le répertoire.
- exécution (execute) : possibilité d'exécuter le fichier ou de traverser le répertoire.

Droits d'accès aux fichiers

9 couples (*type d'utilisateur*, *droit d'accès au fichier*)

$\Rightarrow$ les droits d'accès sont codés sur 9 bits (ils sont indiqués par la commande `ls -l`).



Codage en octal :

`rwxr-x---` = `111101000` = `750`

Modification des droits d'accès

`chmod` (change mode)

`chmod` *protections liste_fichiers*

Pour chaque fichier référencé dans *liste_fichiers*, les protections du fichier deviennent *protections*.

Les *protections* peuvent être caractérisées de 2 manières :

- code octal
- *utilisateur opération droits* où :

  utilisateur : `u` (user), `g` (group), `o` (other)
  opération : `+` (ajout), `-` (suppression), `=` (égale)
  droits : `r` (read), `w` (write), `x` (execute)

Changements de propriétaire et de groupe

`chown` (change owner)

`chown` *utilisateur liste_fichiers*

Pour chaque fichier référencé dans *liste_fichiers*, le propriétaire du fichier devient *utilisateur*.

`chown` *utilisateur.groupe liste_fichiers*

change également le groupe auquel le fichier appartient.

`chown -R` *utilisateur.groupe liste_fichiers*

Avec l'option `-R`, la modification est également apportée aux fichiers contenus dans les répertoires de *liste_fichiers*.

`chgrp` (change group)

`chgrp` *groupe liste_fichiers*

change le groupe auxquels les fichiers de *liste_fichiers* appartiennent.

Changements de propriétaire et de groupe

`chown` (change owner)

`chown` *utilisateur liste_fichiers*

Pour chaque fichier référencé dans *liste_fichiers*, le propriétaire du fichier devient *utilisateur*.

`chown` *utilisateur.groupe liste_fichiers*

change également le groupe auquel le fichier appartient.

`chown -R` *utilisateur.groupe liste_fichiers*

Avec l'option `-R`, la modification est également apportée aux fichiers contenus dans les répertoires de *liste_fichiers*.

`chgrp` (change group)

`chgrp` *groupe liste_fichiers*

change le groupe auxquels les fichiers de *liste_fichiers* appartiennent.

Autres permissions

3 bits spéciaux :

- **set-uid** permet d'exécuter un fichier avec les privilèges de son propriétaire et non pas ceux de l'utilisateur qui lance l'exécution.
- **set-gid** même chose avec le groupe.
- **bit de collage** (sticky bit) assure le maintien de l'exécutable en mémoire même lorsqu'aucune exécution n'est en cours.

Types de fichiers

fichiers ordinaires : programmes, données. Un fichier est décrit par un i-nœud.

répertoires : ensemble de fichiers. Le contenu d'un répertoire est un ensemble de couples (nom_fichier,i-nœud).

fichiers spéciaux : spécifient les périphériques. Ces fichiers sont vus par l'utilisateur comme des fichiers ordinaires.

Structure d'un i-nœud

i-nœud = descripteur de fichier

- taille en nombre d'octets
- adresse sur le disque
- identification du propriétaire
- permissions d'accès : lecture, écriture, exécution
- type de fichier
- date de dernière modification
- compteur de références

Liens

Un lien permet de désigner un fichier à partir de plusieurs endroits différents dans l'arborescence, et éventuellement avec des noms différents.

compteur de références = nombre de façons de désigner un même fichier.

La suppression d'un fichier n'est effective que lorsque le compteur de références est nul.

`ln` (*link*)

`ln` *ancien nouveau*

crée un lien de référence *nouveau* vers le fichier de référence *ancien*.

Liens

Un lien permet de désigner un fichier à partir de plusieurs endroits différents dans l'arborescence, et éventuellement avec des noms différents.

compteur de références = nombre de façons de désigner un même fichier.

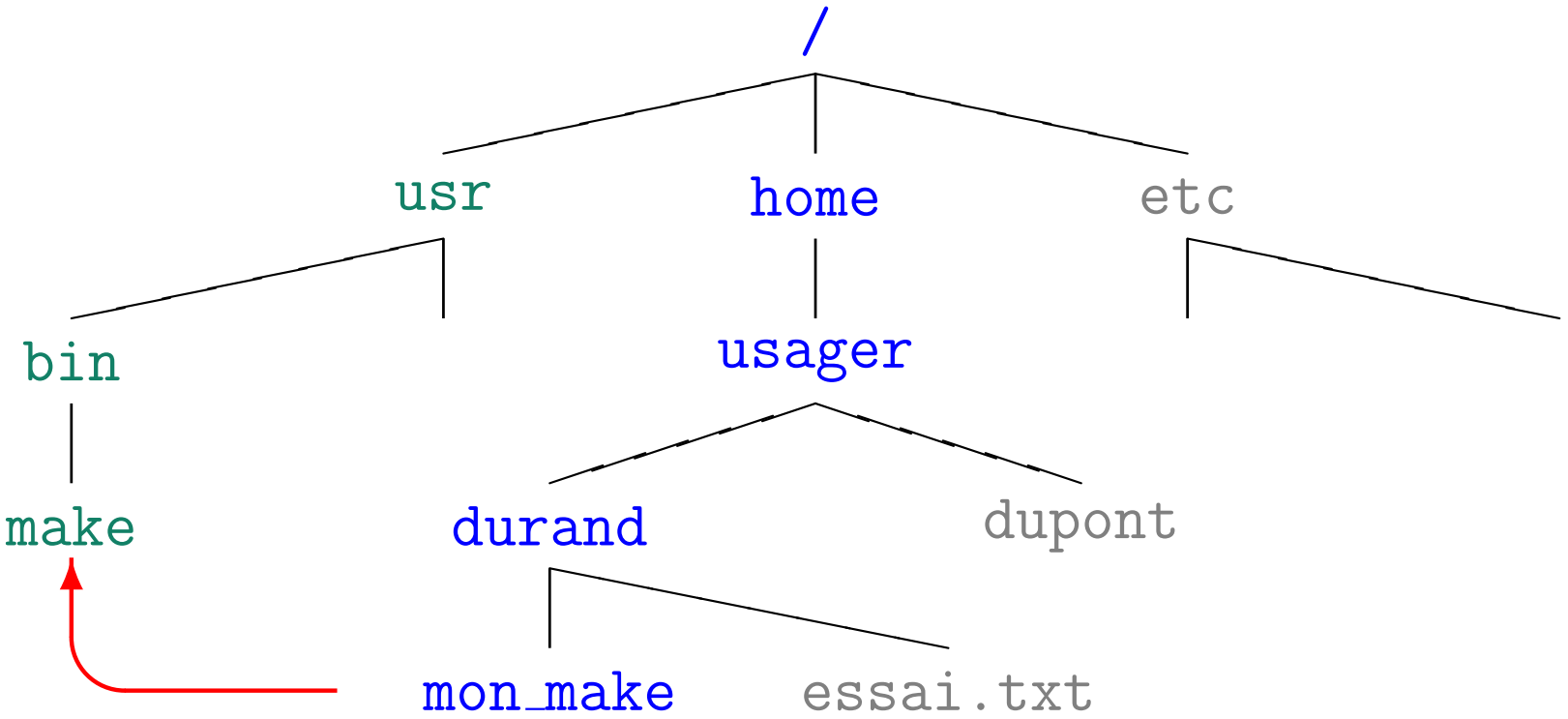
La suppression d'un fichier n'est effective que lorsque le compteur de références est nul.

`ln` (*link*)

`ln` *ancien nouveau*

crée un lien de référence *nouveau* vers le fichier de référence *ancien*.

Exemple de lien



1 Historique et généralités

2 Systèmes de fichiers

3 Commandes UNIX
- Manuel en ligne
- Visualisation de fichiers
- Manipulation de lignes de fichier
- Recherche de fichiers

4 Entrées/sorties et processus

5 Environnement utilisateur et scripts shell

Manuel en ligne

`man` *comm*

affiche page à page le chapitre du manuel sur la commande *comm*.

`man -k` *sujet*

permet d'obtenir une documentation sur le *sujet*.

Visualisation de fichiers

`more` *liste_fichiers*

affiche page à page le contenu des fichiers référencés dans *liste_fichiers*.

`cat` *liste_fichiers*

affiche sur la sortie standard le contenu des fichiers dans *liste_fichiers*, ou reproduit l'entrée standard si la liste est vide.

`diff` *fichier1 fichier2*

affiche les lignes de *fichier1* et *fichier2* qui diffèrent.

`head` *liste_fichiers*

affiche les premières lignes des fichiers référencés.

`tail` *liste_fichiers*

affiche les dernières lignes des fichiers référencés.

Visualisation de fichiers

`more` *liste_fichiers*

affiche page à page le contenu des fichiers référencés dans *liste_fichiers*.

`cat` *liste_fichiers*

affiche sur la sortie standard le contenu des fichiers dans *liste_fichiers*, ou reproduit l'entrée standard si la liste est vide.

`diff` *fichier1 fichier2*

affiche les lignes de *fichier1* et *fichier2* qui diffèrent.

`head` *liste_fichiers*

affiche les premières lignes des fichiers référencés.

`tail` *liste_fichiers*

affiche les dernières lignes des fichiers référencés.

Visualisation de fichiers

`more` *liste_fichiers*

affiche page à page le contenu des fichiers référencés dans *liste_fichiers*.

`cat` *liste_fichiers*

affiche sur la sortie standard le contenu des fichiers dans *liste_fichiers*, ou reproduit l'entrée standard si la liste est vide.

`diff` *fichier1 fichier2*

affiche les lignes de *fichier1* et *fichier2* qui diffèrent.

`head` *liste_fichiers*

affiche les premières lignes des fichiers référencés.

`tail` *liste_fichiers*

affiche les dernières lignes des fichiers référencés.

Manipulation de lignes de fichier

`wc` *liste_fichiers* (word count)

affiche sur la sortie standard le nombre de lignes, mots et caractères des fichiers dans *liste_fichiers*, ou de l'entrée standard si la liste est vide.

`tr` *chaîne1 chaîne2* (transpose)

copie son entrée standard sur sa sortie standard en remplaçant toutes les occurrences de *chaîne1* par *chaîne2*.

Manipulation de lignes de fichier

`wc` *liste_fichiers* (word count)

affiche sur la sortie standard le nombre de lignes, mots et caractères des fichiers dans *liste_fichiers*, ou de l'entrée standard si la liste est vide.

`tr` *chaîne1 chaîne2* (transpose)

copie son entrée standard sur sa sortie standard en remplaçant toutes les occurrences de *chaîne1* par *chaîne2*.

Manipulation de lignes de fichier

`cut` *options liste_fichiers*

imprime sur la sortie standard, les parties de lignes des fichiers référencés, comme spécifié par les *options*.

```
cut -d. -f3 fich.txt
```

sélectionne, dans le fichier `fich.txt` le troisième *champ* (option `-f3`, f=field) de chaque ligne, les champs étant délimités par des `.` (option `-d.`, d=delimiter).

`sort` *liste_fichiers*

trie les lignes des fichiers référencés. Attention : s'il y a plusieurs fichiers, leurs lignes sont mélangées.

`uniq` *liste_fichiers*

quand plusieurs lignes identiques se suivent, n'en garde qu'une. Donc, supprime les lignes dupliquées dans un fichier trié.

Manipulation de lignes de fichier

`cut` *options liste_fichiers*

imprime sur la sortie standard, les parties de lignes des fichiers référencés, comme spécifié par les *options*.

```
cut -d. -f3 fich.txt
```

sélectionne, dans le fichier `fich.txt` le troisième *champ* (option `-f3`, f=field) de chaque ligne, les champs étant délimités par des `.` (option `-d.`, d=delimiter).

`sort` *liste_fichiers*

trie les lignes des fichiers référencés. Attention : s'il y a plusieurs fichiers, leurs lignes sont mélangées.

`uniq` *liste_fichiers*

quand plusieurs lignes identiques se suivent, n'en garde qu'une. Donc, supprime les lignes dupliquées dans un fichier trié.

Manipulation de lignes de fichier

`cut` *options liste_fichiers*

imprime sur la sortie standard, les parties de lignes des fichiers référencés, comme spécifié par les *options*.

```
cut -d. -f3 fich.txt
```

sélectionne, dans le fichier `fich.txt` le troisième *champ* (option `-f3`, f=field) de chaque ligne, les champs étant délimités par des `.` (option `-d.`, d=delimiter).

`sort` *liste_fichiers*

trie les lignes des fichiers référencés. Attention : s'il y a plusieurs fichiers, leurs lignes sont mélangées.

`uniq` *liste_fichiers*

quand plusieurs lignes identiques se suivent, n'en garde qu'une. Donc, supprime les lignes dupliquées dans un fichier trié.

Recherche de fichiers

`grep` *chaîne liste_fichiers*

recherche, dans les fichiers référencés, les lignes contenant *chaîne*.

`find` *répertoire critères*

recherche tous les fichiers de la sous-arborescence de *répertoire* satisfaisant les *critères* indiqués.

```
find ~ -name "t*a*"
```

recherche, dans l'arborescence de l'utilisateur, tous les fichiers dont le nom commence par `t` et contient `a`.

```
find ~ -name "t*a*" -exec grep coucou {} \;
```

recherche, dans l'arborescence de l'utilisateur, tous les fichiers dont le nom commence par `t` et contient `a`. Puis, parmi les fichiers dont le nom convient, sélectionne ceux contenant la chaîne de caractères `coucou`.

Recherche de fichiers

`grep` *chaîne liste_fichiers*

recherche, dans les fichiers référencés, les lignes contenant *chaîne*.

`find` *répertoire critères*

recherche tous les fichiers de la sous-arborescence de *répertoire* satisfaisant les *critères* indiqués.

```
find ~ -name "t*a*"
```

recherche, dans l'arborescence de l'utilisateur, tous les fichiers dont le nom commence par `t` et contient `a`.

```
find ~ -name "t*a*" -exec grep coucou {} \;
```

recherche, dans l'arborescence de l'utilisateur, tous les fichiers dont le nom commence par `t` et contient `a`. Puis, parmi les fichiers dont le nom convient, sélectionne ceux contenant la chaîne de caractères `coucou`.

1 Historique et généralités

2 Systèmes de fichiers

3 Commandes UNIX

4 Entrées/sorties et processus
- Entrées/sorties, redirections
- Processus
- Signaux

5 Environnement utilisateur et scripts shell

Entrées/sorties

- Entrées : données fournies à une commande
- Sorties : ce qui est écrit par la commande

Les entrées et sorties se font a priori sur des canaux spécifiques :

entrée standard associée au clavier

sortie standard associée à l'écran

sortie erreur standard également associée à l'écran

Entrées/sorties

- Entrées : données fournies à une commande
- Sorties : ce qui est écrit par la commande

Les entrées et sorties se font a priori sur des canaux spécifiques :

entrée standard associée au clavier

sortie standard associée à l'écran

sortie erreur standard également associée à l'écran

Redirections : pourquoi ?

On peut vouloir modifier les entrées/sorties, parce que, par exemple :

- les entrées sont contenues dans un fichier
- les sorties sont trop longues pour être lues à l'écran, donc on veut les mettre dans un fichier

$\Rightarrow$ on redirige le canal associé.

Redirection des entrées

Redirection de l'entrée standard

comm < *nom_fichier*

La commande *comm* prend ses entrées dans le fichier référencé par *nom_fichier*.

Redirection des sorties

Redirection de la sortie standard

comm > *nom_fichier*

redirige les sorties de la commande *comm* sur le fichier référencé par *nom_fichier*. Ce fichier est créé s'il n'existe pas ou écrasé s'il existe déjà.

comm >> *nom_fichier*

redirige les sorties de la commande *comm* sur le fichier référencé par *nom_fichier*. Ce fichier est créé s'il n'existe pas ou les sorties sont écrites à la fin du fichier s'il existe déjà.

Redirection de la sortie erreur standard

comm 2> *nom_fichier*

redirige les erreurs générées lors de l'exécution de la commande *comm*.

Redirection des sorties

Redirection de la sortie standard

comm > *nom_fichier*

redirige les sorties de la commande *comm* sur le fichier référencé par *nom_fichier*. Ce fichier est créé s'il n'existe pas ou écrasé s'il existe déjà.

comm >> *nom_fichier*

redirige les sorties de la commande *comm* sur le fichier référencé par *nom_fichier*. Ce fichier est créé s'il n'existe pas ou les sorties sont écrites à la fin du fichier s'il existe déjà.

Redirection de la sortie erreur standard

comm 2> *nom_fichier*

redirige les erreurs générées lors de l'exécution de la commande *comm*.

Processus

- Un processus est l'activité liée à l'exécution d'un programme.
- Un utilisateur peut avoir plusieurs processus en cours à un instant donné.
- Les différents processus existant à un instant donné sont indépendants et le processeur leur est attribué de façon imprévisible pour l'utilisateur.
- Un interpréteur de commandes (*shell*) est lancé lorsqu'un utilisateur se connecte.

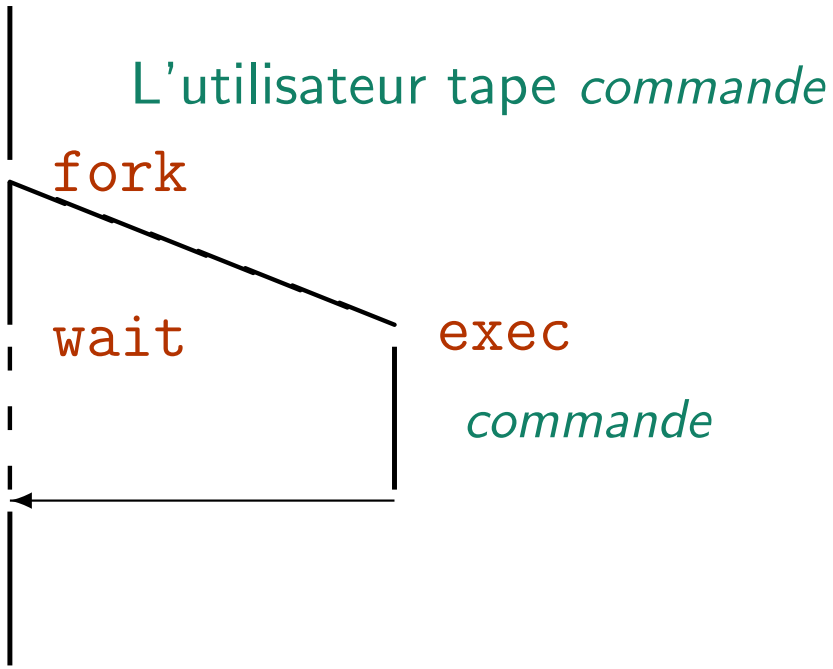
Exécution de commandes

Deux cas possibles :

commande interne : l'action est exécutée par l'interpréteur de commandes lui-même ;

commande externe : le nom de l'action est le nom d'un fichier contenant un programme exécutable. Le processus *shell* est dupliqué, et sa copie est remplacée par l'exécutable de la commande. Le processus d'origine attend la fin de l'exécution de la commande.

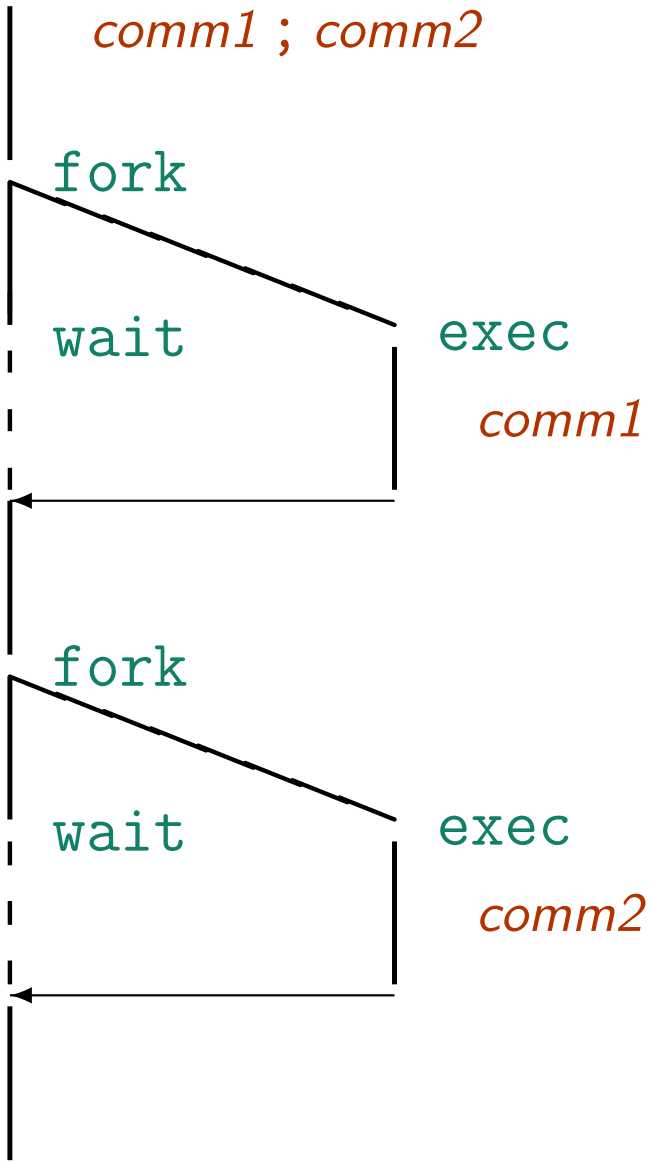
Exécution de commande externe



Enchaînements de processus

- séquentiel : *comm1* ; *comm2*
- parallèle : *comm1* | *comm2*
- tâche de fond : *commande* &

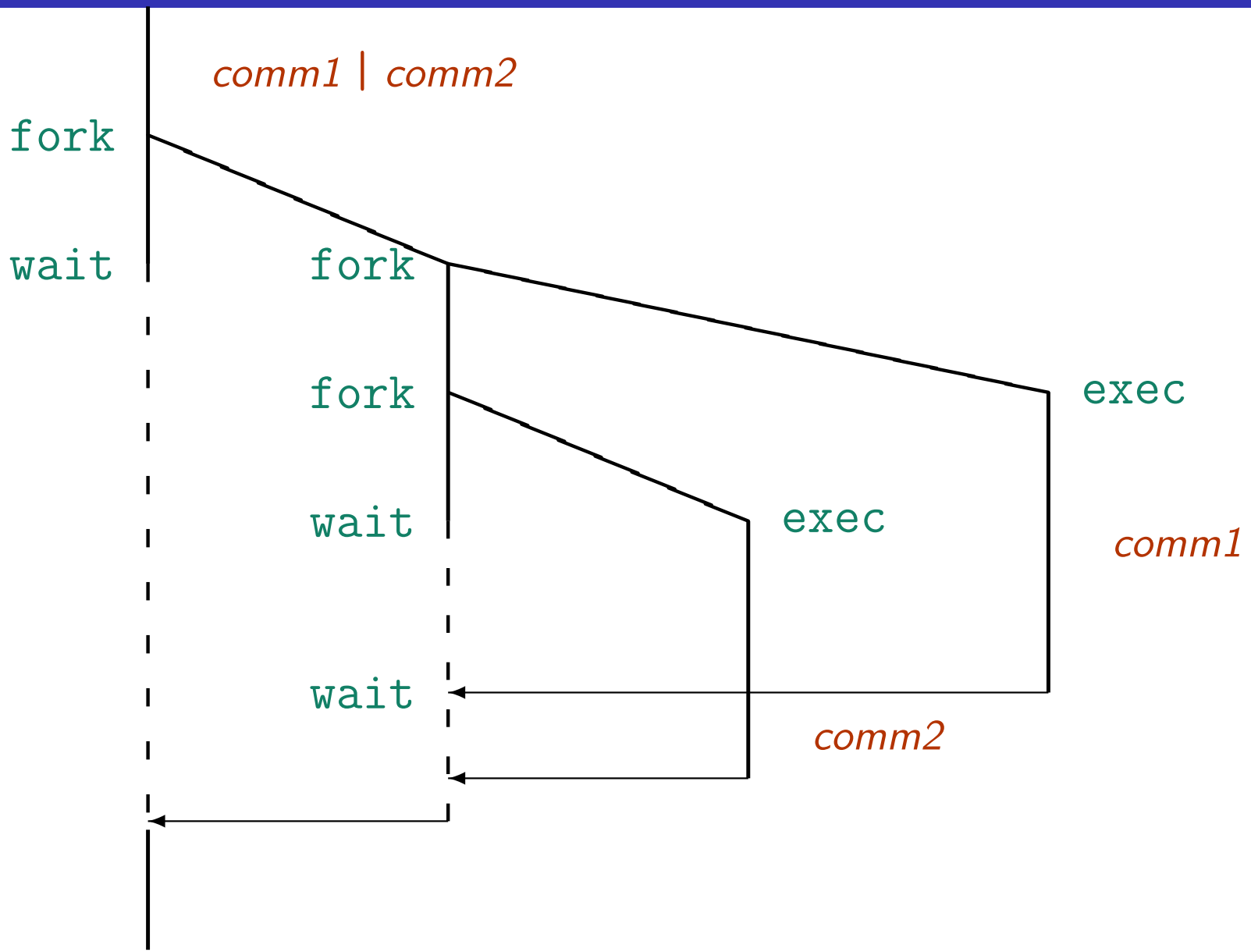
Enchaînement séquentiel



Enchaînement parallèle — tubes

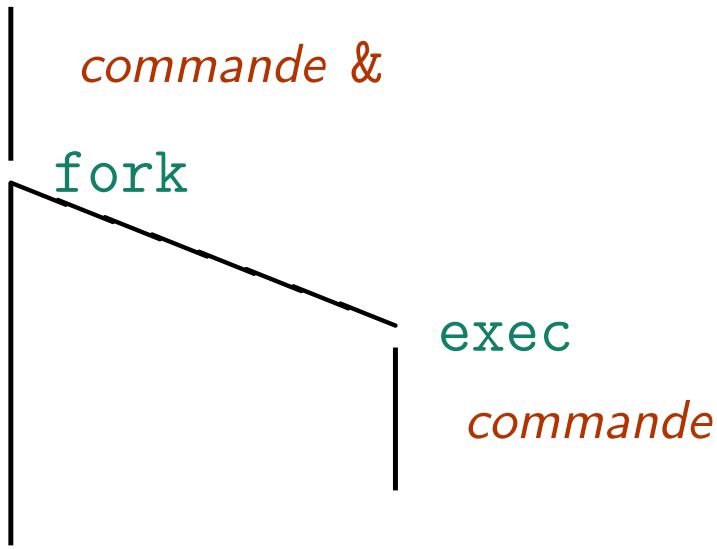
Dans *comm1* | *comm2*, | représente un tube (pipe) : les sorties de *comm1* sont les entrées de *comm2*.

Enchaînement parallèle



Tâches de fond

commande & lance la commande *commande* en tâche de fond : l'interpréteur n'attend pas la fin de l'exécution de la commande et permet de relancer immédiatement une autre commande. Le système affiche le numéro du processus créé.



Suspension et reprise

La suspension d'un processus qui s'exécute en avant-plan s'effectue en appuyant sur les touches `<CTRL>-z`.

La reprise d'une tâche suspendue peut se faire de deux manières :

- `fg` (foreground) relance la tâche dans le terminal ;
- `bg` (background) relance la tâche en tâche de fond.

L'arrêt complet du processus s'effectue en utilisant la commande `kill`.

Attention : lorsque l'on a appuyé sur `<CTRL>-z`, on a l'impression que le processus est *mort*. Ce n'est pas le cas, il est seulement *suspendu*.

Gestion des tâches

L'interpréteur de commandes (*shell*) maintient une table des processus en cours d'exécution, qu'ils soient alloués ou non au processeur et suspendus ou non.
Lorsqu'un processus est lancé en *tâche de fond*, une ligne est affichée :

```
[1] 25647
```

Cette ligne indique que c'est la tâche numéro 1, et que l'identificateur du processus est 25647.

Gestion des tâches

Lors de la terminaison d'une tâche de fond, un ligne est affichée dans le terminal :

```
[2]+  Done                    emacs
```

Ceci indique que la tâche numéro 2 s'est terminée, et que c'était `emacs`.

Contrôle des tâches

`jobs` affiche la table des tâches.

`%n` ou `fg %n` met la tâche numéro *n* en avant-plan.

`%n &` ou `bg %n` met la tâche numéro *n* en arrière-plan (tâche de fond).

Au lieu de préciser le numéro, on peut utiliser le nom de la commande ou le début de ce nom (s'il n'y a pas d'ambiguité).

Liste des processus

Deux commandes permettent d'accéder aux processus en cours d'exécution :

`ps`

affiche l'état des processus en cours : identificateur de processus (`PID`), terminal (`TTY`), temps, et commande.

Options de `ps` :

- `u` les processus avec le nom de l'utilisateur propriétaire
- `a` les processus des autres utilisateurs
- `x` tous les processus non attachés à un terminal

`top`

affiche l'état des processus en temps réel.

Les touches `h` (help) ou `?` permettent d'accéder à l'aide en ligne de la commande `top`.

Liste des processus

Deux commandes permettent d'accéder aux processus en cours d'exécution :

ps

affiche l'état des processus en cours : identificateur de processus (PID), terminal (TTY), temps, et commande.

Options de ps :

- u les processus avec le nom de l'utilisateur propriétaire
- a les processus des autres utilisateurs
- x tous les processus non attachés à un terminal

top

affiche l'état des processus en temps réel.

Les touches h (help) ou ? permettent d'accéder à l'aide en ligne de la commande top.

Signaux

Les signaux permettent d'avertir un processus qu'un événement important s'est produit. Le processus peut alors réagir à cet événement.

Principaux signaux :

`SIGINT` (2) (interrupt) est émis lorsque l'on tape `<CTRL>-c`.

`SIGQUIT` (3) (quit) est émis lorsque l'on tape `<CTRL>-\`.

`SIGKILL` (9) (kill) tue un processus, quel que soit son état.

`SIGALRM` (13) (alarm) est associé à une horloge.

`SIGTERM` (15) (terminate) est émis lorsqu'un processus termine normalement.

`SIGTSTP` (20) (terminal stop) est émis lorsque l'on tape `<CTRL>-z`.

Arrêt d'une tâche

L'arrêt d'une tâche se fait par envoi de *signaux*.

`kill -l`

affiche la liste des signaux connus.

`kill` *signal liste_de_pid*

envoie le *signal* à tous les processus de la *liste_de_pid*.

1 Historique et généralités

2 Systèmes de fichiers

3 Commandes UNIX

4 Entrées/sorties et processus

5 Environnement utilisateur et scripts shell
- Environnement utilisateur
- Variables
- Scripts shell

Exécution lors de la connection

Lorsque l'utilisateur se connecte, un processus *shell* est exécuté. De plus, des commandes, soit communes, soit propres à chacun, sont exécutées. Elles se trouvent dans un fichier du répertoire utilisateur :

- `.login`
- `.profile`
- `.cshrc`
- `.bashrc`

Environnement utilisateur

Les commandes du *shell* permettent d'initialiser des variables, soit pour leur donner une valeur différente de la valeur par défaut, soit pour les définir.

L'ensemble de ces variables constitue l'environnement *shell*.

Variables

Le nom d'une variable est une chaîne de caractères contenant des lettres, des chiffres ou le caractère _ et commençant toujours par une lettre.

La valeur d'une variable est une chaîne de caractères.

Variables d'environnement

Les principales variables d'environnement sont :

- `PS1` : premier *prompt*
- `PS2` : second *prompt*, utilisé pour continuer une commande pas terminée
- `HOME` : référence absolue du répertoire utilisateur
- `PATH` : liste des chemins dans lesquels les commandes appelées vont être cherchées
- `LOGNAME` : nom d'utilisateur
- `TERM` : type du terminal utilisé

Environnement et commandes

`printenv`

affiche la liste des variables d'environnement et leur valeur.

`which` *commande*

affiche le chemin d'accès à la *commande*.

`locate` *chaîne*

affiche tous les noms de fichiers (depuis la racine) contenant la chaîne de caractères *chaîne* et auxquels l'utilisateur a accès.

Environnement et commandes

`printenv`

affiche la liste des variables d'environnement et leur valeur.

`which` *commande*

affiche le chemin d'accès à la *commande*.

`locate` *chaîne*

affiche tous les noms de fichiers (depuis la racine) contenant la chaîne de caractères *chaîne* et auxquels l'utilisateur a accès.

Environnement et commandes

`printenv`

affiche la liste des variables d'environnement et leur valeur.

`which` *commande*

affiche le chemin d'accès à la *commande*.

`locate` *chaîne*

affiche tous les noms de fichiers (depuis la racine) contenant la chaîne de caractères *chaîne* et auxquels l'utilisateur a accès.

Utilisation de variables

Affectation : `var=val`

Valeur : `$var`

Si la variable `var` n'a pas été définie, son contenu est la chaîne de caractères vide.

Portée : la variable est seulement connue du processus *shell* dans laquelle elle a été affectée. Pour qu'elle soit transmise aux *sous-shells*, elle doit être exportée :

```
export var
```

Utilisation de variables

Affectation : `var=val`

Valeur : `$var`

Si la variable `var` n'a pas été définie, son contenu est la chaîne de caractères vide.

Portée : la variable est seulement connue du processus *shell* dans laquelle elle a été affectée. Pour qu'elle soit transmise aux *sous-shells*, elle doit être exportée :

```
export var
```

Utilisation de variables

Affectation : `var=val`

Valeur : `$var`

Si la variable `var` n'a pas été définie, son contenu est la chaîne de caractères vide.

Portée : la variable est seulement connue du processus *shell* dans laquelle elle a été affectée. Pour qu'elle soit transmise aux *sous-shells*, elle doit être exportée :

```
export var
```

Variables numériques

Une variable peut avoir une valeur numérique entière. Le *shell* peut alors évaluer des expressions arithmétiques.

```
var1=3
var2=$(($var1+4))
var3=$[$var1*5]
```

Délimiteurs

Les délimiteurs permettent d'effectuer des opérations à l'intérieur de chaînes de caractères.

'*chaîne*'

la *chaîne* de caractères entre apostrophes (quotes) est utilisée telle quelle. En particulier, s'il y a des appels à des variables, aucune substitution n'est effectuée.

"*chaîne*"

la substitution des variables contenues dans la *chaîne* de caractères entre guillemets est effectuée.

`*chaîne*`

la *chaîne* de caractères entre apostrophes inversées (backquotes) est considérée comme une commande *shell* et est exécutée.

Délimiteurs

Les délimiteurs permettent d'effectuer des opérations à l'intérieur de chaînes de caractères.

'*chaîne*'

la *chaîne* de caractères entre apostrophes (quotes) est utilisée telle quelle. En particulier, s'il y a des appels à des variables, aucune substitution n'est effectuée.

"*chaîne*"

la substitution des variables contenues dans la *chaîne* de caractères entre guillemets est effectuée.

`*chaîne*`

la *chaîne* de caractères entre apostrophes inversées (backquotes) est considérée comme une commande *shell* et est exécutée.

Délimiteurs

Les délimiteurs permettent d'effectuer des opérations à l'intérieur de chaînes de caractères.

'*chaîne*'

la *chaîne* de caractères entre apostrophes (quotes) est utilisée telle quelle. En particulier, s'il y a des appels à des variables, aucune substitution n'est effectuée.

"*chaîne*"

la substitution des variables contenues dans la *chaîne* de caractères entre guillemets est effectuée.

`*chaîne*`

la *chaîne* de caractères entre apostrophes inversées (backquotes) est considérée comme une commande *shell* et est exécutée.

Délimiteurs

Les délimiteurs permettent d'effectuer des opérations à l'intérieur de chaînes de caractères.

'*chaîne*'

la *chaîne* de caractères entre apostrophes (quotes) est utilisée telle quelle. En particulier, s'il y a des appels à des variables, aucune substitution n'est effectuée.

"*chaîne*"

la substitution des variables contenues dans la *chaîne* de caractères entre guillemets est effectuée.

`*chaîne*`

la *chaîne* de caractères entre apostrophes inversées (backquotes) est considérée comme une commande *shell* et est exécutée.

Affichage

`echo` *chaîne*

affiche la *chaîne* de caractères, avec éventuelle substitution des variables, suivant les délimiteurs utilisés.

`echo -n` *chaîne*

affiche la *chaîne* de caractères, sans retour à la ligne.

Alias

Les alias permettent de substituer une chaîne de caractères à un mot lorsqu'il est utilisé comme premier mot d'une commande simple.

Création d'un alias : `alias` *nom_alias = chaîne*

Suppression d'un alias : `unalias` *nom_alias*

Liste des alias définis : `alias`

Scripts *shell*

Un *script shell* est un fichier contenant une suite de commandes *shell*.

Un *script shell* permet de rassembler plusieurs commandes, par exemple, si l'on doit exécuter plusieurs fois une suite de commandes relativement longue. C'est alors une commande écrite par l'utilisateur.

Début d'un script *shell*

On peut indiquer, dans la première ligne du fichier, l'interpréteur shell à utiliser :

```
#!/bin/bash
```

Des commentaires peuvent être insérés dans le fichier, et ne sont pas interprétés par le *shell*. Les commentaires sont des chaînes de caractères commençant par #.

Début d'un script *shell*

On peut indiquer, dans la première ligne du fichier, l'interpréteur shell à utiliser :

```
#!/bin/bash
```

Des commentaires peuvent être insérés dans le fichier, et ne sont pas interprétés par le *shell*. Les commentaires sont des chaînes de caractères commençant par #.

Paramètres

Un *script shell* peut accepter des paramètres :
monscript param_1 . . . param$_n$

Les paramètres sont référencés comme des variables de nom 1, 2, . . . , 9.

```
echo "valeur du paramètre 2 = $2"
```

Autres variables

`$0` : nom de la commande appelée

`$*` : liste des paramètres

`$#` : nombre de paramètres

`$$` : numéro du processus *shell* correspondant à la commande

Exécution d'un *script shell*

Pour exécuter un *script shell*, il y a deux manières :

- `source` *monscript*
- Changer le mode de *monscript* pour qu'il devienne exécutable, puis l'exécuter (comme une commande).

Structures de contrôle

si . . . alors . . . sinon . . . finsi

```
if liste_commandes_1
then liste_commandes_2
else liste_commandes_3
fi
```

cas

```
case chaîne_caractères in
  motif_1 ) liste_commandes_1 ;;
     ⋮                ⋮
  motif_n ) liste_commandes_n ;;
  * ) liste_commandes_défaut ;;
esac
```

Structures de contrôle

si . . . alors . . . sinon . . . finsi

```
if liste_commandes_1
then liste_commandes_2
else liste_commandes_3
fi
```

cas

```
case chaîne_caractères in
   motif_1 ) liste_commandes_1 ;;
      ⋮                ⋮
   motif_n ) liste_commandes_n ;;
   * ) liste_commandes_défaut ;;
esac
```

Tests

Tests sur les chaînes de caractères

`[ chaîne1 = chaîne2 ]`

teste si les deux chaînes de caractères sont égales

`[ chaîne1 != chaîne2 ]`

teste si les deux chaînes de caractères sont différentes

`[ -n chaîne ]`

teste si la chaîne de caractères est non vide

`[ -z chaîne ]`

teste si la chaîne de caractères est vide

Tests

Tests sur les valeurs numériques

`[ nb1 -eq nb2 ]` : égalité (*equal*)

`[ nb1 -ne nb2 ]` : inégalité (*not equal*)

`[ nb1 -gt nb2 ]` : plus grand (*greater than*)

`[ nb1 -ge nb2 ]` : plus grand ou égal (*greater or equal*)

`[ nb1 -lt nb2 ]` : plus petit (*lower than*)

`[ nb1 -le nb2 ]` : plus petit ou égal (*lower or equal*)

Tests

Tests sur les fichiers

`[ -d` *fichier* `]` : teste si le fichier est un répertoire

`[ -f` *fichier* `]` : teste si *fichier* est un nom de fichier

`[ -r` *fichier* `]` : teste le droit de lecture sur le fichier

`[ -w` *fichier* `]` : teste le droit d'écriture sur le fichier

`[ -x` *fichier* `]` : teste le droit d'exécution sur le fichier

Boucles

pour . . . faire . . . finpour

```
for variable in liste_chaînes_caractères
do
   liste_commandes
done
```

tant que . . . faire . . . fintq

```
while liste_commandes_1
do
   liste_commandes_2
done
```

Boucles

pour . . . faire . . . finpour

```
for variable in liste_chaînes_caractères
do
   liste_commandes
done
```

tant que . . . faire . . . fintq

```
while liste_commandes_1
do
   liste_commandes_2
done
```

Autres instructions

`set` *chaîne*

la *chaîne* de caractères devient la nouvelle liste de paramètres

`read` *liste_variables*

les variables prennent les valeurs fournies par l'entrée standard

`exit` *entier*

le script termine et renvoie l'*entier* comme code de retour

Autres instructions

`set` *chaîne*

la *chaîne* de caractères devient la nouvelle liste de paramètres

`read` *liste_variables*

les variables prennent les valeurs fournies par l'entrée standard

`exit` *entier*

le script termine et renvoie l'*entier* comme code de retour

Autres instructions

`set` *chaîne*

la *chaîne* de caractères devient la nouvelle liste de paramètres

`read` *liste_variables*

les variables prennent les valeurs fournies par l'entrée standard

`exit` *entier*

le script termine et renvoie l'*entier* comme code de retour

Exemple 1

```
set `ls`
for i in $*
do
  if [ -d $i ]
  then echo "$i est un répertoire"
  fi
  if [ $i = "toto" ]
  then echo "Voulez-vous voir le contenu de toto?"
    read rep
    case $rep in
      o|O ) cat $i ;;
      n|N ) echo "Pas de visualisation de toto" ;;
      * ) echo "Réponse incorrecte" ;;
    esac
  fi
done
```

Exemple 2

```
chaine = $1
ps | grep chaine | grep -v grep |
  while read pid reste
  do
    kill -9 $pid
  done
```