Principes des Systèmes d'exploitation

IUT de Villetaneuse
D. Buscaldi

1. Introduction

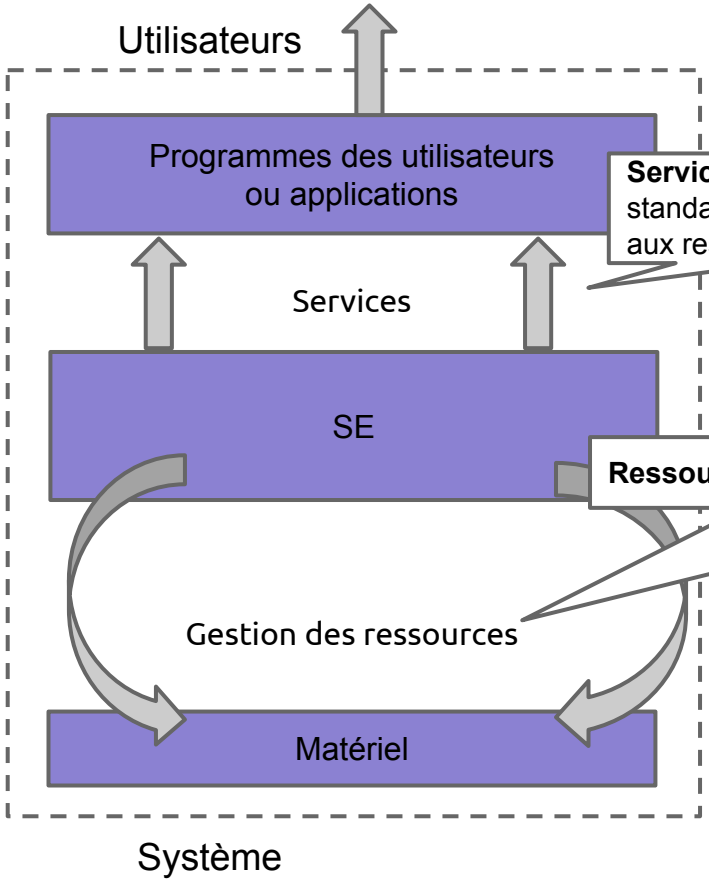
Système d'Exploitation (SE)

- Logiciel de base tournant à tout moment qui
  - facilite l'usage du système informatique
    - Permet d'accéder au matériel de façon **transparente**: un programme n'a pas à savoir s'il écrit sur un disque ext3 ou une clé USB fat32
  - accroît l'efficacité de l'utilisation
    - optimise l'usage de la machine (taux d'occupation du CPU, minimisation des mouvements des têtes de lecture des disques, minimiser le swap, gestion de l'énergie sur les systèmes portables, etc...)
  - accroît la fiabilité des opérations
    - éviter de planter !
    - tolérance à l'erreur (blocs disques défectueux, reboot sauvage, etc)
    - fonctionnement sûr et prévisible dans diverses situations

Types de SE

- Mono- vs. Multi- utilisateur
  - Mono: DOS, Windows 95, Android, iOS (mais on ne peut pas être “root”)…
  - Multi: Linux, FreeBSD, Windows (récents) …
    - Gestion des droits d’accès pour la protection du système et des données

- Mono- vs. Multi- tâche
  - Mono: un seul processus en exécution (que des vieux SE)
  - Multi- tâche:
    - Plusieurs processus en exécution au même temps
    - On a besoin de:
      - Protéger les ressources partagées (mémoire, cpu, etc..)
      - Empêcher les processus de se gêner entre eux
      - Permettre aux processus d'échanger des informations entre eux



Services: interfaces standardisé pour l'accès aux ressources

Exemples
- Gestion de la mémoire centrale
- Gestion de la mémoire de masse (disques)
- Communication avec les périphériques (pilotes, contrôleurs)
- Partage des ressources entre plusieurs usagers
- Communication avec l'utilisateur par l'interprétation d'un langage de commande (shell)
- ...

Ressources: tout ce qui est nécessaire à la réalisation du traitement

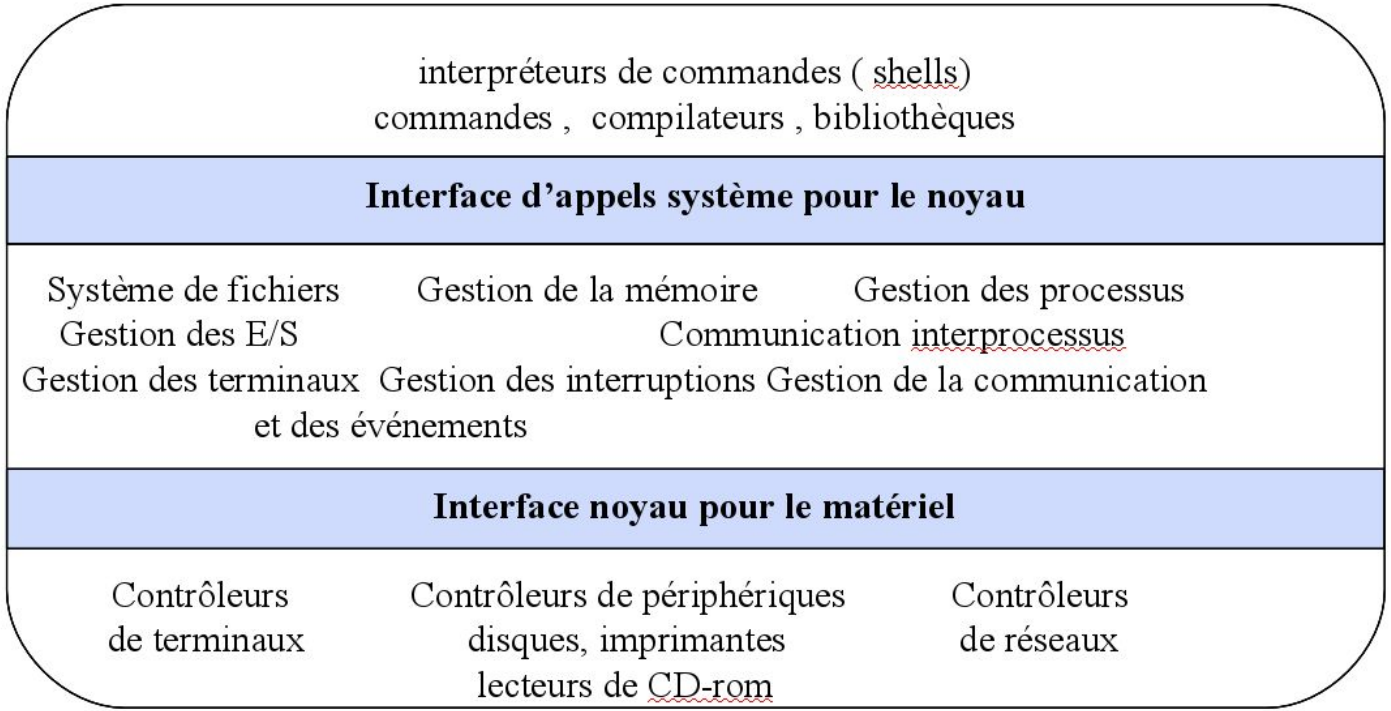
Exemples
- UC (temps d'unité centrale)
- mémoire (place dans la mémoire centrale MC)
- unité d'entrées/sorties (canal d'E/S, buffers)
- périphériques (disques, lecteur CD, imprimante…)
- programmes d'application (éditeurs, compilateurs,…)
- fichiers de données
- objets et moyens du système (verrous, sémaphores,…)

Gestion de ressources

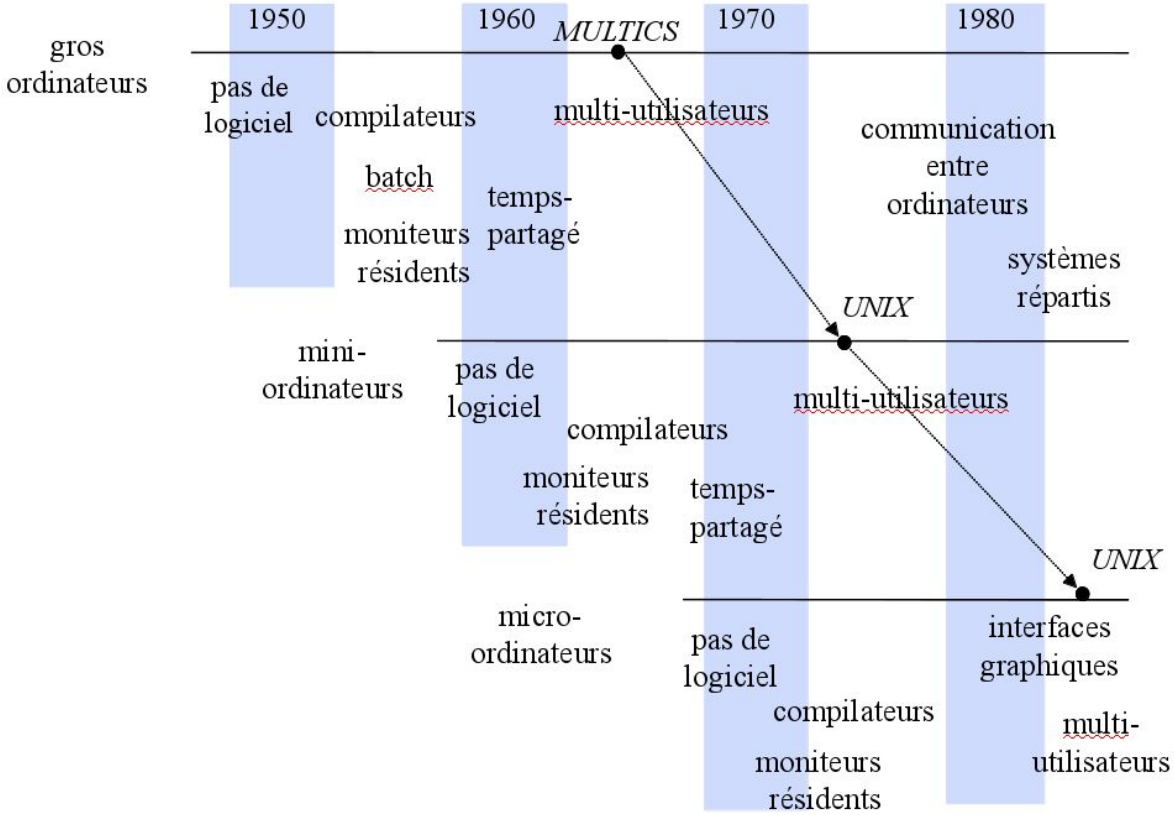
- allocation
  - allocation de la mémoire, de l'UC, de l'espace sur disque,…
- contrôle
  - contrôle d'accès aux fichiers, de la libération des buffers,…
- coordination de l'utilisation
  - Ex: coordination de la lecture de fichier avec l'exécution du programme, des activités dans l'UC, des demandes d'impression …

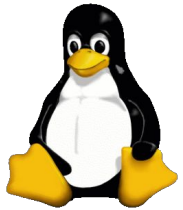
Composants d'un SE

- Découpage en modules par groupe de fonctions



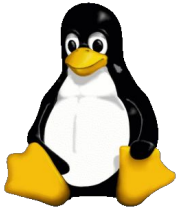
Évolution des SE





Linux

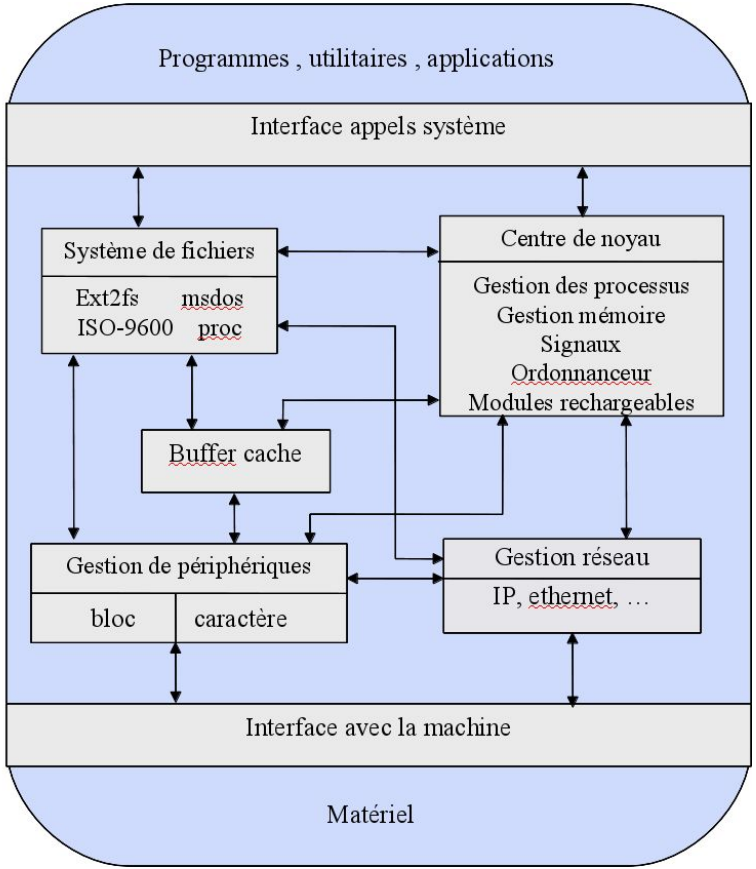
- Unix avec interface graphique pour les PC
- Logiciel libre (ouvert et gratuit)
- Très portable (PC, MacBook,smartphones, Xbox, etc)
- Interface simple et élégante (les appels systèmes)
- Met en œuvre beaucoup de notions intéressantes: processus, droits d'accès, mémoire virtuelle, journalisation, temps réel, modules dynamiques, etc



Linux: un peu d'histoire

- 1969: UNIX (Thompson et Ritchie)
- 1987: Minix (Tanenbaum)
- 1991: premier noyau Linux (Torvalds)
- 1994: v1.0 qui donne le premier système GNU/Linux
  - depuis, de nombreuses distributions:
  - Debian, Ubuntu, Fedora, SuSE, Red Hat, Mandriva, Slackware, ArchLinux, Gentoo, etc
- Distribution: ensemble cohérent de logiciels, la plupart étant logiciels libres, assemblés autour du noyau (kernel) Linux

Architecture de Linux



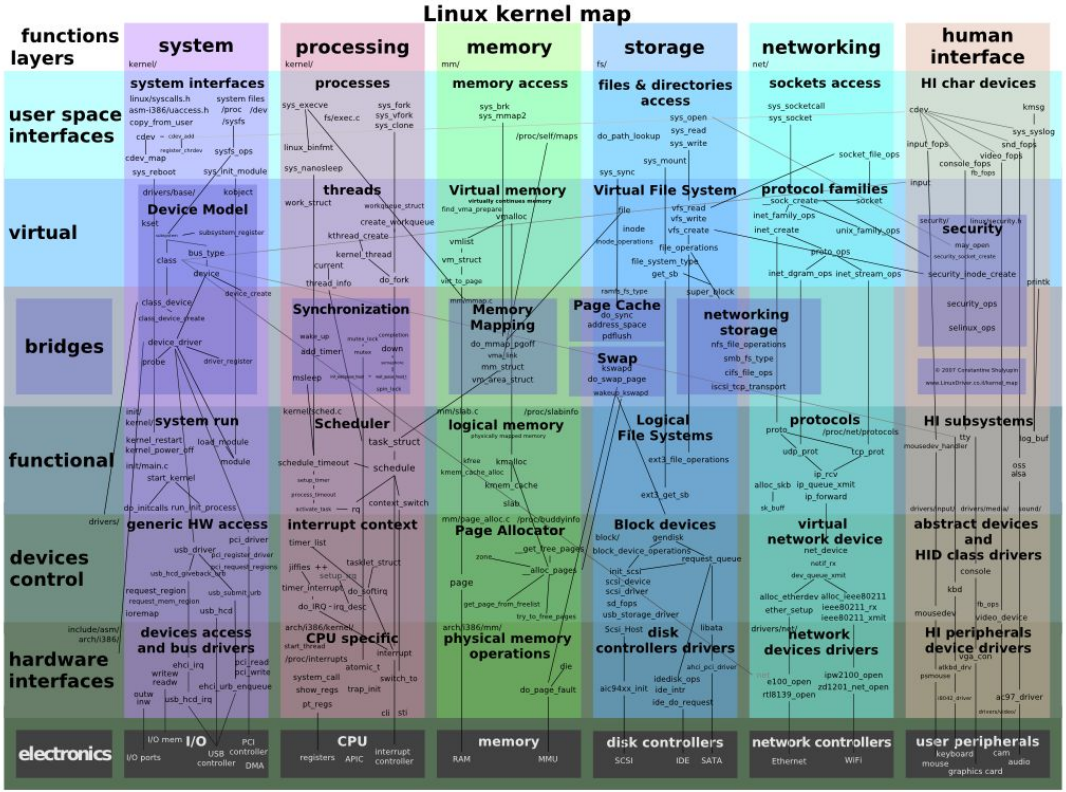
Noyau:

- Espace mémoire isolé, dans lequel est placé tout ou partie du système d'exploitation
- Noyau monolithique: la totalité des programmes du système d'exploitation résident dans l'espace du noyau

  Exemples: Linux, FreeBSD...
- Micro-Noyau: le noyau contient le strict minimum, c'est-à-dire l'ordonnanceur et le programme qui simule la mémoire virtuelle

  Exemples: Minix, MacOS X, ...

Noyau Linux



Utilisateur

Machine

2. L'interpréteur de commandes

Interaction avec l'ordinateur

- Exécution d'une application (programme):
  - fournir les paramètres et les données
  - récupérer les résultats
- Développement des programmes
  - écrire les programmes
  - compilation
  - test et mise au point
- Préparation de documents, impression ….

Tout cela implique une interaction forte avec le terminal et le système de fichiers

Le saviez-vous…?

Avant d'être un SE, Linux était un émulateur de terminal que Linus utilisait pour se connecter via un modem au serveur de son université.

Interfaces

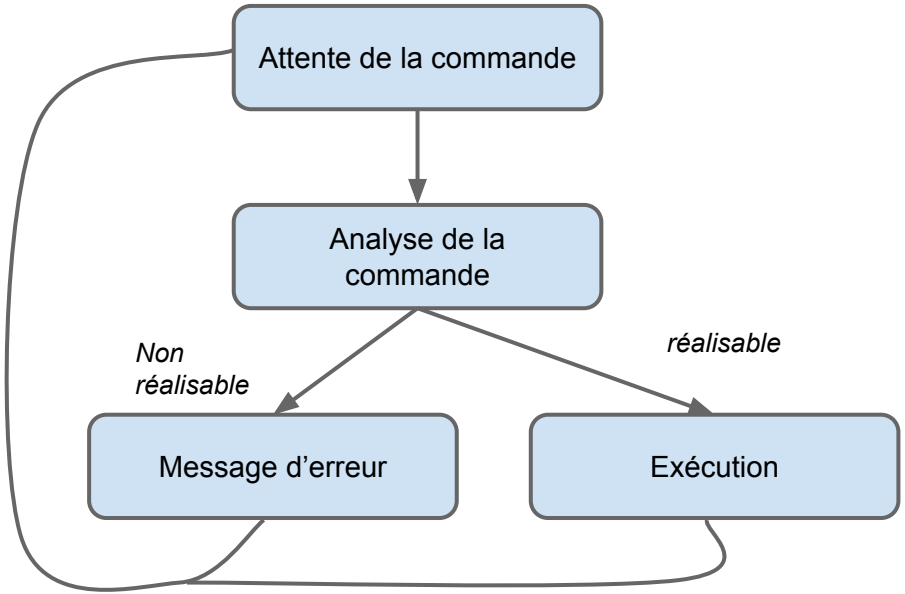
- Interface : moyen pour indiquer au SE les actions voulues

- 2 types d'interfaces:
  - interfaces graphiques : simples à utiliser
  - interpréteur de commandes : souples et puissants

- Linux : interface graphique + interpréteurs de commandes

- Stockage des données, résultats, programmes, documents…
  - la plupart des commandes de base concerne les fichiers

Interpréteur de commandes



- Un programme qui lit et exécute les commandes ligne par ligne
- Il ne fait pas partie du SE
  - pour SE: un programme comme les autres
  - on peut le remplacer avec un autre (Bash, Csh, Zsh, etc)
  - pour l'utilisateur : moyen d'accès aux services du SE
- A la connexion le SE doit en créer un
- Sous Unix on les appelle shells (coquille du SE)

Les deux propriétés des shells

- P1 : Les commandes doivent respecter certaines formes pour être correctement interprétées
- P2 : La Shell interagit étroitement avec le clavier et l'écran

➔ P1 La Shell a sa syntaxe : c'est aussi un langage

Ne pas confondre avec langages de programmation

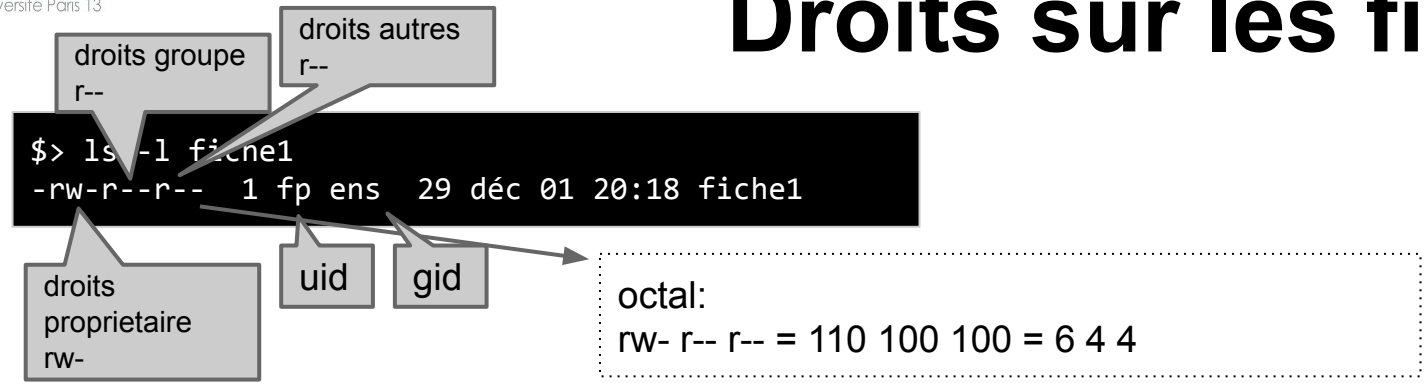
➔ P2 A chaque shell sont associés 3 fichiers " logiques" qui représentent le terminal

- entrée standard : clavier
- sortie standard : écran
- sortie standard d'erreur : écran

Forme générale des commandes

- arguments :
  - paramètres optionnels modifiant/précisant le comportement
    - Ils peuvent être isolés ou regroupés
    - Ex : ls -a -l ou ls -al
  - arguments nécessaires à l'exécution de la commande
  - Ex : cp fichier_source fichier_cible

Droits sur les fichiers



- Droits pour 3 types d'opération sur les fichiers et répertoires :
  - - lecture (r - read)
  - - écriture (w - write)
  - - exécution (x - eXecute)
- Droits pour 3 types d 'utilisateurs :
  - propriétaire
  - membres du groupe propriétaire et
  - les autres.
- 3 groupes de 3 : rwx rwx rwx

Commandes de base

- Répertoires :
  - `mkdir nom_rep` : création
  - `rmdir nom_rep` : suppression
  - `pwd` : affichage du répertoire de travail
  - `cd nom_rep` : changement du répertoire de travail
- Fichiers
  - `ls` : lister les fichiers
  - `rm nom_fich` : supprimer
  - `mv ref1 ref2` : déplacer / renommer
  - `cp fich ref2` : copier
  - `chmod mod ref` : changer les droits
  - `more ref` : afficher le contenu
  - `head (tail) –n ref` : afficher premières (dernières) lignes

Redirection des E/S

- Sortie des commandes peut être orientée sur un fichier physique
  - Entrée par le clavier peut être remplacée par la lecture d'un fichier

```
$> commande > fichier_sortie
$> commande < fichier_entrée
$> commande 2> fichier_erreur
$> commande &> fichier_sortie_et_erreur
```

- Si on veut éviter d'écraser le fichier de sortie ou ajouter à la fin :

```
$> commande >> fichier_sortie_cumulée
```

- Exemple:

```
$> ls -l > liste_fichiers
```

Filtres

- Généralement les commandes ne lisent pas de l'entrée et quelquefois n'affichent rien à la sortie
  - Ex : ls , date… n'utilisent pas l'entrée,
  - cd, cp …n'affichent rien à la sortie

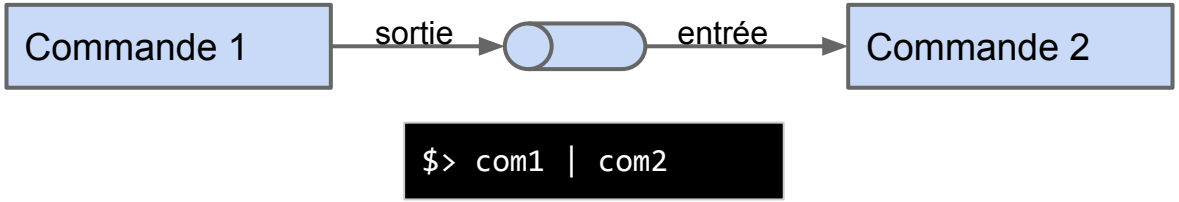
- Filtre : commande qui nécessite une entrée et produit une sortie
  - Ex: cat : concaténation
  - cut : filtrage des colonnes
  - grep : recherche des chaînes
  - more, head , tail : affichages du tout, du début, de la fin
  - wc : statistiques sur les fichiers de caractères

Exemple avec “cat”

```
$> cat > fichier
J’écris dans le fichier à partir du clavier….
^D
$> cat fichier
J’écris dans le fichier à partir du clavier….
$> cat >> fichier
J’ajoute encore, vous verrez…^D
$> cat fichier
J’écris dans le fichier à partir du clavier….
J’ajoute encore, vous verrez…$>
```

Tubes

- Mécanisme qui relie la sortie d'une commande à l'entrée d'une autre



- Avec des tubes on peut écrire des commandes très compactes:

```
$> ls -l | grep rwx | head -10 | cut -t ' ' -f  2-  > resultat.txt
```

lister (format long)

chercher la chaine rwx

prendre les 10 premières

supprimer la première colonne

sauvegarder le résultat dans un fichier

Exécution des commandes

- La shell lance le programme correspondant
  - une nouvelle activité (processus) démarre dans le SE
- Pendant cette activité la shell attend (exécution au premier plan)
- Exécution en mode détaché (en arrière-plan ou asynchrone)
  - on peut éviter l'attente du shell:

```
$> commande  &
```

Exemple:

```
$> simulateur.x  < données > resultat &
[1]  1060
$>
```

Job 1 - processus 1060

- Exécution en séquence (commande2 commence après commande1):

```
$> commande1 ; commande2
```

Grep

- Grep est une commande très importante qui nous permet d'extraire à partir d'un fichier toutes ses lignes qui contiennent une chaîne de caractères:
  - ***grep http /var/network.log*** -> extrait toutes les lignes qui contiennent le mot "http"
- Si la chaîne contient un espace il faut utiliser des guillemets simples:
  - ***grep 'protocol http' /var/network.log***
- Option -i :
  - ***grep -i*** nous permet d'ignorer la différence entre majuscules et minuscules
- Expressions régulières:
  - exemple: ***grep -E "expression" fichier.txt***
  - nous permet d'utiliser les expressions régulières, une suite de symboles décrivant un ensemble (fini ou infini) de mots

Expressions Régulières

Exemple	Explication
A	retrouve la lettre ‘A’
[a-z]	retrouve n’importe quelle lettre en minuscule
[A-Z]	retrouve n’importe quelle lettre en majuscule
[0-9]	retrouve n’importe quelle chiffre
[aeiouAEIUO]	retrouve n’importe quelle voyelle
[^aeiouAEIOU]	retrouve n’importe quel caractère sauf une voyelle
.	retrouve n’importe quel caractère
^	début de ligne
$	fin de ligne
x*	une suite d’occurrences de x (0 ou plus)
x+	une suite d’occurrences de x (1 ou plus)
x?	une occurrence optionnelle de x (0 ou 1)
x{n,m}	entre n et m occurrences de x (note: x{n} -> exactement n occurrences de x)
x|y	x ou y

Exemples

- **[Ww]ork(s|ing|ed)?**
  - on peut retrouver les formes Work, work, Works, works, Working, working,Worked ou worked.
- **x[ab2X]y**
  - décrit les chaînes xay, xby, x2y, xXy
- **x.+[yz]**
  - toute mot qui commence par x, suivi par une séquence de n'importe quels caractères, suivie par y ou z
- **[A-Z]{2}-[0-9]{3}-[A-Z]{2}**
  - expression qui correspond aux plaques d'immatriculation (en France)

Bash

- Bourne Again SHell
- Shell native de Linux (alternatives: csh, zsh, etc.)
- Configuration:
  - Fichier `.bashrc` dans le répertoire home
  - Exemple de fichier `.bashrc`:

```
alias ll= ‘ls -l --color’
```

- Alias nous permet de construire des synonymes
- Historique des commandes
  - commande `history`
- On peut écrire des programmes en Bash

Programmation Bash

Exemple de script bash:

```
#!/bin/bash

for FILE in $*
do
if [ -e $FILE ]
then
echo "Le fichier $FILE est présent dans le
répertoire courant."
fi
done
exit 0
```

Les scripts commencent avec **#!/bin/bash**

On peut utiliser des variables

On peut utiliser des structures de contrôle tel que if-then, while, for, etc...

Variables

- Variables prédéfinies, genre:

  PWD : répertoire courant; HOME : répertoire de connexion; PATH

- Variables définies par l'utilisateur (locales)
  - Affectation: `variable=valeur`
  - Exemples:
    - res=2
    - name='Albert Einstein'
- Attention: le type de base des variables est la chaîne de caractères

```
#!/bin/bash

val=2+2
echo $val

val=$((2+2))
echo $val
```

Utilisation des variables

- `$var` ou `${var}` pour récupérer la valeur de la variable
  - `${#var}` longueur de la variable en nombre de caractères
- $# donne le nb de paramètres du fichier shell
- $* est la liste (sous format de chaine) des paramètres, séparés par un blanc
- $$: numéro du processus shell correspondant à la commande
- $0: nom du script shell
- $1, $2, $3, … : paramètres
  - La commande **shift** décale la numérotation des paramètres de position

```
#!/bin/bash
echo "Nombre de parametres :" $#
echo "premier parametre : $1"
```

Expressions arithmétiques et tests

- Expressions arithmétiques entre $((et)) :
  - exemple: `$(($var1 + $var2 * 2))`
- Tests sur fichiers: `[ -type_test ref ]`
  - où -type_test peut être:
    - -d (répertoire?), -e (éxiste?), -f (fichier?), -r, -w, -x (droits de lecture/écriture/éxécution?), …
  - exemple:

N'oubliez pas l'espace après **[** et avant **]**

```
#!/bin/bash
if [ -d $1 ]
then
echo "$1 est un répertoire"
fi
```

[ ] c'est un alias pour la commande **`test`**: ici vous pouvez écrire aussi **`test -d $1`**

- Comparaison des chaînes: `[ c1 op c2 ]`
  - où op peut être == (identiques) ou != (différentes)
- Comparaison arithmétique `[ arg1 op arg2 ]`:
  - où op peut être -eq (=), -ne (<>), -lt (<), -le (<=), -gt (>), -ge (>=)

Structures de contrôle

- If-then-else:

```
if condition1
then liste_commandes1
elif condition2
then liste_commandes2
else liste_commandes3
fi
```

Note: if, then, else sur lignes différentes

- Case:

```
case $1 in (titi | toto) echo "found titi or toto";;
           (tata | tutu) echo "found tata or tutu";;
esac
```

- For:

```
for var in liste_de_valeurs
do
liste_de_commandes
done
```

For

- Interprétation: pour tous les objets dans la liste faire un traitement

```
for i in `ls -al`
do echo $i
done
```

- Deuxième interprétation: expression arithmetique
  - expr_arith1 : l'expression arithmétique d'initialisation.
  - expr_arith2 : la condition d'arrêt de l'itération.
  - expr_arith3 : l'expression arithmétique qui fixe le pas d'incrémentation ou de décrémentation

```
for (( expr_arith1 ; expr_arith2 ; expr_arith3 ))
do
 suite_cmd
done
```

```
for (( x=1,y=10 ; x<4 ; x++,y-- ))
do
echo $(( x*y ))
done
```

While/Until

```
while liste_de_commandes
do
liste_de_commandes
done
```

```
until liste_de_commandes
do
liste_de_commandes
done
```

- Exemples:

```
i=0
while (( $i<5 ))
do
echo $i
i=$(( $i + 1 ))
done
```

```
i=0
until [ $i -gt 4 ]
do
echo $i
i=$(( $i + 1 ))
done
```

Tableaux

- Ils peuvent contenir n'importe quel type de données
- Attention à utiliser l'expansion de paramètres (symbole ${}) :

Exemple:

```
#!/bin/bash
# Declaration explicite (sans taille) :
declare -a MonTableau
# ou aussi declare -a MonTableau[10]
MonTableau[0]='12' MonTableau[1]='bonjour'
echo "$((MonTableau[0])) -- $((MonTableau[1]))"
echo "${MonTableau[0]} -- ${MonTableau[1]}"
```

Fonctions

- On utilise le mot **function** suivi par le nom de la fonction
- Ou nom_fonction suivi par ()
- La valeur de retour est le résultat du dernier commande
  - On peut spécifier aussi la valeur de retour avec la commande **return** *n*
  - La commande return ne peut retourner qu'un code de retour

Exemple:

```
#!/bin/bash
function untar {
   cp $1 /store
   tar -xvf $1
   echo 'Done'
}

untar () {
   cp $1 /store
   tar -xvf $1
   echo 'Done'
}
```

Substitution de commandes

- Une commande *cmd* entourée par une paire de parenthèses () précédées d'un caractère $ est exécutée par le shell puis la chaîne $(*cmd*) est remplacée par les résultats de la commande cmd écrits sur la sortie standard.
- Ces résultats peuvent alors être affectés à une variable ou bien servir à initialiser des paramètres de position.

Exemple:

```
repert=$(pwd)
echo mon repertoire est $repert
```

Tours de Hanoï

- Le problème des « tours de Hanoï » peut s'énoncer de la manière suivante :
  - - conditions de départ : plusieurs disques sont placés sur une table A, les uns sur les autres, rangés par taille décroissante, le plus petit étant au dessus de la pile
  - - résultat attendu : déplacer cette pile de disques de la table A vers une table B
  - - règles de fonctionnement :
    - on dispose d'une troisième table C,
    - on ne peut déplacer qu'un disque à la fois, celui qui est placé en haut d'une pile et le déposer uniquement sur un disque plus grand que lui ou bien sur une table vide.
- Le programme shell récursif suivant traite ce problème ; il utilise quatre arguments : le nombre de disques et le nom de trois tables.

Tours de Hanoï (code)

```
#!/bin/bash
if (( $1 > 0 ))
 then
 hanoi $(( $1 - 1)) $2 $4 $3
 echo Deplacer le disque de la table $2 a la table $3
 hanoi $(( $1 - 1)) $4 $3 $2
fi
```

```
$> hanoi 3 A B C
Deplacer le disque de la table A a la table B
Deplacer le disque de la table A a la table C
Deplacer le disque de la table B a la table C
Deplacer le disque de la table A a la table B
Deplacer le disque de la table C a la table A
Deplacer le disque de la table C a la table B
Deplacer le disque de la table A a la table B
$>
```