Licence ASRALL

2021-2022

Principes et utilisation Linux

Étienne André

IUT Charlemagne, Université de Lorraine
Etienne.Andre@univ-lorraine.fr
www.loria.science/andre/enseignement/Linux/

Version: 6 octobre 2021

Objectifs & organisation du cours

Page web :

`www.loria.science/andre/enseignement/Linux/`

- Objectifs : Comprendre le fonctionnement d'un système d'exploitation.
- Utilisation (avancée) d'une machine Linux (Unix).
- Automatisation des tâches (scripts).
- Organisation : 7 cours-TP
- Évaluation : contrôle continu + un contrôle final

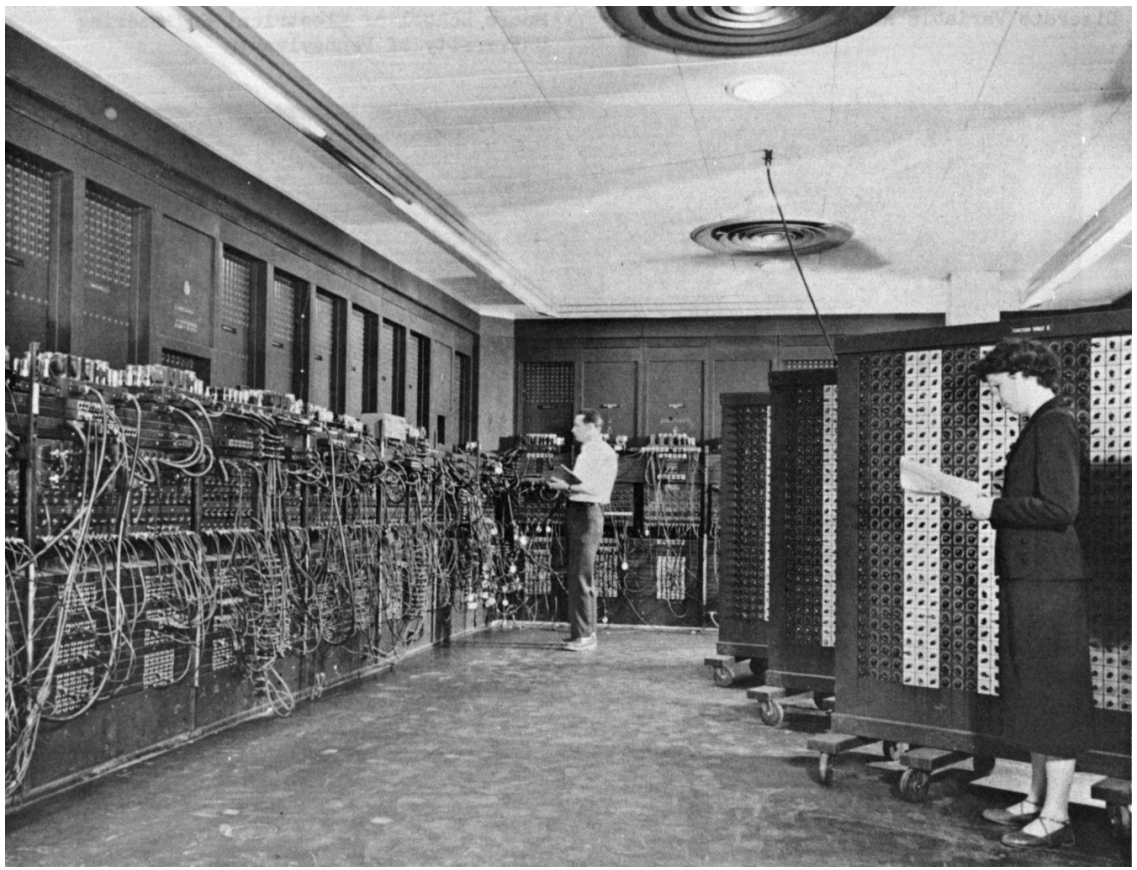
Plan du cours

Plan du cours

1re génération (1945-1955) : exploitation porte ouverte

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

ENIAC 1946



2e génération (1955-1965) : traitement par lots

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

UNIVAC 1954

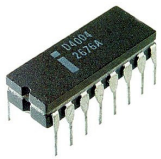
3e génération (1965-1980) : multi-programmation puis traitement partagé

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

PDP-7

4e génération (1980-présent) : réseaux et systèmes répartis

- 1969 : premier microprocesseur



- Réseaux/Cloud

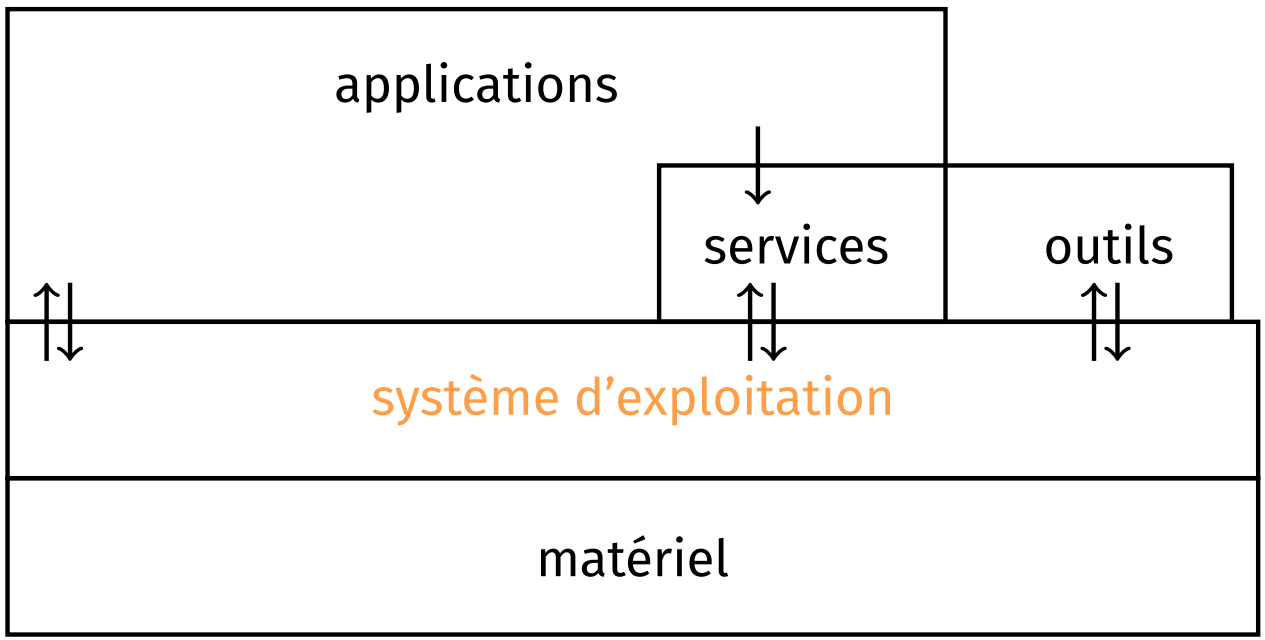
Système informatique : Matériels

- **Traitement de l'information** : ordinateurs, serveurs, terminaux mobiles, etc.
- **Entrée / Sortie** : Écrans, scanners, imprimantes, etc.
- **Stockage de l'information** : disques et mémoires de stockage et d'archivage.
- **Communication** : cartes réseaux, concentrateurs, commutateurs, routeurs, supports de transmission.

Système informatique : Logiciels

- Applications : Logiciels achetés ou développés pour résoudre des problèmes spécifiques : Jeux, éducation, publication, dessins, calcul scientifique, etc.
- Logiciels de base : Applications pour le traitement de problèmes usuels
  - **Outils** : Traitement de texte, Impression, Compilateurs, Interpréteurs, Outils réseaux (mail, navigateur Web, etc).
  - **Services** : Systèmes graphiques, communication réseaux (protocoles), *base de données*, etc.
  - **Système d'exploitation** : Gestion d'un ordinateur : lancement de programmes, gestion du processeur et des périphériques, etc.

Interactions entre les composantes



Plan du cours

Système d'exploitation

- Objectif : offrir une interface de gestion d'un ordinateur
- **Système d'exploitation = machine virtuelle**
- Cacher l'hétérogénéité des matériels (processeurs, disques, cartes réseaux, etc.)
- Plusieurs classes de systèmes :
  - Mono/Multi-utilisateurs
  - Mono/Multi-tâches
  - Monoprocesseur, Multiprocesseur, Réparti, parallèle
  - Temps réel, embarqués, ...
- Exemples : Microsoft Windows, Linux/Unix, macOS (Apple), iOS (Apple), Android, ...

Système d'exploitation : Concepts principaux

- **Utilisateur** (*user*) : réel ou logiciel, possède des droits et nécessite d'être authentifié lors de la connexion.
- **Fichier** (*file*) : une structure logique qui délimite une zone de stockage de données sur disque. Il est caractérisé par un type qui dépend de la nature des données qu'il contient et par des attributs (comme les droits, date de création, date de modification)
- **Processus** (*process*) : Un programme en cours d'exécution.

Plan du cours

Le système Unix/Linux : Historique

- Création de Unix aux laboratoires Bell (USA) en 1969.
- But : gestion d'un mini-ordinateur pour une petite équipe de programmeurs.
- Système développé en langage C.
- Intéresse rapidement les universités, ensuite les constructeurs.
- Linux : Version d'Unix pour micro-ordinateurs, 1991.
- Différentes distributions : Slackwre, Red Hat, Debian, SuSE, Ubuntu, Mint, etc.
- De nos jours :
  - respect de la norme POSIX $\Rightarrow$ compatibilité.
  - interface graphique $\Rightarrow$ convivialité.

Le système Unix/Linux : Caractéristiques

- Multi-plateforme
- Multitâches et multi-utilisateurs
- **Un système ouvert**
- Différents environnements de commandes : les shells
- Système de fichiers hiérarchique
- Gestion hiérarchisée des processus

Plan du cours

Les fichiers

Définition (fichier)

Un fichier est une suite de données sauvegardée sur un support de stockage permanent (disque dur, mémoire flash, CD, DVD).

Le codage utilisé détermine la nature du fichier : texte, image, son, vidéo, documents, programme exécutable, etc.

Les fichiers

Définition (fichier)

Un fichier est une suite de données sauvegardée sur un support de stockage permanent (disque dur, mémoire flash, CD, DVD).

Le codage utilisé détermine la nature du fichier : texte, image, son, vidéo, documents, programme exécutable, etc.

Règles de nommage :

- Nom d'un fichier : chaîne de caractères
  En pratique, limitez-vous aux caractères alphabétiques, aux chiffres, ainsi qu'aux caractères suivants : « . », « _ », « - »
  - Éviter : `: / \ ! ?` (espace)
- Souvent on utilise : **nom.extension**
  L'extension permet à l'utilisateur de se souvenir du type du fichier : `txt`, `pdf`, `gif`, `png`, `jpeg`, `avi`, `mpeg`
- Linux fait la différence entre les caractères majuscules et les minuscules.
  Exemple : `Fichier.txt` $\neq$ `fichier.txt`

Les répertoires

Définition (répertoire)

Les répertoires (ou dossiers) sont des fichiers particuliers qui peuvent contenir d'autres fichiers.

Les répertoires

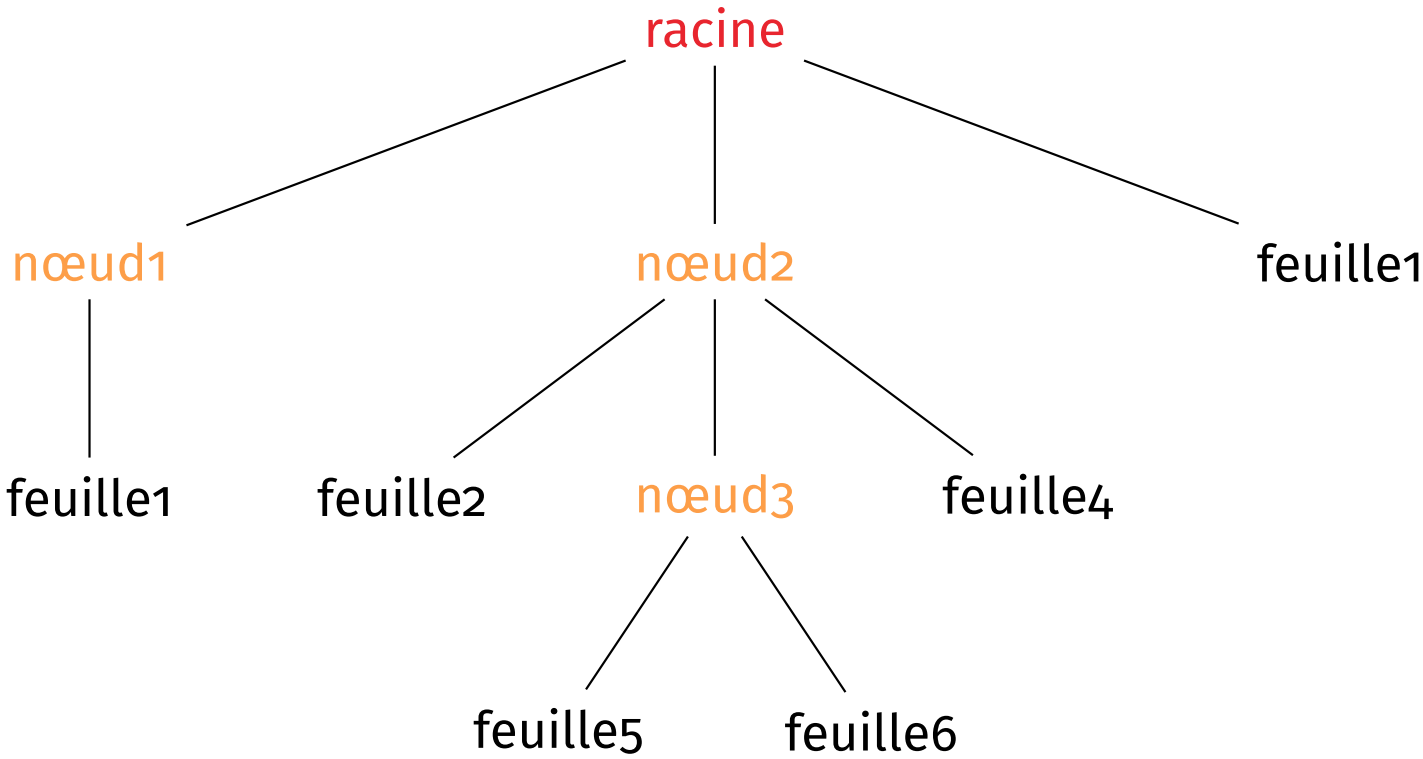
Définition (répertoire)

Les répertoires (ou dossiers) sont des fichiers particuliers qui peuvent contenir d'autres fichiers.

- Les règles de nommage vues pour les fichiers restent valides.
- Dans un même répertoire, deux fichiers différents ne peuvent pas avoir le même nom.
- Un répertoire peut contenir des fichiers et des sous-répertoires $\rightarrow$ Système de fichiers sous forme d'arbre.

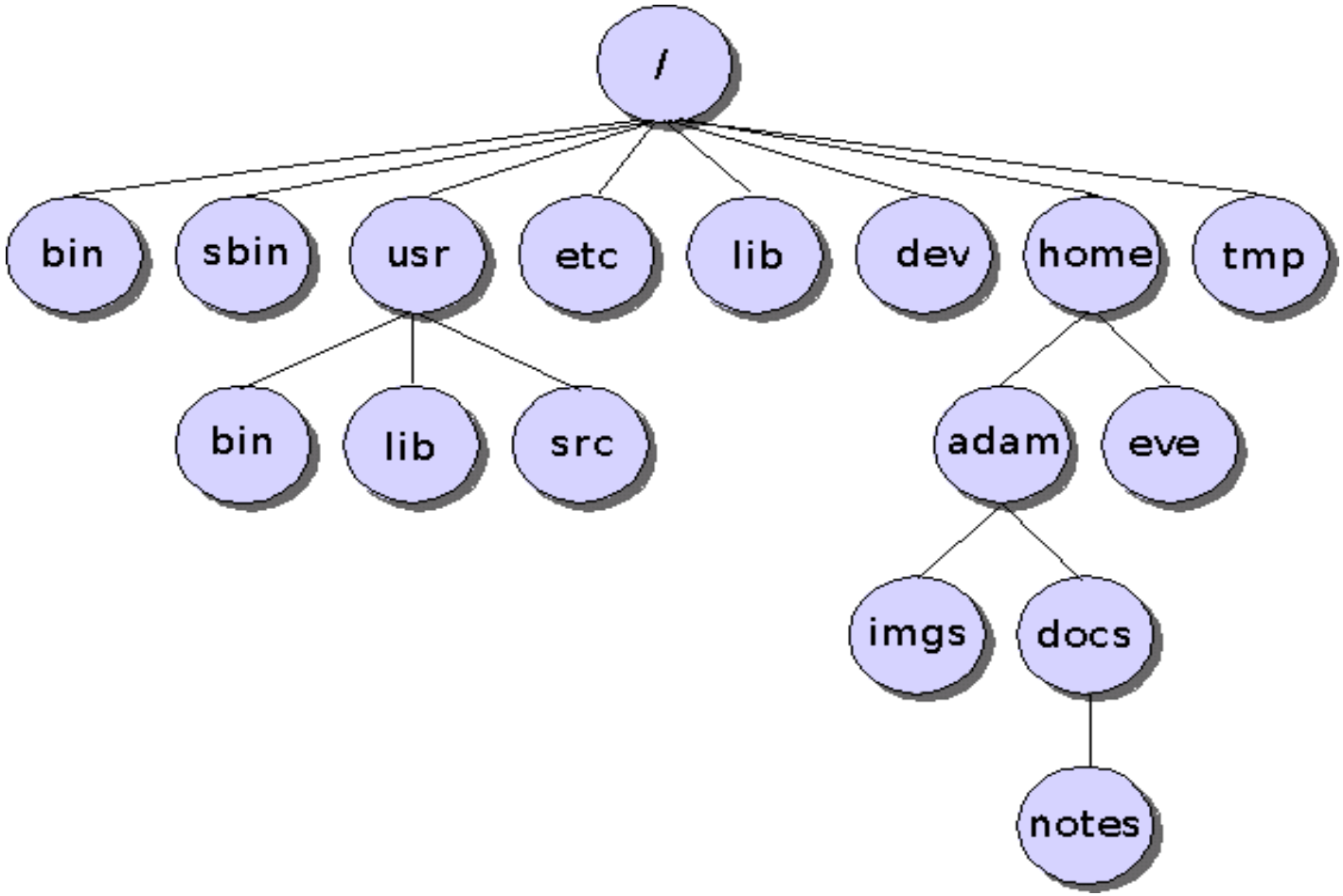
Plan du cours

Structure arborescente du système de fichiers



racine = répertoire racine nœuds = répertoires feuilles = fichiers

Structure arborescente du système de fichiers



Répertoires

Répertoire racine désigné par / est le seul répertoire qui n'a pas de répertoire parent

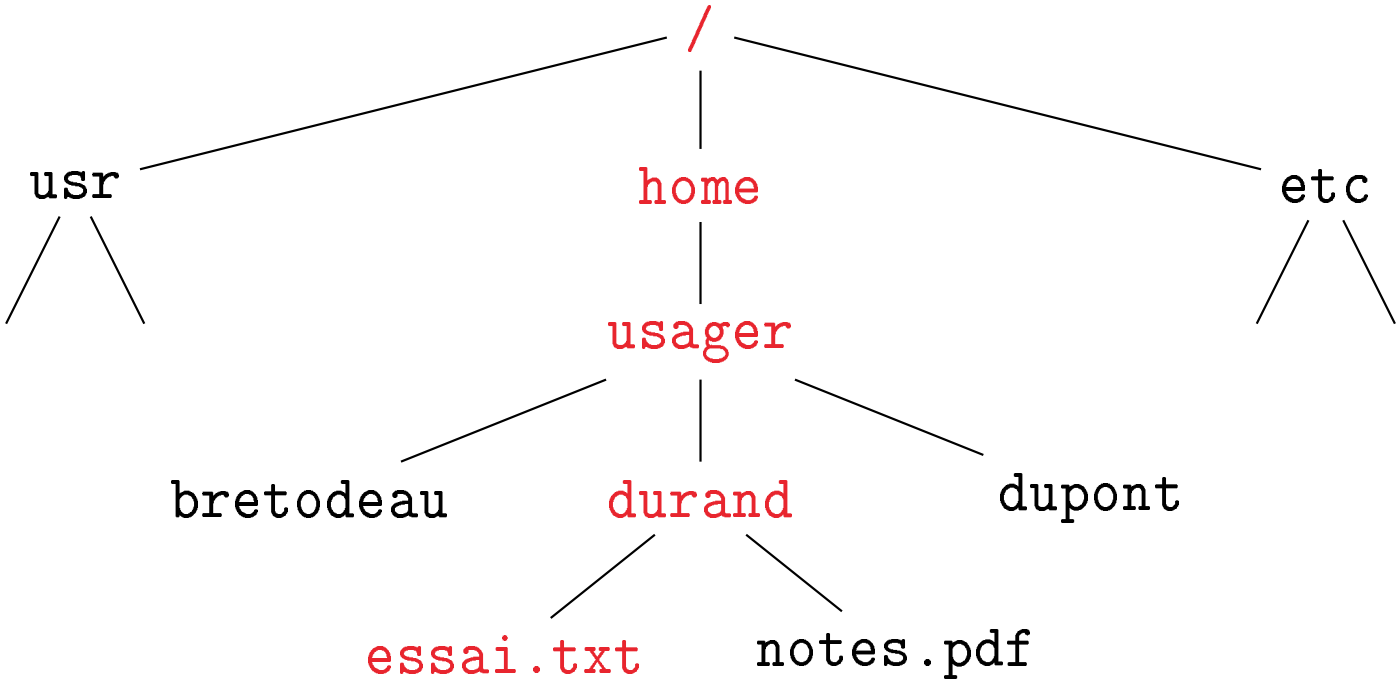
Répertoire personnel désigné par ∼
est le répertoire contenant les fichiers d'un utilisateur.
Exemple : pour l'utilisateur `eve` le répertoire personnel est `/home/eve`

Répertoire courant/de travail désigné par .

Répertoire parent Le répertoire qui contient le répertoire courant est désigné par ..

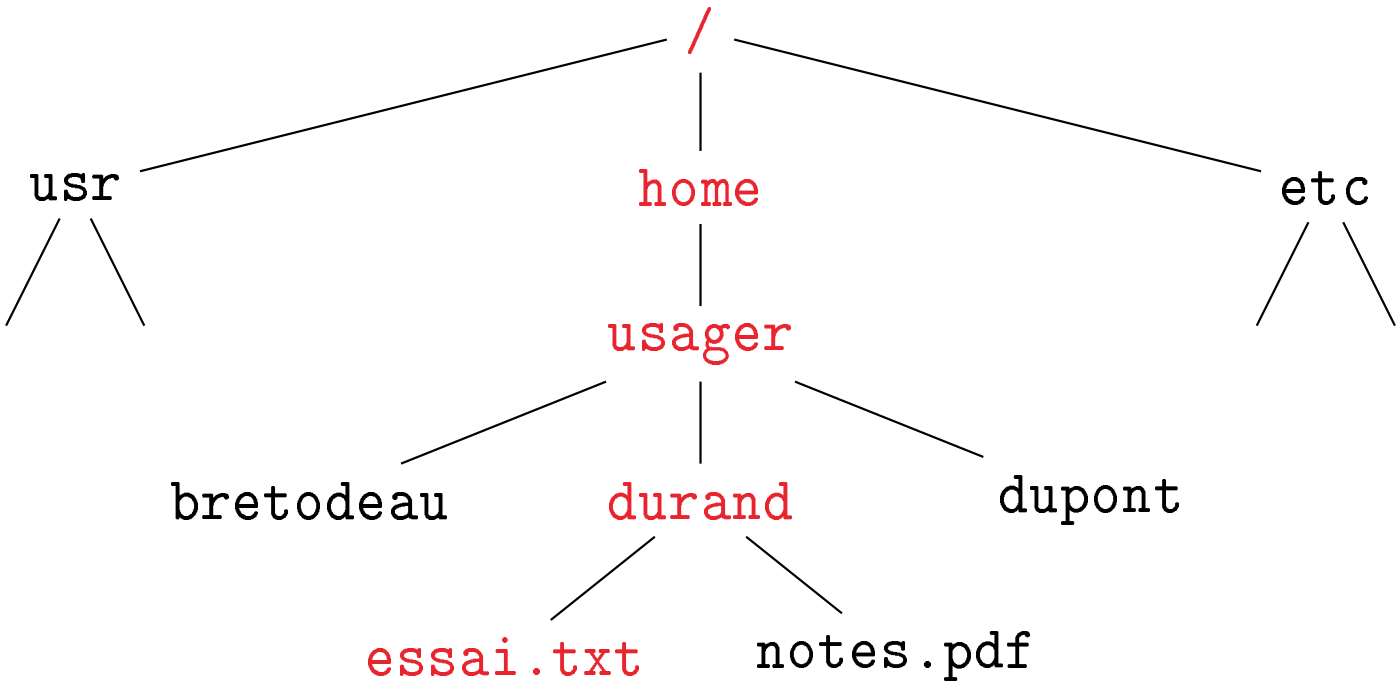
Fichiers et répertoires : Référence absolue

Chaque fichier ou répertoire est désigné d'une **manière unique** par le chemin depuis la racine. On appelle ce chemin référence absolue.



Fichiers et répertoires : Référence absolue

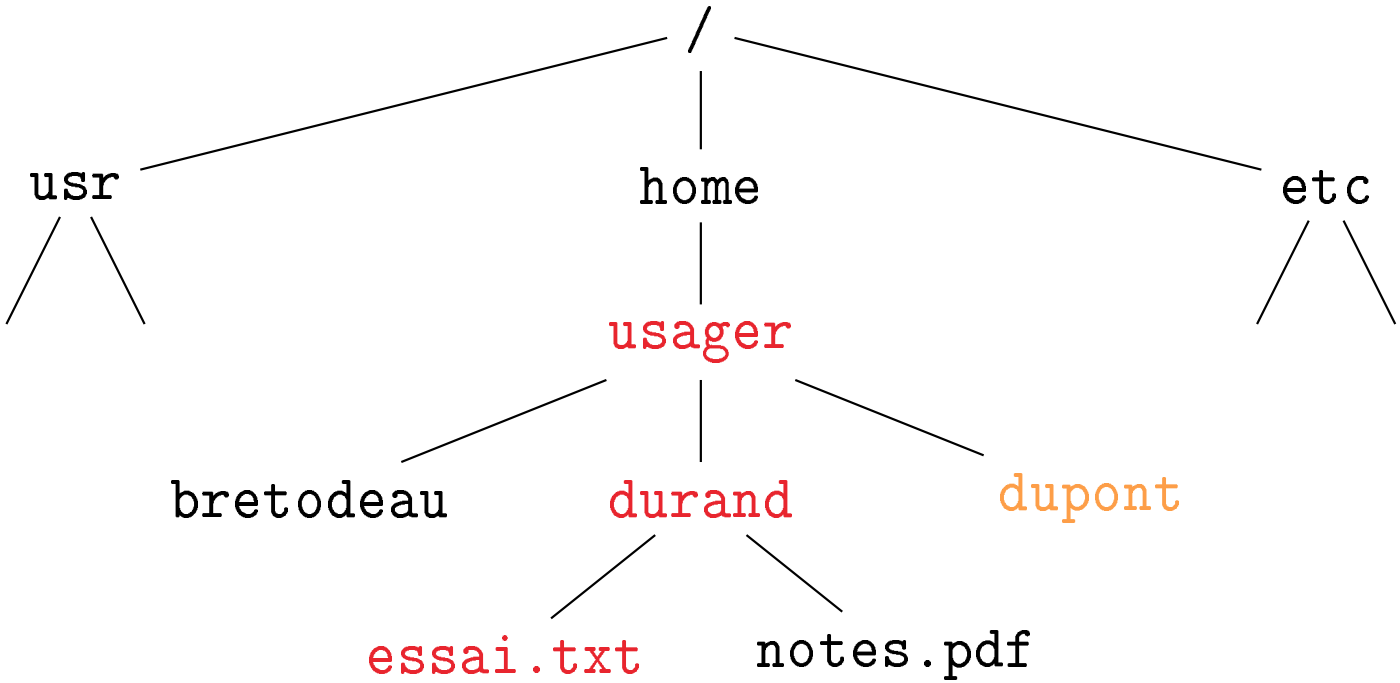
Chaque fichier ou répertoire est désigné d'une **manière unique** par le chemin depuis la racine. On appelle ce chemin référence absolue.



`/home/usager/durand/essai.txt`

Fichiers et répertoires : Référence relative

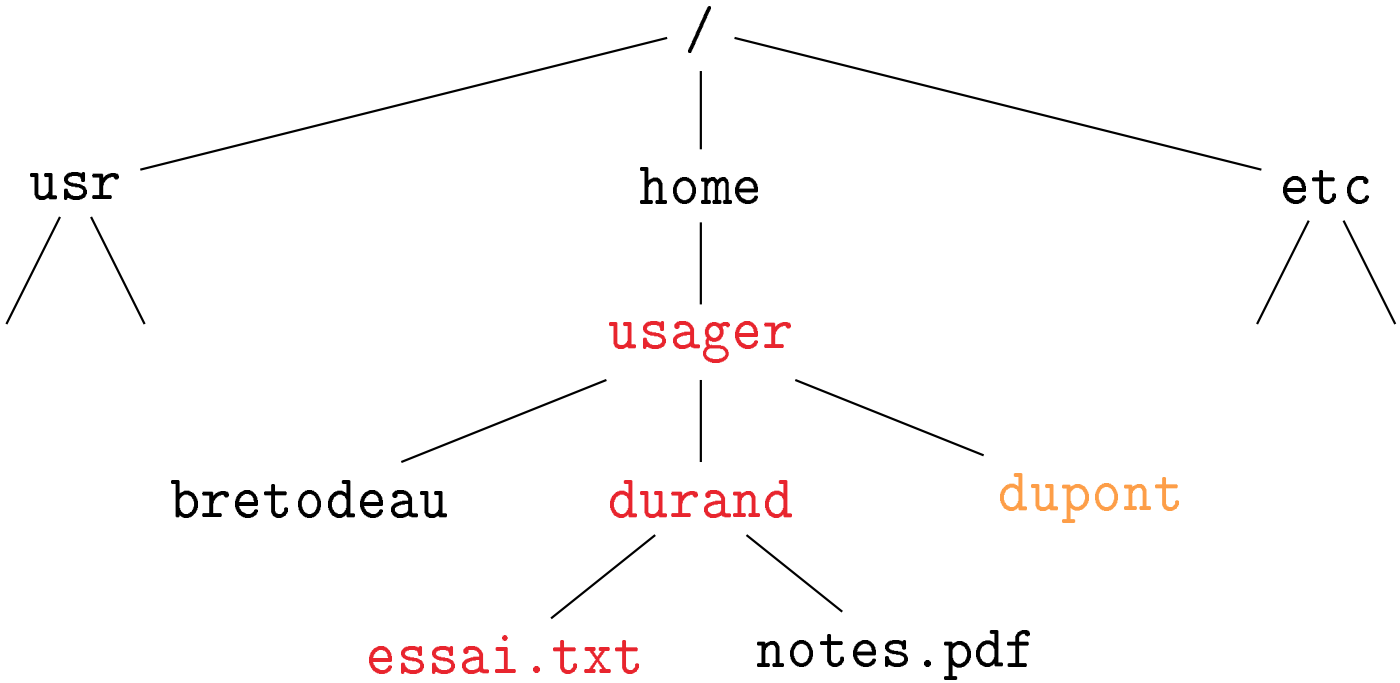
On peut désigner un fichier par une référence relative au répertoire courant.



À partir du répertoire personnel de `dupont`, le fichier `essai.txt` peut être désigné par la référence rélative :

Fichiers et répertoires : Référence relative

On peut désigner un fichier par une référence relative au répertoire courant.



À partir du répertoire personnel de `dupont`, le fichier `essai.txt` peut être désigné par la référence rélative :

```
../usager/durand/essai.txt
```

Plan du cours

Commandes de navigation

- `pwd` : (*Print Working Directory*) affiche le nom du répertoire courant

- `cd` : (*Change Directory*)
  `cd nom_répertoire` Le répertoire de travail devient le répertoire `nom_répertoire`
  `cd ~nom_utilisateur` Le répertoire de travail devient le répertoire personnel de l'utilisateur *nom_utilisateur*
  `cd ~` ou simplement `cd` Le répertoire de travail devient le répertoire personnel de l'utilisateur qui exécute la commande
  `cd ..` Aller vers le répertoire parent du répertoire courant.

Commandes d'affichage

- `ls` : (*list*)
  `ls` affiche le contenu du répertoire courant
  `ls nom_répertoire` affiche le contenu du répertoire désigné (s'il est accessible).
  `ls -l nom_répertoire` affiche le contenu de `nom_répertoire` avec des informations détaillées :
  - nom des fichiers/sous-répertoires,
  - tailles,
  - droits,
  - propriétaires,
  - date de dernière modification.

Création, copie, renommage/déplacement et suppression

- `touch` :
  touch fichier : si le fichier existe, mettre à jour la date de dernière modification. Sinon créer un fichier vide avec le nom *fichier*.
- `cp` : (*copy*)
  `cp ancien nouveau` : copier le fichier *ancien* dans *nouveau*
  `cp -R AncienRép NouveauRép` : copier d'une manière récursive le répertorie *AncienRép* dans *NouveauRép*
- `mv` : (*move*)
  `mv ancien nouveau` : déplacer le fichier *ancien* dans *nouveau*, en le renommant si *nouveau* désigne un fichier
- `rm` : (*remove*)
  `rm fichier` : effacer le fichier *fichier*
  `rm -r repertoire` : effacer *repertoire*, et récursivement tout son contenu.

Création et suppression de répertoire

- `mkdir` : (*make directory*)
  `mkdir répertoire` : créer le répertoire *répertoire*
- `rmdir` : (*remove directory*)
  `rmdir répertoire` : effacer le répertoire vide *répertoire*

Méta-caractères du `shell`

- `*` : toute chaîne de caractères ne commençant pas par .
- `?` : un caractère quelconque
- `[abc]` : un caractère quelconque parmi `a`, `b` ou `c`
- `[a-d]` : un caractère quelconque dans la plage allant du caractère `a` au caractère `d`
- `{mot1,mot2,mot3}` : prend comme valeur chaque élément de l'ensemble, donc d'abord `mot1`, ensuite `mot2` et finalement `mot3`.
- `{1..4}` : spécifie l'ensemble d'éléments qui vont de `1` à `4`.

Méta-caractères : exemples d'utilisation

```
ls -l *.txt
```

affiche seulement les fichiers ayant l'extension `txt`

```
rm td[1-3].pdf
```

efface les fichiers `td1.pdf`, `td2.pdf` et `td3.pdf`

```
touch td-M11{05,21}.pdf
```

met à jour la date (ou crée) les fichiers `td-M1105.pdf` et `td-M1121.pdf`

```
cp /tmp/?2022.data .
```

copie dans le répertoire courant tous les fichiers qui sont dans le répertoire `/tmp` dont le nom est composé d'une chaîne de cinq caractères qui se termine par 2022, et qui ont l'extension `data`

Plan du cours

Chercher un fichier/répertoire : la commande `find`

- `find [répertoire] [critère] [action]`
  chercher dans la sous-arborescence de `[répertoire]` les fichiers satisfaisant `[critère]` et exécuter `[action]` sur chaque fichier retrouvé.
- Exemple :

```
find ~ -name "*.pdf" -print
```

  Afficher tous les fichiers ayant l'extension `pdf` qui se trouvent dans la sous-arborescence enracinée dans le répertoire personnel de l'utilisateur.

La commande `find` : actions

- `-print` (utilisé par défaut) afficher le résultat de la recherche sous la forme d'une liste de noms.
- `-exec cmd {} \;` exécuter la commande `cmd` sur chaque résultat retrouvé. Les accolades {} sont remplacées par la liste des fichiers trouvés.
- `-ok cmd {} \;` même effet que `exec` mais en demandant la confirmation de l'utilisateur pour chaque exécution de `cmd`
- Exemple :

```
find ~ -name "*.pdf" -exec ls -l  {} \;
```

affiche les informations détaillées sur tout fichier ayant l'extension `.pdf` dans le répertoire personnel.

La commande `find` : Critères

- `-name nom` : rechercher selon le nom de l'objet.
- `-size [+|-]n[c|k|b]` : rechercher selon la taille de l'objet. + : plus grand, - : plus petit, rien : exactement, c : octet, k : kibi octet, b : bloc de 512 o
- `-mtime [+|-]n` : rechercher selon la date de modification de l'objet. + : plus grand, - : plus petit, rien : exactement, n = nombre de jours
- `-atime [+|-]n` : rechercher selon la dernière date d'accès à l'objet.
- `-ctime [+|-]n` : rechercher selon la date de création de l'objet.
- `-type [f|d|l]` : rechercher selon le type de l'objet (f : fichier, d : répertoire et l : lien)
- `-newer fichier` : rechercher les fichiers plus récents que le fichier donné.

La commande `find` : Critères

- `-perm droit` : rechercher selon les droits associés à l'objet.
- `-user uid` : rechercher selon l'identité du propriétaire.
- `-group gid` : rechercher selon l'identité du groupe.
- `-nouser` : rechercher les objets sans utilisateur
- `-nogroup` : rechercher les objets sans groupe.

La commande `find` : Critères composés

- `-a` : ET logique (*and*)
- `-o` : OU logique (*or*)
- `!` : NON logique

- `\( ... \)` : parenthèses

- Exemples :
  - ```
    find /tmp -size 0 -a -user dupont -print
    ```

    Afficher les noms des fichiers dans l'arborescence `/tmp`, de taille 0 octet appartenant à l'utilisateur `dupont`.
  - ```
    find / -user dupont -o -user martin -print
    ```

    Afficher les fichiers appartenant à l'utilisateur `dupont` ou à l'utilisateur `martin`

La commande `find` : Critères composés

- `-a` : ET logique (*and*)
- `-o` : OU logique (*or*)
- `!` : NON logique

- `\( ... \)` : parenthèses

- Exemples :
  - ```
    find /tmp -size 0 -a -user dupont -print
    ```

    Afficher les noms des fichiers dans l'arborescence `/tmp`, de taille 0 octet appartenant à l'utilisateur `dupont`.
  - ```
    find / -user dupont -o -user martin -print
    ```

    Afficher les fichiers appartenant à l'utilisateur `dupont` ou à l'utilisateur `martin`

Quel est l'effet de la commande suivante ?

```
find ~ \(-user dupont -o -user martin\) -a -size 0
    -print
```

Plan du cours

Utilisateurs du système

Tout utilisateur du système

- possède un numéro d'utilisateur : `uid` (*user identifier*)
- appartient à au moins un groupe. Chaque groupe est à son tour désigné par un identificateur `gid` (*group identifier*)

- La commande `id user_login` donne l'uid de l'utilisateur et la liste des groupes auxquels il appartient.
- Chaque ressource (ex. fichier) a un seul propriétaire.

Utilisateurs du système

Tout utilisateur du système

- possède un numéro d'utilisateur : `uid` (*user identifier*)
- appartient à au moins un groupe. Chaque groupe est à son tour désigné par un identificateur `gid` (*group identifier*)

- La commande `id user_login` donne l'uid de l'utilisateur et la liste des groupes auxquels il appartient.
- Chaque ressource (ex. fichier) a un seul propriétaire.

Pour changer d'utilisateur :

- `su` *nom_utilisateur* on devient *nom_utilisateur*
- `su` on devient administrateur du système (`root`)

Droits d'accès aux fichiers

On spécifie pour chaque ressource 3 groupes de droits d'accès :

- Les droits du propriétaire : user
- Les droits des utilisateurs appartenant au même groupe que le propriétaire : group
- Les autres : others

Droits d'accès aux fichiers

On spécifie pour chaque ressource 3 groupes de droits d'accès :

- Les droits du propriétaire : user
- Les droits des utilisateurs appartenant au même groupe que le propriétaire : group
- Les autres : others

Trois types de droits d'accès :

- Droit de lecture (*Read*) désigné par : r
- Droit d'écriture (*Write*) désigné par : w
- Droit d'exécution (*eXecute*) désigné par : x

Le droit d'exécution sur un répertoire donne l'autorisation de le traverser.

Codage des droits d'accès

- Les droits d'accès sont codés alors sur 9 bits :

user			group			others		
r	w	x	r	w	x	r	w	x
400	200	100	40	20	10	4	2	1

- Expression des droits en code octal :
  - `rw-r--` $\rightarrow$ 640
  - `rwxrwxr-` $\rightarrow$ 774

Lorsqu'un nouveau fichier/répertoire est crée,
quels seront ses droits d'accès ?

Manipulation des droits d'accès par défaut

Les droits par défaut sur les ressources crées par un utilisateur peuvent être fixés par la commande umask

- `umask code_octal`

Manipulation des droits d'accès par défaut

Les droits par défaut sur les ressources crées par un utilisateur peuvent être fixés par la commande umask

- `umask code_octal`

Lorsqu'un nouveau fichier est crée, ses droits sont calculés comme suit :

	$(666)_8$	ET (bit-à-bit)
NOT	$(\text{code\_octal})_8$	
	droits fichier	

Manipulation des droits d'accès par défaut

Les droits par défaut sur les ressources crées par un utilisateur peuvent être fixés par la commande umask

- `umask code_octal`

Lorsqu'un nouveau fichier est crée, ses droits sont calculés comme suit :

	$(666)_8$	ET (bit-à-bit)
NOT	$(\text{code\_octal})_8$	
	droits fichier	

Lorsqu'un nouveau répertoire est crée, ses droits sont calculés comme suit :

	$(777)_8$	ET (bit-à-bit)
NOT	$(\text{code\_octal})_8$	
	droits répertoire	

Exemple (défaut) : umask 022 donne aux fichiers les droits 644

- `umask` montre la valeur courante.

Manipulation de droits d'accès

- `chmod droits référence` : changer les droits d'accès de *référence*.
- Les droits peuvent être exprimés de deux manières :
  - Code octal : chmod 640 exemple.txt
  - Code symbolique : `cible opérateur droit`
    - Cible : `u` (user), `g` (group), `o` (others)
    - Opérateur : + (ajouter) , - (enlever), = (attribuer seulement ces droits)
    - Droit : `r` (read), `w` (write), `x` (execute)
- Exemples :
  - chmod ug+rw,u+x toto : le propriétaire et le groupe ont le droit de lire et modifier, le propriétaire a le droit d'exécuter. Attention les autres droits restent inchangés.
  - chmod o=g,o-x toto : Les autres ont les mêmes droits que le groupe mais sans le droit d'exécution.

chmod ugo+r,ug+w,go-x,u+x toto : que fait cette commande ?

Changement de propriétaire/groupe

- La commande `chown` (*change owner*)
  - chown user fichier : Le propriétaire de *fichier* devient *user*.
  - chown -R user répertoire : appliquer récursivement le changement sur toutes les ressources dans *répertoire*.
  - chown user.group répertoire : changer le groupe aussi.

- La commande `chgrp` (*change group*)
  - chgrp group fichier : Le groupe de *fichier* devient *group*.

Types de fichiers

- fichiers ordinaires : programmes, données. Un fichier est décrit par un nœud d'index ou i-node (*index node*).
- répertoires : ensemble de fichiers. Le contenu d'un répertoire est un ensemble de couples (nom fichier, nœud d'index).
- fichiers spéciaux : spécifient les périphériques. Ces fichiers sont vus par l'utilisateur comme des fichiers ordinaires.

Structure d'un i-node

i-node = descripteur de fichier

- i-number numéro d'inode
- taille 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

Pour le voir la liste des fichiers avec leur i-number : `ls -i`

Liens

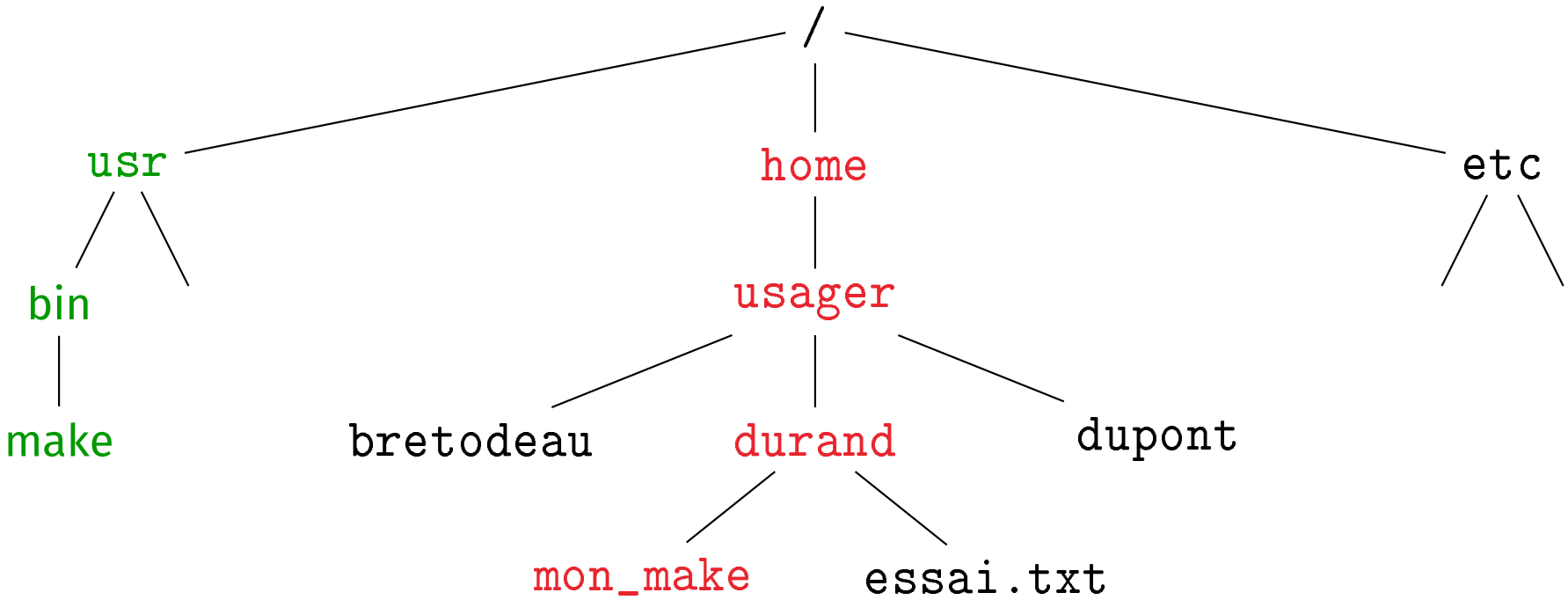
Définition (Lien)

Un lien est une référence supplémentaire qui peut désigner une ressource existante (ex. un fichier) à partir de n'importe quel nœud de l'arborescence du système de fichier.

Liens

Définition (Lien)

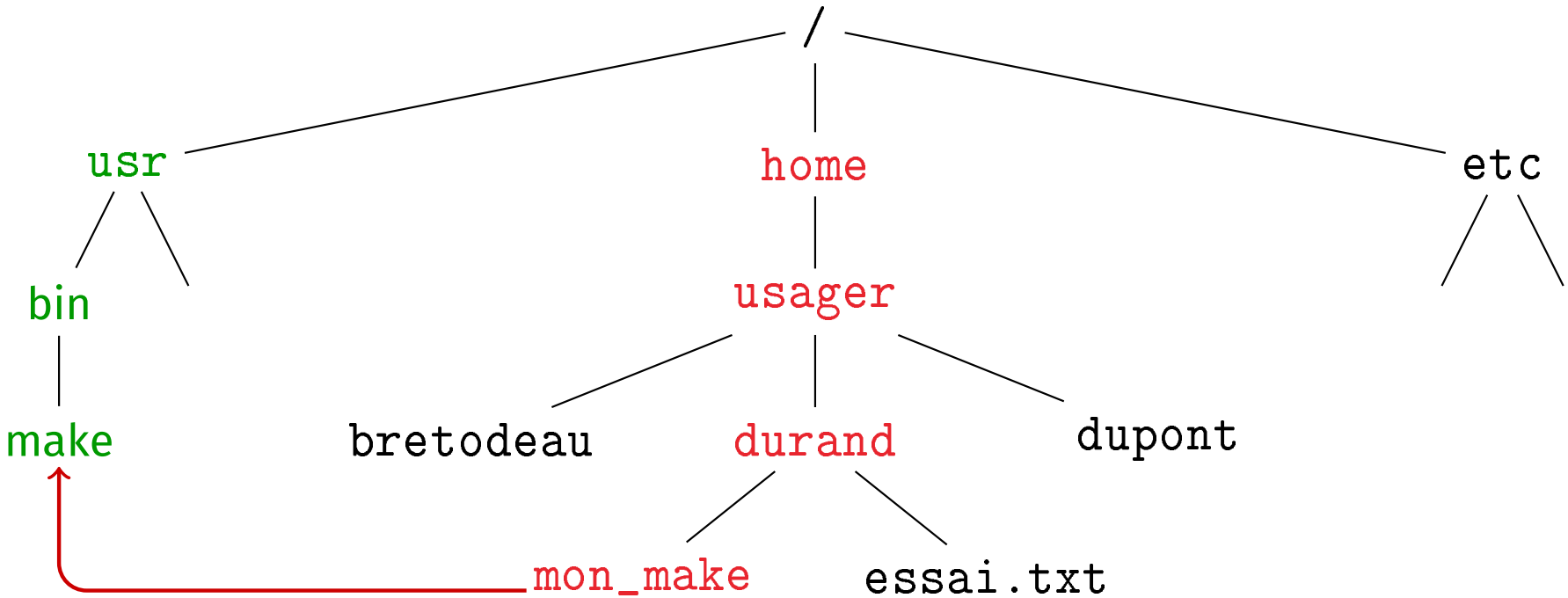
Un lien est une référence supplémentaire qui peut désigner une ressource existante (ex. un fichier) à partir de n'importe quel nœud de l'arborescence du système de fichier.



Liens

Définition (Lien)

Un lien est une référence supplémentaire qui peut désigner une ressource existante (ex. un fichier) à partir de n'importe quel nœud de l'arborescence du système de fichier.



Liens

Un lien est une sorte de deuxième étiquette qui implante un raccourci dans l'arborescence.

- Chaque ressource compte le nombre de références qui la désigne : compteur de références.
- Le compteur de références est affiché par la commande `ls -l`.
- Exemple :

```
-rw-r--r-- 2  eandre  staff  0 21 sep 04:55 ex
```

- Une ressource est complètement supprimée si le compteur de références est égal à 0.

Liens : commandes

- Deux types de liens :
  - Physique (même i-node)
  - Symbolique (i-node différent)
- `ln ressource nom_lien_physique`
- `ln -s ressource nom_lien_symbolique`
- Exemple :

```
> touch exemple.txt
> ln exemple.txt exemple.physique
> ln -s  exemple.txt exemple.symbolique
> ls -l exemple.*
-rw-r--r--  2 eandre  staff    0 21 sep 05:06
    exemple.phy
lrwxr-xr-x  1 eandre  staff   11 21 sep 05:07
    exemple.sym -> exemple.txt
-rw-r--r--  2 eandre  staff    0 21 sep 05:06
    exemple.txt
```

La commande `file`

- `file référence` : permet de se renseigner sur le type de référence donné et la nature des données contenues dans le fichier désigné.
- Types de références :
  - Répertoire (*directory*)
  - Fichier (*file*)
  - Lien (*link*)
- Nature de données : texte, pdf, image (png, gif, ?), etc.

Plan du cours

Plan du cours

Manuel en ligne

- ```
  man macommande
  ```

  affiche page à page le chapitre du manuel sur la commande `macommande`.

- ```
  man -k sujet
  ```

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

Voir aussi l'option `-help`. Exemple :

```
ls --help
```

Manuel en ligne

- ```
  man macommande
  ```

  affiche page à page le chapitre du manuel sur la commande `macommande`.

- ```
  man -k sujet
  ```

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

Voir aussi l'option `-help`. Exemple :

```
ls --help
```

Attention !

Il est souvent plus facile de chercher le fonctionnement d'une commande sur Internet.

Plan du cours

Manipulation de fichiers textes

- **echo** `chaîne` : afficher `chaîne` sur la sortie standard (l'écran).
  - `echo hello world` : affiche `hello world`.
- **more** `fichier` : afficher le contenu de `fichier` page par page. Le passage à la page suivante se fait en appuyant sur `espace`.
- **cat** `fichier` : afficher le contenu de `fichier` sur la sortie standard.
- **wc** `[options] fichier` : afficher le nombre de caractères (`-c`), de mots (`-w`) ou de lignes (`-l`) dans `fichier`
  - `wc -l toto` : affiche le nombre de lignes dans `toto`.
- **head -n** `fichier` : afficher les n premières lignes de `fichier`.
- **tail -n** `fichier` : afficher les n dernières lignes de `fichier`.
  - `tail -2 toto` : affiche les 2 dernières lignes du fichier `toto`

Les filtres textes

- **sort** `fichier` : trier les lignes de `fichier`.
- **diff** `fich1 fich2` : afficher les lignes de `fich1` et `fich2` qui sont différentes.
- **uniq** `fichier` : remplacer des lignes consécutives identiques par une seule ligne (supprimer la duplication).
- **tr** `str1 str2` : remplacer chaque occurrence de `str1` par `str2`.
- **cut** `options fichier` : découper chaque ligne de `fichier` en liste de valeurs et renvoyer un des champs selon les options.
  - `cut -d: -f2 toto` : Les lignes du fichier `toto` seront découpées en champs (*field*) de valeurs séparés par `:` (`-d` définit le séparateur). On affiche pour chaque ligne ainsi découpée le deuxième champ (option `-f2`).

Filtre : La commande `grep`

- **grep** `motif fichier` : afficher les lignes dans `fichier` qui contiennent `motif`.
- Quelques options (voir `man grep` pour une liste complète) :
  1. `-`*num* : afficher `num` lignes avant et après la ligne où figure le motif.
  2. `-B` *num* : afficher `num` lignes avant la ligne où figure le motif.
  3. `-A` *num* : afficher `num` lignes après la ligne où figure le motif.
  4. `-n` : afficher le numéro de ligne.
  5. `-c` : afficher **le nombre** d'occurrence du motif dans le fichier.
  6. `-i` : insensible à la casse (ignorer les différences entre minuscules et majuscules).
  7. `-v` : chercher les lignes qui **ne contiennent pas** le motif.
- Variations sur le motif :
  1. `grep '^[aA]' fich` : chercher les lignes qui commencent par `a` ou `A` dans le fichier `fich`.
  2. `grep '[aA]$' fich` : chercher les lignes qui se terminent par `a` ou `A` dans le fichier `fich`.

Plan du cours

Outils réseaux : les adresses

La communication entre machines nécessite l'utilisation d'interfaces réseaux.

- Exemple : cartes réseaux Ethernet, port sériale, USB, etc.
- Une interface réseau a une adresse physique. Dans le cas d'une carte Ethernet on parle d'adresse MAC.
- Une adresse MAC est une suite de 6 octets qui identifie d'une manière unique une carte. Convention : notée en hexadécimal.
- Une interface réseau relié à l'Internet doit avoir une adresse logique dite adresse IP.
- Une adresse IP (version 4) est une suite de 32 bits notée sous forme de 4 entiers (en décimal) séparés par « . »
- `10.0.0.1` est l'adresse `00001010 00000000 00000000 00000001` (voir le cours de réseaux !)

Outils réseaux : les adresses

- Les machines connectées à l'Internet peuvent avoir aussi des adresses symboliques plus faciles à manipuler
- Exemple : `www.nancy.fr` correspond à l'adresse `87.252.12.64`
- Des systèmes de mise en correspondance entre adresses symboliques et adresses IP (et *vice versa*) sont en place.
  - Serveur de nom de domaine (DNS)

Les commandes `hostname` et `ifconfig`

- `hostname` : afficher le nom symbolique de la machine.
  - Exemple :

```
hostname toto
```

    Attribue le nom symbolique `toto` à la machine.

- `ifconfig` : outil de configuration des interface réseaux (pour l'administrateur). Ça permet aussi de se renseigner sur les caractéristiques d'une interface réseau donnée de la machine.
  Exemples :
  -

```
ifconfig eth0
```

    affiche les information de configuration de l'interface `eth0` (première carte réseau Ethernet de la machine).
  -

```
ifconfig eth0 10.0.0.1
```

    attribue l'adresse `10.0.0.1` à l'interface `eth0`.

La commande nslookup

- `nslookup adrsym` : chercher l'adresse IP correspondant à l'adresse symbolique donnée.
  - Exemple :

```
nslookup nancy.fr
Server:    127.0.0.53
Address:   127.0.0.53#53

Non-authoritative answer:
Name: nancy.fr
Address: 87.252.12.64
```

La commande traceroute

- `traceroute adr` : donner la route entre la machine locale et la machine désignée par adr si la route existe.

```
> traceroute framasoft.org
traceroute to framasoft.org (144.76.131.212), 30 hops max, 60
    byte packets
 1  gw-289.iutnc.site.univ-lorraine.fr (100.68.23.254)  0.582
    ms * *
 2  * * *
 3  * gw1.lothaire.gw.lothaire.net (193.50.80.74)  0.387 ms *
 4  * * *
 5  fw-dcncy-nat-ul.vdom.fw.lothaire.net (193.49.140.9)
    0.493 ms  0.450 ms *
 6  193.49.140.13 (193.49.140.13)  0.619 ms *  0.628 ms
 7  * * *
 8  * * gw1.lothaire.gw.lothaire.net (193.50.80.74)  0.685 ms
 9  * * vl50-be1-ren-nr-nancy-rtr-091.noc.renater.fr
    (193.51.181.110)  1.231 ms
10  xe1-1-11-paris1-rtr-131.noc.renater.fr (193.51.177.232)
    6.353 ms xe0-0-8-paris1-rtr-131.noc.renater.fr
    (193.51.177.82)  6.998 ms *
11  * * *
12  ae5.mx1.lon2.uk.geant.net (62.40.98.178)  21.651 ms *
    21.445 ms
13  * * *
```

Les commandes `ping` et `ssh`

- `ping adr` : tester si la machine ayant l'adresse `adr` existe et, si c'est le cas, donner des informations sur la qualité de la route vers cette machine.
-
```
> ping www.nancy.fr
PING www.nancy.fr (87.252.12.64) 56(84) bytes of
    data.
64 octets de NANCY-WEB-01.oxyd.net (87.252.12.64)
    : icmp_seq=1 ttl=53 temps=12.7 m
```
- `ssh login@adr` : se connecter à distance à la machine ayant l'adresse `adr` et en tant que l'utilisateur identifié par `login`.

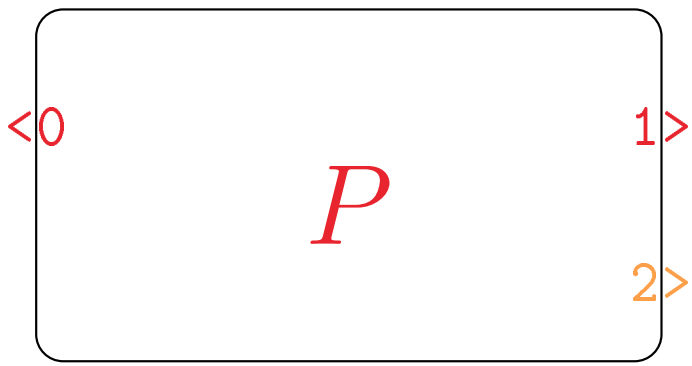
Plan du cours

Plan du cours

Entrées/sorties

Définition (Entrées/sorties)

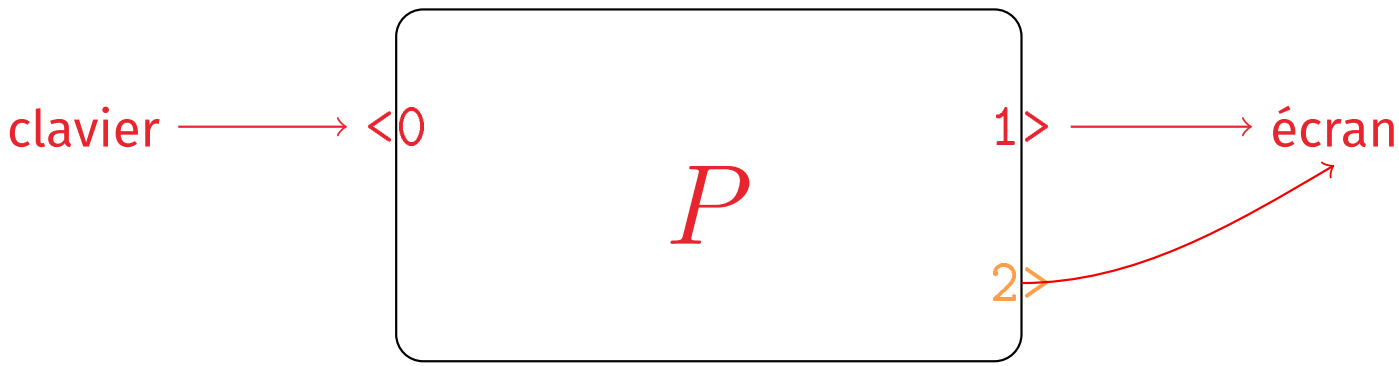
- Entrées : ce sont les paramètres fournis à une commande,
- Sorties : ce sont les affichages faits par une commande.



Entrées/sorties

Définition (Entrées/sorties)

- Entrées : ce sont les paramètres fournis à une commande,
- Sorties : ce sont les affichages faits par une commande.



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

- Entrée Standard <0 : le clavier
- Sortie Standard 1> : l'écran
- Sortie Erreur Standard 2> : aussi associée à l'écran

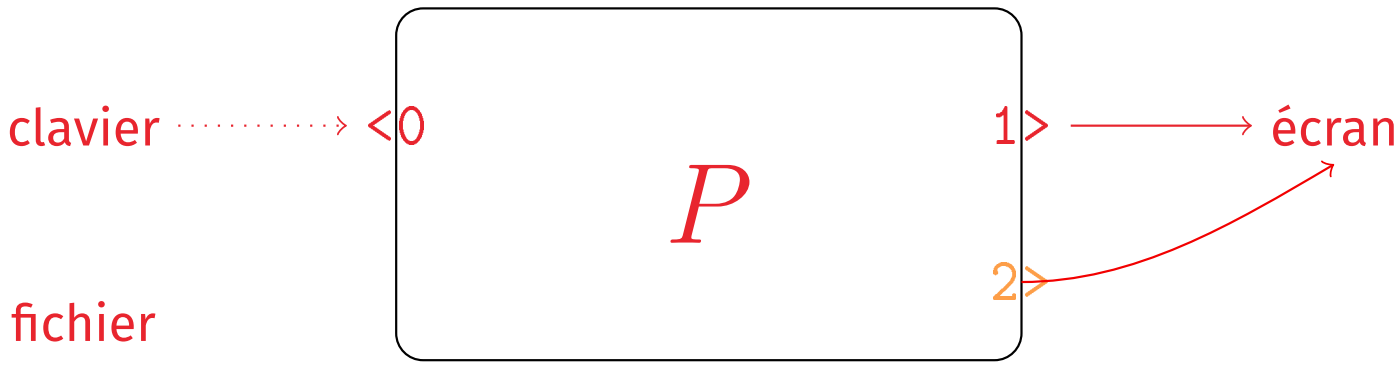
Redirection des entrées/sorties

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.

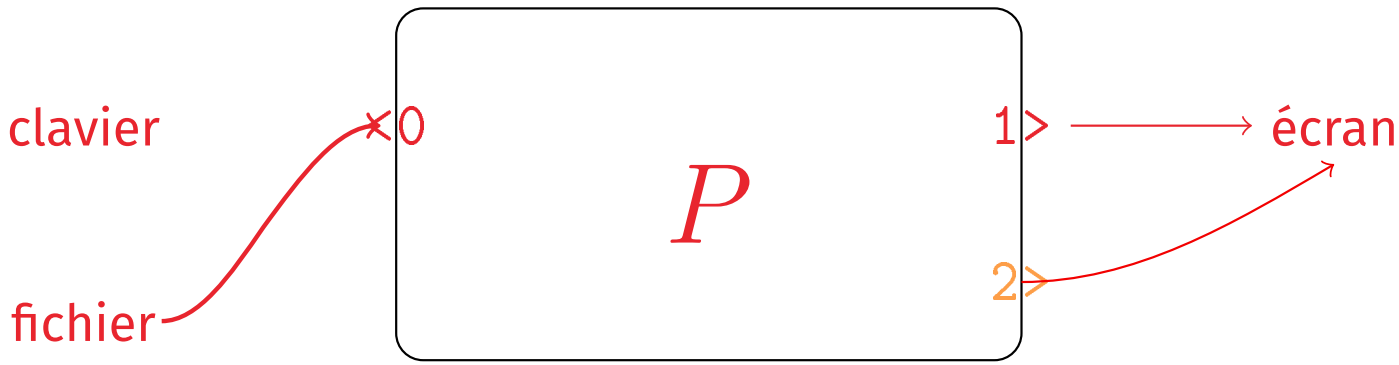
Redirection de l'entrée standard



Redirections de l'entrée standard

- `commande < fichier` : l'entrée de `commande` est le contenu de `fichier`

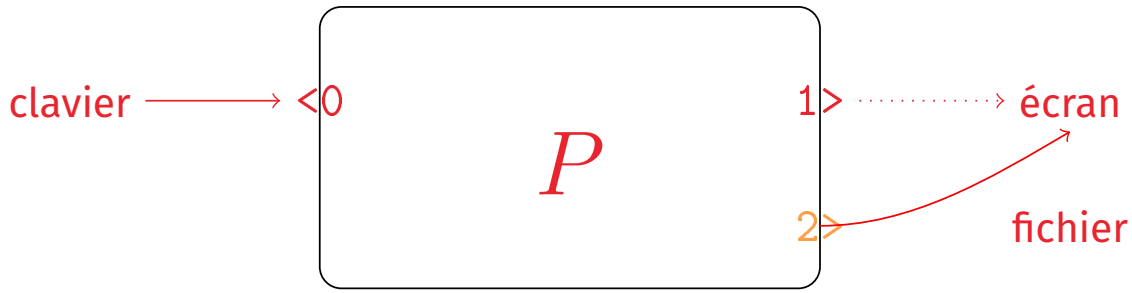
Redirection de l'entrée standard



Redirections de l'entrée standard

- `commande < fichier` : l'entrée de `commande` est le contenu de `fichier`

Redirections de la sortie standard



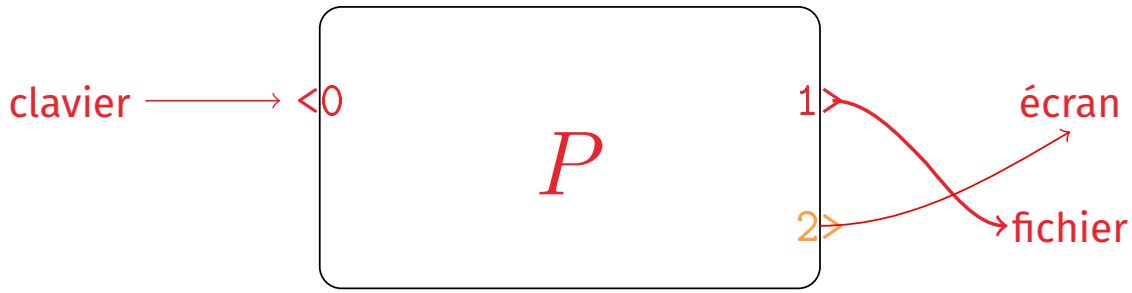
Redirections de la sortie standard

- `commande > fichier` : l'affichage sera écrit dans `fichier` et non à l'écran. Si `fichier` existe, son ancien contenu est effacé.
- `commande » fichier` : l'affichage sera écrit à la fin de `fichier` et non à l'écran.

Exemple :

```
echo "première ligne" > toto
echo "deuxième ligne" >> toto
```

Redirections de la sortie standard



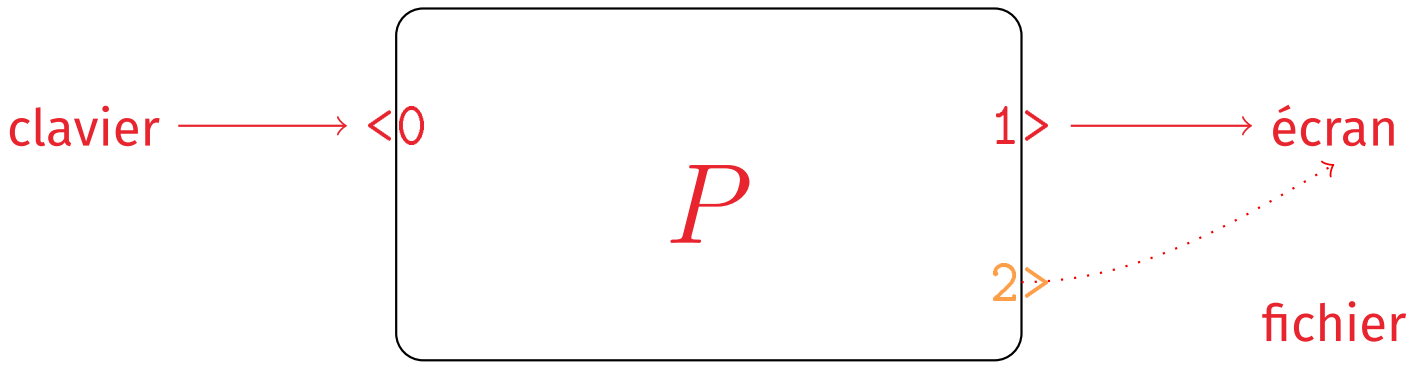
Redirections de la sortie standard

- `commande > fichier` : l'affichage sera écrit dans `fichier` et non à l'écran. Si `fichier` existe, son ancien contenu est effacé.
- `commande » fichier` : l'affichage sera écrit à la fin de `fichier` et non à l'écran.

Exemple :

```
echo "première ligne" > toto
echo "deuxième ligne" >> toto
```

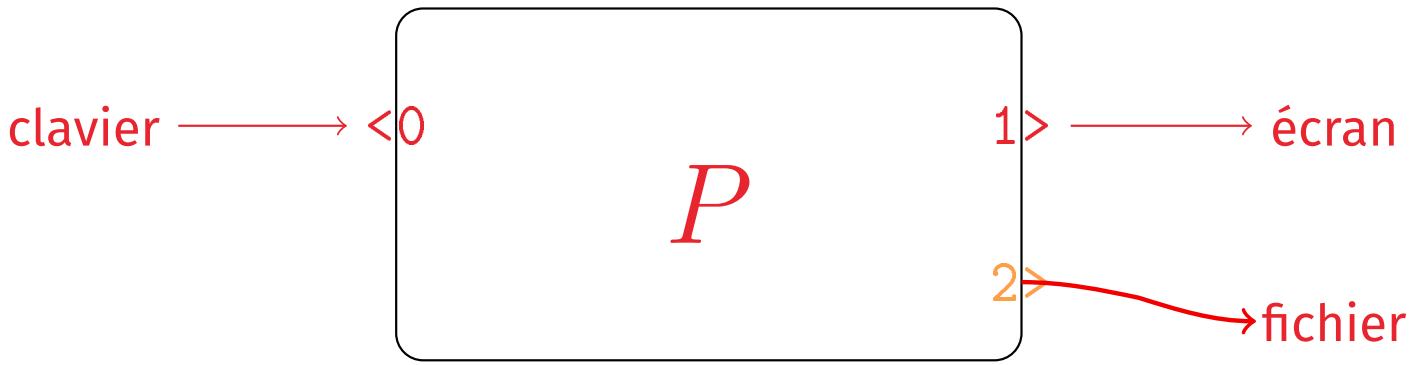
Redirection de l'erreur standard



Redirections de l'entrée standard

- `commande 2> fichier` : redirige les erreurs générées lors de l'exécution de la commande.

Redirection de l'erreur standard



Redirections de l'entrée standard

- `commande 2> fichier` : redirige les erreurs générées lors de l'exécution de la commande.

Plan du cours

Les processus

Définition (Processus)

Un processus est un programme en exécution.

- 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.

Les processus

Définition (Processus)

Un processus est un programme en exécution.

- 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.

Valeur de sortie

- exit 0 $\rightarrow$ le programme a terminé correctement.
- exit $\neq 0$ $\rightarrow$ le programme a terminé avec une erreur

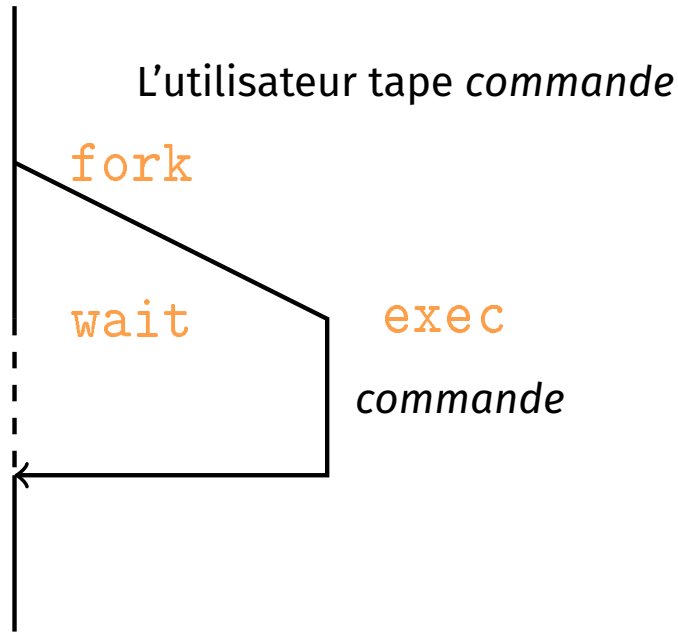
Exécution des commandes

Un interpréteur de commandes (shell) est lancé lorsqu'un utilisateur se connecte.

Lorsqu'on execute une commande il y a 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 (*fork*), 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



Exécution de processus

- Plusieurs processus s'exécutent au même temps sur le même processeur en mode : **temps partagé**
- État d'un processus :
  - **prêt** si toutes les ressources requises sont disponibles sauf le processeur.
  - **bloqué** s'il manque une ressource autre que le processeur.
  - **en exécution** si le processeur est alloué au processus
- L'ordonnanceur (*scheduler*) du système d'exploitation se charge de choisir un processus parmi les processus **prêts** pour lui allouer le processeur pendant une courte période de temps appelée : **quantum**.
- Différentes politiques de sélection du processus prêt à exécuter existent (par exemple : système de priorité).
- À l'expiration du quantum le processus en exécution bascule dans l'état prêt.

Les processus : Identifiants

- Un processus est identifié par un numéro unique `PID` (Process IDentifier).
- Chaque processus linux est créé par un autre processus, appelé **processus père** (ou parent) et désigné par un identificateur `PPID` (Parent Process Identifier). Le processus créé est appelé **processus fils**.
- Un processus primitif (`init`, $\texttt{PID} \leftarrow 1$) est créé au démarrage du système.

Les processus : Arborescence

- Un processus de gestion de swap (`pid` $\leftarrow 0$) est créé avant afin de se charger de migration de processus entre la mémoire centrale et la mémoire secondaire (disque).
- La création d'un processus fils se fait par **clonage** du processus père grâce à l'appel système **fork()**.
- Conséquence : Les processus sont organisés dans une structure arborescente
- **L'arrêt d'un processus père entraîne l'arrêt de tous les descendants**
- La commande nohup cmd permet au processus fils exécutant la commande `cmd` de survivre à l'arrêt de son processus père.

Enchaînement de processus

- **comm1 ; comm2** : Exécution séquentielle. D'abord la `comm1` est exécutée, ensuite `comm2`. Il n'y a pas d'interaction entre les deux.
- **comm1 | comm2** : Exécution en parallèle avec redirection. La sortie de `comm1` devient l'entrée de `comm2`. | représente un tube (pipe)

Exemple :

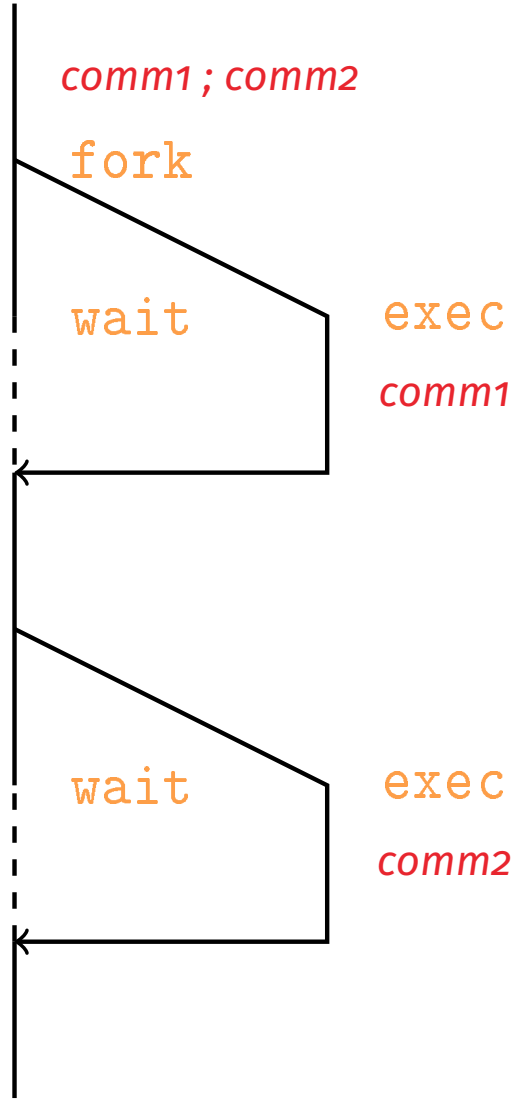
- ```
  find ~ -name "*.pdf" ; ls -a
  ```

  Cette commande trouve tous les fichiers ayant l'extension `.pdf` dans la sous-arborescence de l'utilisateur, puis affiche le contenu du répertoire courant.

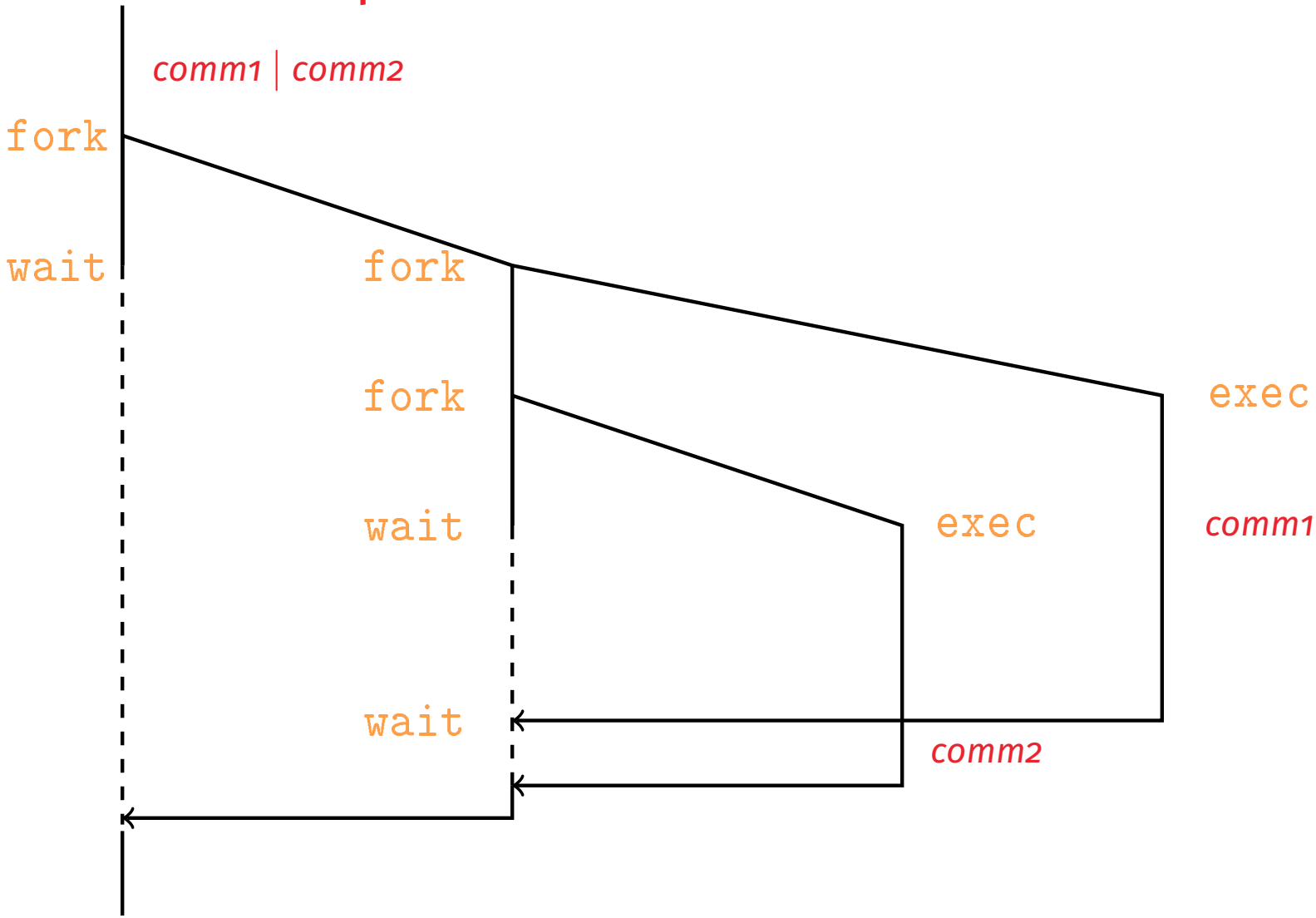
- ```
  find ~ -name "*.pdf" | ls -a
  ```

  Cette commande affiche les informations détaillés sur les fichiers `pdf` qui se trouvent dans l'arborescence de l'utilisateur.

Enchaînement séquentiel



Enchaînement parallèle — tubes



Enchaînement : exemples avec les filtres textes (1/2)

- Soit le fichier `annuaire.txt` dont le contenu est :

```
Quentin:Tarantino:Quentin.Tarantino@gmail.com
Kar-Wai:Wong:wkw@af.hkbu.edu.hk
Julia:Ducournau:Julia.Ducournau@kazak.fr
Kiyoshi:Kurosawa:Kiyoshi.Kurosawa@mail.yahoo.co.jp
Kar-Wai:Wong:wongkarwai@riseup.net
Jean-Luc:Godard:jeanluc@lescahiers.fr
Kar-Wai:Wong:wkw@af.hkbu.edu.hk
```

```
$ head -1 annuaire.txt | cut -d: -f3
Quentin.Tarantino@gmail.com
```

```
more annuaire.txt | grep -i '^Kar-Wai' | cut -d:
    -f3 | wc -l
```

affiche 3, le nombre d'adresses électroniques de Wong Kar-Wai.

Enchaînement : exemples avec les filtres textes (2/2)

```
1 Quentin:Tarantino:Quentin.Tarantino@gmail.com
2 Kar-Wai:Wong:wkw@af.hkbu.edu.hk
3 Julia:Ducournau:Julia.Ducournau@kazak.fr
4 Kiyoshi:Kurosawa:Kiyoshi.Kurosawa@mail.yahoo.co.jp
5 Kar-Wai:Wong:wongkarwai@riseup.net
6 Jean-Luc:Godard:jeanluc@lescahiers.fr
7 Kar-Wai:Wong:wkw@af.hkbu.edu.hk
```

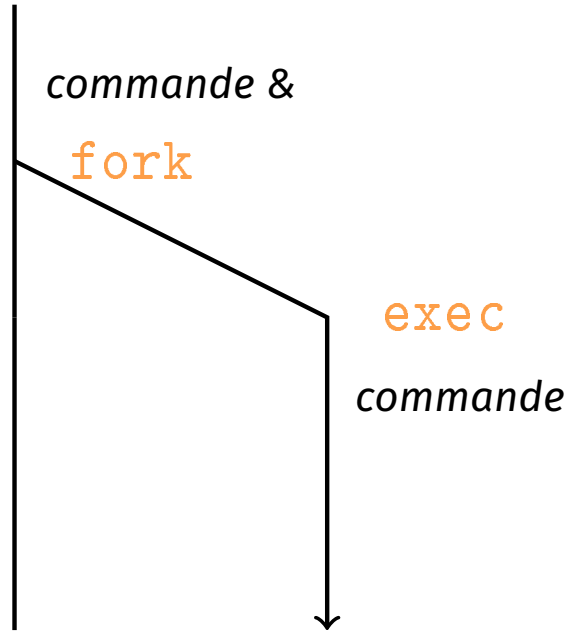
- Que font les deux commandes suivantes ? :

```
more annuaire.txt | cut -d: -f1-2 | sort | uniq
    | wc -l

tail -1 annuaire.txt | tr : '\n' | wc -l
```

Exécution en tâche de fond

- **appli &** : l'application `appli` est exécutée en parallèle avec l'interpréteur de commandes. Ainsi, l'interpréteur n'attend pas la fin de l'application et peut exécuter d'autres commandes.



Il est préférable de lancer toutes les applications dotées d'interface graphique en tâche de fond.

Exécution en tâche de fond

L'interpréteur de commandes (shell) maintient une table des processus en cours d'exécution.

Exécution en tâche de fond

L'interpréteur de commandes (shell) maintient une table des processus en cours d'exécution.

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.

Exécution en tâche de fond

L'interpréteur de commandes (shell) maintient une table des processus en cours d'exécution.

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.

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`.

Processus : Commandes de gestion

ps (*process status*)

Affiche les informations sur les processus exécutés dans le système.

- Options principales :
  - a : informations sur les processus de tous les utilisateurs
  - u : informations sur les processus de l'utilisateur propriétaire
  - x : informations sur tous les processus, y compris ceux qui ne sont pas contrôlés par un terminal.

  Souvent on utilise :

```
ps -aux
```

- D'autres options :
  - -u uid : afficher les processus appartenant à l'utilisateur `uid`
  - -e : informations sur tous les processus
  - -f : afficher des informations détaillées, y compris le `PPID`

Processus : Commandes de gestion

- **top** : affiche l'état des processus en temps réel
- **jobs** : affiche la liste des tâches lancées par l'interpréteur de commandes.
- **fg %n** : permet de ramener la tâche numero n à l'exécution en premier plan.
- **bg %n** : permet de ramener la tâche numero n à l'exécution comme tâche de fond.

Droits des processus

- Les processus peuvent accéder aux systèmes de fichiers et modifier le contenu des fichiers.

$\Rightarrow$ Les processus ont également des droits d'accès aux fichiers.

- Par défaut, les droits d'un processus sont ceux de l'utilisateur ayant lancé le processus.

Les bits SUID et SGID (1/2)

Problème

- `passwd` permet de modifier le mot de passe d'un utilisateur.
- Tous les utilisateurs peuvent exécuter cette commande.
- Or, cette commande modifie les fichiers `/etc/passwd` et `/etc/shadow` possédés par `root`.

$\Rightarrow$ impossible en théorie car lorsque la commande est exécutée par l'utilisateur `machin`, un processus est lancé avec les droits de `machin` qui n'a pas les droits pour modifier `/etc/passwd` et `/etc/shadow`.

Les bits SUID et SGID (2/2)

Solution : le bit SUID, u+s

- SUID = Set User ID
- Si le bit SUID est positionné à 1 sur un fichier exécutable alors le processus créé lors du lancement de l'exécutable a les droits du propriétaire du fichier
- Exemple : le bit SUID de `/usr/bin/passwd` est positionné à 1 $\Rightarrow$ tout processus créé par l'exécution de `passwd` a les droits de `root` (et peut modifier `/etc/passwd` et `/etc/shadow`).

Le bit SGID, g+s

- SGID = Set Group ID
- bit SGID : même fonctionnement que le bit SUID mais pour les groupes. Le processus créé a tous les droits du groupe propriétaire du fichier exécutable.

Le bit de collage

Originalement ce bit a été introduit pour assurer le maintien de l'exécutable en mémoire même lorsqu'aucune exécution n'est en cours...

Le bit de collage

Originalement ce bit a été introduit pour assurer le maintien de l'exécutable en mémoire même lorsqu'aucune exécution n'est en cours...

Bit de collage (sticky bit, $+t$)

Si le sticky bit est positionné à 1 sur un répertoire avec le droit exécutable, alors tous les utilisateurs peuvent utiliser ce répertoire, mais ils ont accès exclusivement à leurs fichiers.

- Exemple : le répertoire `/tmp` a le sticky bit positionné.

Plan du cours

Interaction avec les processus : Les signaux

Définition (Signal)

Un signal est un événement asynchrone destiné à un ou plusieurs processus.

- Les deux commandes **trap -l** et **kill -l** donnent la liste des signaux du système.
- Chaque signal est identifié par un entier et par un nom symbolique.
- Un signal peut être envoyé par le système ou par un autre processus.
- À la réception d'un signal, un processus interrompt son exécution et exécute une fonction de traitement du signal avant de reprendre son exécution (si c'est encore possible).
- Chaque signal est associé à un traitement par défaut que le processus peut changer ou même ignorer.

Exemples de signaux

Signal	Num.	Fonction	Raccourcis clavier
SIGINT	2	Arrêter un processus	CTRL+c
SIGQUIT	3	Quitter après un *core dump*	CTRL+\
SIGSTOP	20	Suspendre un processus	CTRL+z
SIGKILL	9	*Tuer* un processus	-
SIGCONT	18	Reprendre un processus	-

Envoi & traitement de signaux

- **kill -signum pid** : envoyer le signal numéro `signum` au processus identifié par `pid`.

```
kill -9 2323 # arrête immédiatement le processus
    2323
```

- **trap 'cmd' numsig** : `cmd` est la nouvelle commande de traitement à exécuter à réception du signal `signum`.

```
trap 'echo hello' 2
# affiche 'hello' chaque fois qu'on envoie le
    signal SIGCONT (taper CTRL+c)
```

- **trap '' numsig** : inhiber le traitement du signal `numsig`.
- **trap numsig** : rétablir le traitant par défaut du signal `numsig`.

Plan du cours

Environnement utilisateurs

- À la connexion d'un utilisateur, le système exécute un ensemble de commandes pour configurer l'environnement de travail :
  1. Spécifier les commandes accessibles,
  2. Renommer certaines commandes (définition des alias)
  3. Définir les droits par défaut
  4. Options de présentation

- Les commandes de configuration sont spécifiées dans différents fichiers : `.login`, `.profile`, `.bashrc`

- La configuration de l'environnement est décrite par un ensemble de variables d'environnement

Plan du cours

Variables d'environnement

- Le nom d'une variable est une chaîne alpha-numérique avec la possibilité d'utiliser le caractère « _ »
- Le nom ne doit pas commencer par un chiffre
  - Noms valables : `var1`, `_var`
  - Noms invalides : `1var`, `-var`.
- `env` : affiche l'ensemble des variables définies dans un *shell*

Variables d'environnements : exemples

- `PATH` : Liste des répertoires dans lesquels le shell cherche à localiser les exécutables.
  - Lorsqu'on tape une commande (externe) le shell exécute le premier exécutable ayant le nom de la commande demandée en explorant les répertoires listés dans la variable `PATH`.
  - la commande `which nom_commande` retourne la référence absolue (s'il existe) de la commande à exécuter.
- `PS1` : la chaîne de prompt,
- `SHELL` : le type d'interpréteur de commande utilisé (cash, csh, tsh, sh,...)
- `USER` : Le login de l'utilisateur courant.
- `HOME` : le répertoire personnel de l'utilisateur courant.
- `PWD` : le répertoire courant.
- `RANDOM` : un entier aléatoire.

Plan du cours

Les alias

Un alias permet de renommer une commande.

- Déclaration :

```
alias mon_alias=ma_commande
```

- Suppression :

```
unalias mon_alias
```

- Lister les alias :

```
alias
```

Plan du cours

Programmation de scripts shell

- Pourquoi la programmation shell ?
  - Enchaînement complexe de commandes de base.
  - Automatiser des tâches d'administration.

- Avantages :
  - ☺ Code compact : rapide à écrire.
  - ☺ Peu d'interaction avec l'utilisateur.
  - ☺ Accès à toutes les commandes de base du système.

- Inconvénients :
  - ☹ Code compact : difficile à lire et à comprendre.
  - ☹ Différences syntaxiques entre les différents shells.

Les shells existant : bash, sh, csh, zsh ksh, tcsh, ...

Au début de chaque script, on spécifiera :

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

Plan du cours

Manipulation de variables

- Déclaration & initialisation
  - `nom_de_variable=valeur` : création ou affectation d'une variable.
  - Exemple :

```
var1="LP ASRALL"
```

  - **pas d'espaces autour de « = »**
- Saisie clavier
  - `read nom_de_variable`
- Affichage :
  - `echo $nom_de_variable`

```
> echo $var1
'LP ASRALL'
> echo var1
var1
> read var2
IUT Charlemagne
> echo $var2
IUT Charlemagne
```

Portée d'une variable

- Par défaut, une variable est accessible dans le shell où elle est créée.
- export nom_de_variable permet au shell de propager la variable aux processus fils crées après l'exécution de la commande `export`.
- Exemple :

```
> var1=1
> var2=100
> export var1
> bash # exécution d'un shell fils
> echo $var1 # va afficher 1
1
> echo $var2 # ne va rien afficher
```

Types de variables

- Type par défaut : chaîne de caractères.
- La notation `$(($nom_de_variable))` permet de traiter une variable comme un entier
- Exemple :

```
> var1=4
> var2=$(($var1*2))
> echo $var2
8
```

Substitutions

- `$VAR` : la valeur de la variable VAR
- `"$VAR"` : la chaîne de caractère obtenue après avoir remplacé $VAR par sa valeur
- `'$VAR'` : la chaîne littérale $VAR
- `` `$VAR` `` : la chaîne qui résulte de l'exécution de la commande donnée dans `$VAR`

Manipulation de chaînes de caractères

- `${#var}` : retourne la longueur de la chaîne contenue dans la variable `var`
- Exemple :

```
> var="IUT Charlemagne"
> echo ${#var}
15
```

- `${var:i:n}` : extrait n caractères à partir de la position i.
- `${var:i}` : supprime les caractères de 0 à i.

```
> var="IUT Charlemagne"
> echo ${var:0:3}
IUT
> echo ${var:5:6}
harlem
> echo ${var:10}
magne
```

Plan du cours

Variables particulières

- $i : Le i^e paramètre (le paramètre 0 étant le nom du script)
- $# : Le nombre d'arguments passés sur la ligne de commandes
- $* : La liste des arguments passés sur la ligne de commandes
- $$: PID du processus exécutant la commande.

Exemple : manipulation de paramètres

- script param.bash

```
1 #!/bin/bash
2 echo "Le programme $0 a $# paramètres"
3 echo "Le premier paramètre a la valeur $1"
```

- L'appel `param.bash Hello there` affiche :

```
> bash param.bash Hello there
Le programme param.bash a 2 paramètres
Le premier paramètre a la valeur Hello
```

Modification de la liste de paramètres

- set liste_param : redéfinition de la liste de paramètres
- shift n : décaler la liste de paramètres de n paramètres à gauche
- Exemple : script param.bash

```
1 set alpha beta gamma
```

liste =

0	1	2	3
param.bash	alpha	beta	gamma

```
1 shift 2
```

liste =

0	1
beta	gamma

```
1 echo $1 # affiche gamma
```

Plan du cours

Tests et expressions logiques

- Notations :
  - `test expr`
  - `[ expr ]`

- Opérateurs logiques :
  - Négation : `! expr`
  - ET (*and*) : `expr1 -a expr2`
  - OU (*or*) : `expr1 -o expr2`

Tests sur les chaînes de caractères

- Égalité : `s1 = s2`
- Différence : `s1 != s2`
- Chaîne vide : `-z s`
- Chaîne non vide : `-n s`

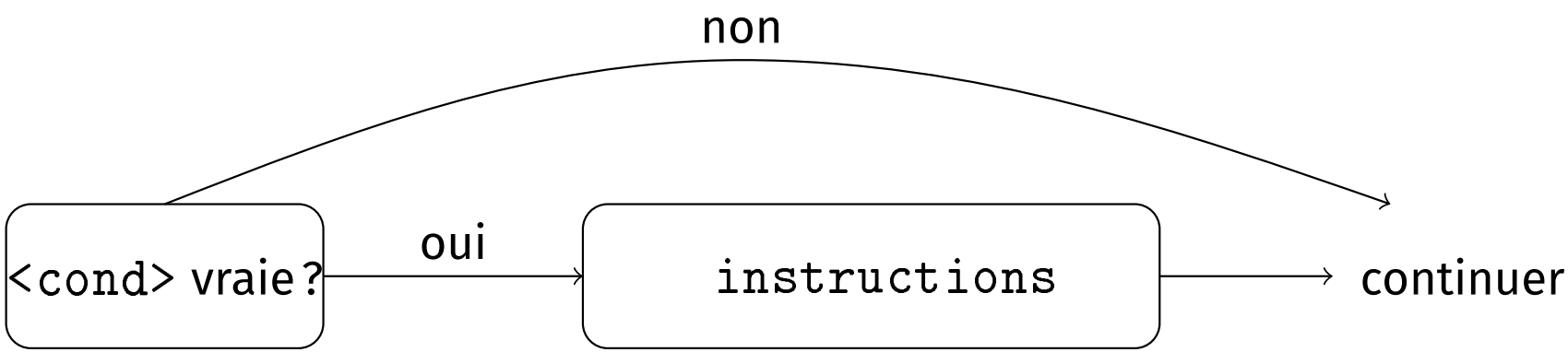
Tests sur les nombres

- `n1 -eq n2` : $=$
- `n1 -ne n2` : $\neq$
- `n1 -lt n2` : $<$
- `n1 -le n2` : $\leq$
- `n1 -gt n2` : $>$
- `n1 -ge n2` : $\geq$

Tests sur les fichiers

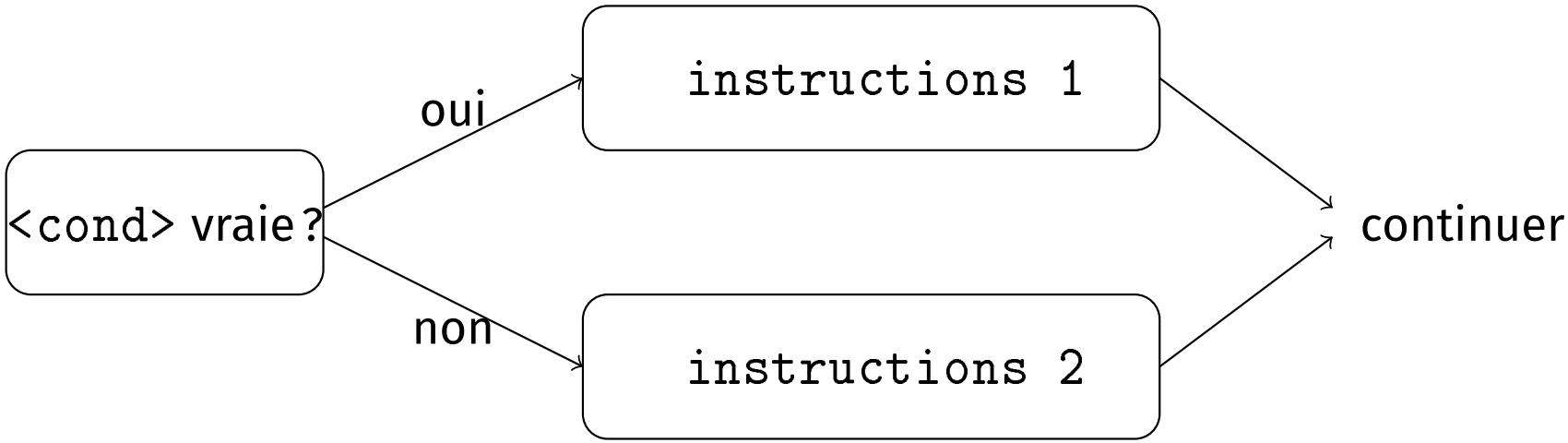
- `-e fich` : existence du fichier `fich`
- `-s fich` : existence d'un fichier non vide
- `-f fich` : existence du fichier ordinaire
- `-d rép` : existence du répertoire `rép`
- `-r fich` : existence du droit de lecture
- `-w fich` : existence du droit d'écriture
- `-x fich` : existence du droit d'exécution

Instruction conditionnelle (1/2)




```
1 if <cond>
2 then
3   instructions
4 fi
```

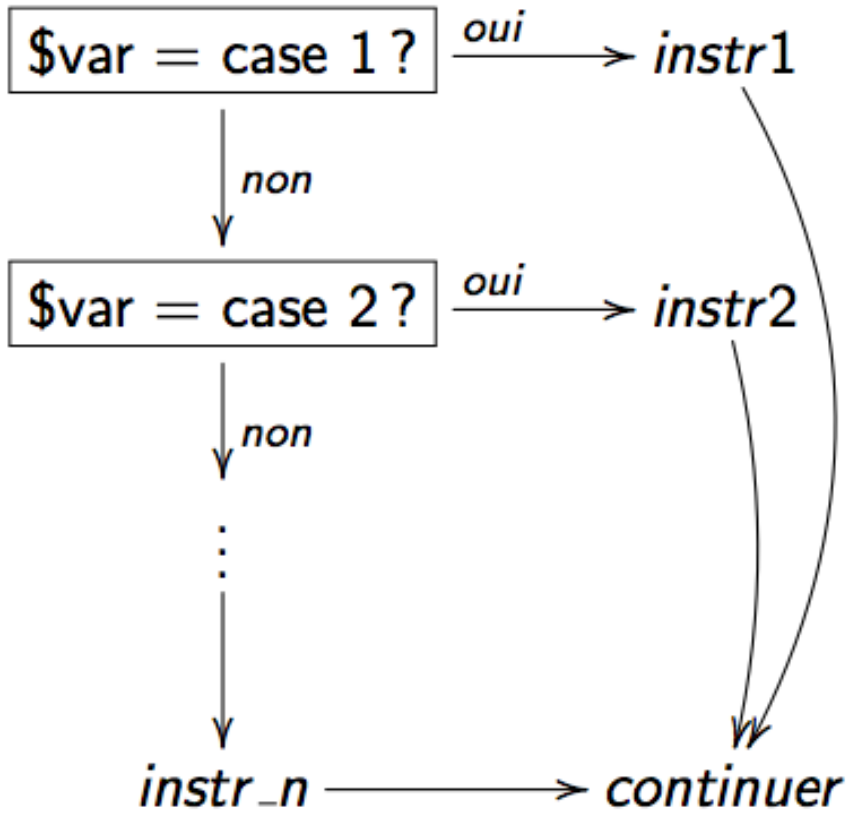
Instructions conditionnelles (2/2)




```
1 if <cond>
2 then
3   instructions 1
4 else
5   instructions 2
6 fi
```

Choix par cas

```
case var in
<cas 1>) instr_1
;;
<cas 2>) instr_2
;;
...
*) instr_n
;;
esac
```



Crédits

Crédits

Diapositives basées en (grande) partie sur le cours de Giulio MANZONETTO (IUT de Villetaneuse, Université Sorbonne Paris Nord), version 2014-2015