Système d'exploitation Linux

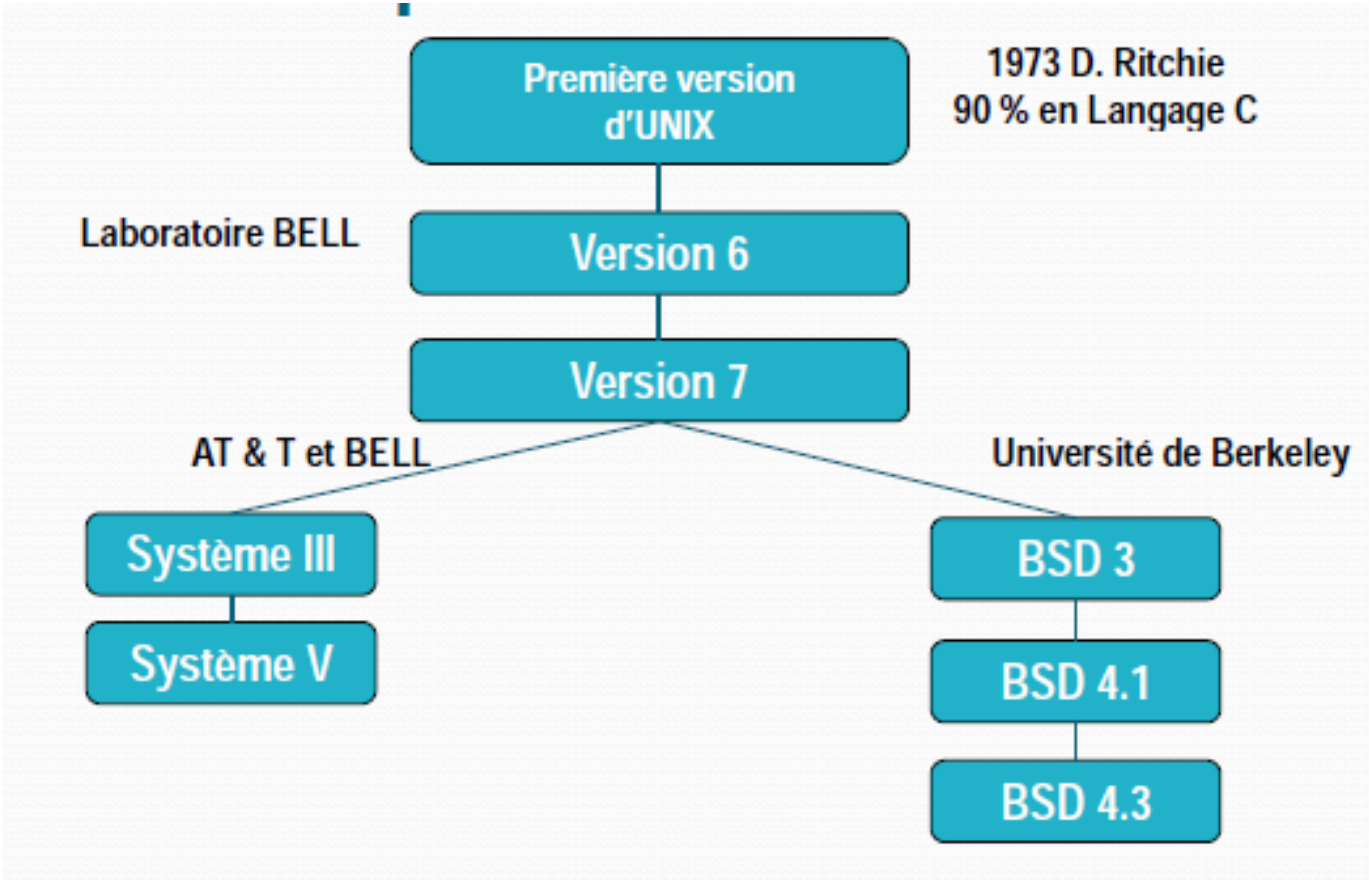
Pr. Youssef GHANOU

Introduction

Historique

- Projet du Système MULTICS (MULTiplexed Information and Computing Service) vers la fin des années 60 entre les laboratoires de BELL et General Electric
- Ken THOMSON, chercheur à BELL, a décidé d'écrire une version allégée de MULTICS en assembleur sur une machine PDP-7
- UNIX a été porté sur d'autres machines PDP11-20 PDP11-45 PDP11-70
  - Réécrire UNIX dans un langage de haut niveau pour faciliter le portagesur d'autres architectures:Réalisation du Langage B qui a été remplacé par la suite par le langage C

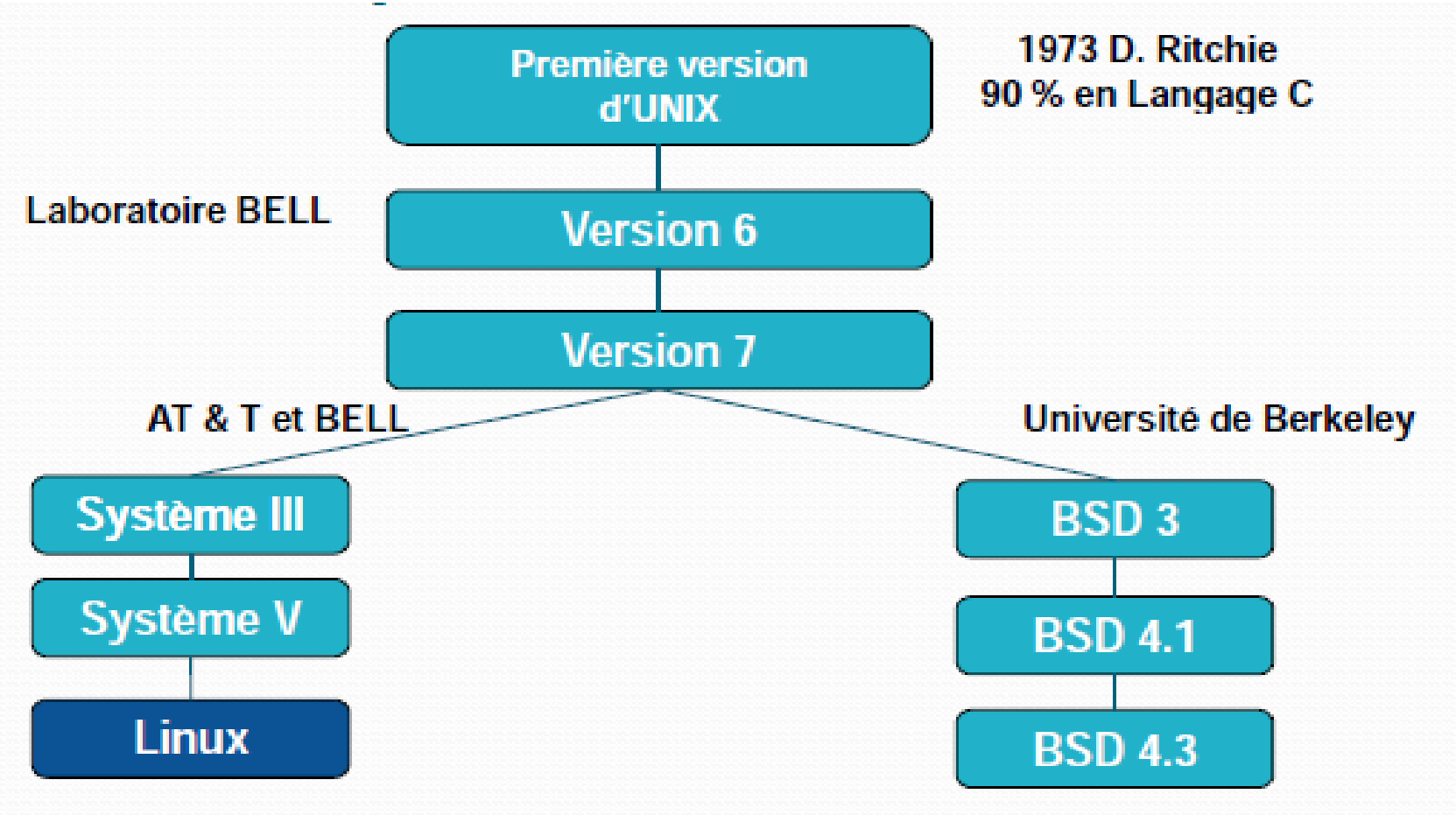
Historique


Première version d'UNIX
1973 D. Ritchie
90 % en Langage C
Laboratoire BELL
Version 6
Version 7
AT & T et BELL
Université de Berkeley
Système III
Système V
BSD 3
BSD 4.1
BSD 4.3

Historique

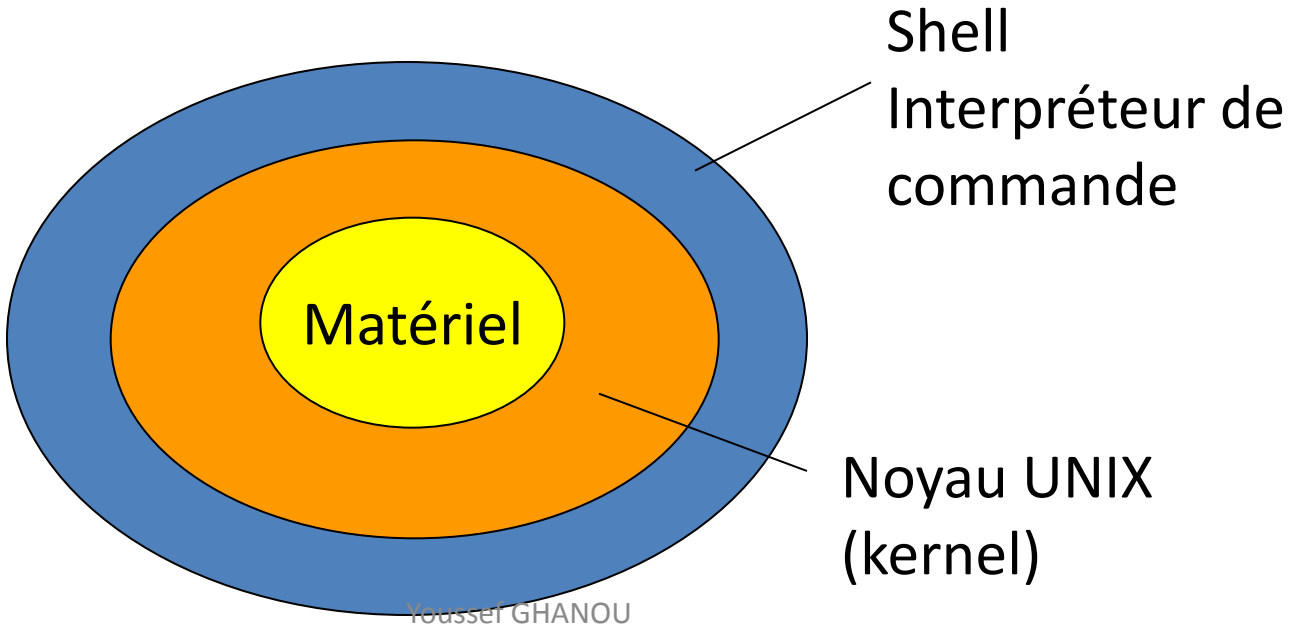
- Vers la fin des années 80, deux versions largement utilisées et sensiblement incompatibles d'UNIX
  - System V release 3
  - BSD4.3 (Berkeley Software Distribution)
- Chaque constructeur ajoutait ses propres améliorations
- Cette incompatibilité a freiné le succès commercial du système
- ➔ la standardisation d'UNIX devient une nécessité

- Standardisation
  - POSIX (Portable Open System Interface eXchange) membre de IEEE
    - Définition d'un ensemble de procédures que doit fournir tout système compatible à la norme
      - Intersection des deux familles
      - Ressemble fortement à leur ancêtre version 7
    - Norme IEEE P1003.1 devenue norme ISO 9945


Première version
d'UNIX
1973 D. Ritchie
90 % en Langage C
Laboratoire BELL
Version 6
Version 7
AT & T et BELL
Université de Berkeley
Système III
Système V
Linux
BSD 3
BSD 4.1
BSD 4.3

Architecture génerale d'Unix

- Le noyau Unix
- Les Shell
- Les programmes utilitaires



Linux

- Propriétés
  - multi-tâches
  - multi-utilisateurs
  - Libre (et gratuit) !!

  - Travailler sous Linux implique une connexion au système
  - Login:
    - Identification de l'utilisateur: *login + mot-de-passeSécurité (login, mot de passe), Seuls les utilisateurs ayant un login et un passwd peuvent se connecter au système*
    - droits accordés par le *super-utilisateur* (**root**)

  - Portabilité
    - Disponible pour plusieurs plateformes (Station de travail, PC, Macintosh)

Linux

- Propriétés
  - Modularité
    - Noyau
    - Utilitaires
  - Système de fichier
    - Arborescent
    - Réparti
    - Réorganisation souple
  - Traitement uniforme des périphériques
    - Un périphérique est traité comme un fichier

Linux

- Propriétés
  - Outils de communication intégrés
    - Talk, write, mail …
  - Système de commandes
    - Très riche
    - Puissant
  - Plusieurs interpréteurs de commandes
    - Exemples : sh, ksh, csh, …
    - Inter chargeables sans redémarrer la machine

Initiation au shell

- Le Shell = interpréteur de commandes
  - interface utilisateur “de base” (interlocuteur avec le syst.)
  - interprétation ligne à ligne
  - plusieurs shells: sh, csh, tcsh, bash, ksh, zsh, ...
  - langage de programmation
- shell par défaut : bash

Initiation au shell - *commandes* -

- Format des commandes:

commande [-option(s)] [argument(s)]

Respecter la casse et les espaces!!

Initiation au shell - *méta caractères* -

- Caractères spéciaux:

  ! ^ * ? [] \ ;

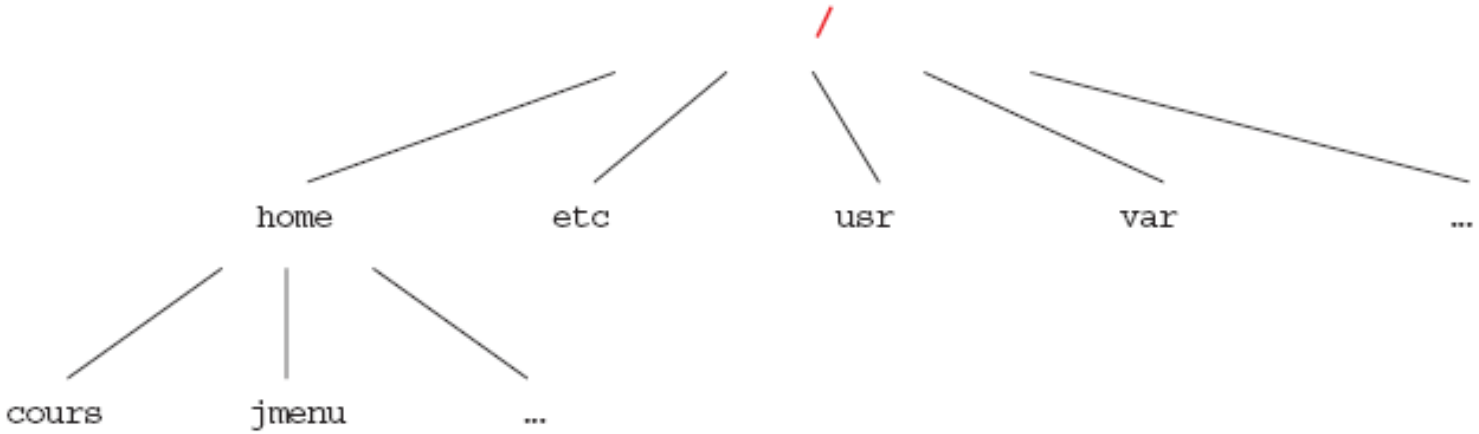
- L'astérisque ou étoile: *****
  - interprété comme toute suite de caractères alphanumériques
  - utiliser **avec précaution** (commande rm par ex...)
- Le point d'interrogation: **?**
  - remplace 1 seul caractère alphanumérique

La gestion des fichiers et des répertoires

La gestion des fichiers et des répertoires

- Stocke les données:
  - Structure arborescente
  - TOUT est fichier
- 3 types de fichiers:
  - fichiers ordinaires
  - répertoires
  - fichiers spéciaux (périph., …)

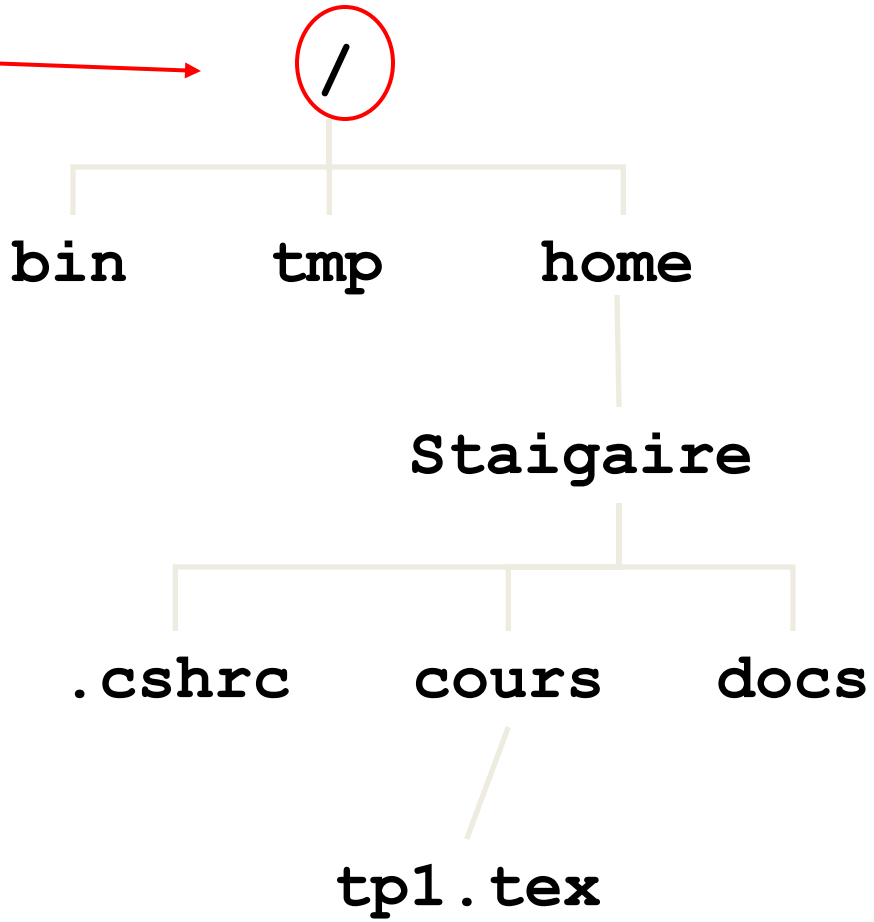
La gestion des fichiers et des répertoires
- l'arborescence -



Le système de fichiers - *l'arborescence* -

répertoire racine

- le répertoire de login: **~**
- le répertoire courant: **.**
- le répertoire supérieur: **..**
- connaître le rép. courant: **pwd**
- lister le contenu: **ls**
  (voir "**man ls**")



chemin d'accès au fichier **tp1.tex**:

-/home/Stagiaire/cours/tp1.tex

-ou bien: **~/cours/tp1.ex**

La gestion des fichiers et des répertoires

- Identificateur (nom)
  - Suite de caractères (jusqu'à 255 caractères)
  - Sensible à la casse
  - Utiliser le caractère d'échappement '\' pour les caractères spéciaux
  - Exemple précéder le caractère espace par \
    Nom\ fichier

La gestion des fichiers et des répertoires

- Caractères "joker"
  - Permettent d'appliquer une commande à un ensemble de fichiers dont le nom vérifie certaines contraintes (ex : le nom commence par la lettre 'p', l'extension est '.doc', …)
  - * : remplace n'importe quelle suite de caractère (y compris la chaîne vide)
  - ? : remplace un et un seul caractère
  - [-] : définit un intervalle

La gestion des fichiers et des répertoires

- ls
  - Affiche le contenu du répertoire courant ou de celui passé en paramètre
- Options

  -l : affiche les informations complètes des fichiers et sous répertoires

  -a : affiche les fichiers cachés

  -R : affichage récursif

  -i : affiche le descripteur des fichiers (i-numéro)

  -d : n'affiche pas le contenu des répertoires

La gestion des fichiers et des répertoires

- pwd
  - Affiche le chemin du répertoire courant
- cd chemin
  - Se déplace vers le répertoire identifié par 'chemin'
- Exemples

$ cd /home/dubois/doc

$ cd ../dubois/doc

La gestion des fichiers et des répertoires

- mkdir (md) nouveau_rep
  - Crée un nouveau_rep dans
    - le répertoire courant
      - nouveau_rep est le nom du répertoire
      - Exemple

        $ mkdir stages
    - Dans le chemin indiqué par la première partie de nouveau_rep
    - Exemple

$ md /home/dupont/stages

$ mkdir ../dupont/stages

La gestion des fichiers et des répertoires

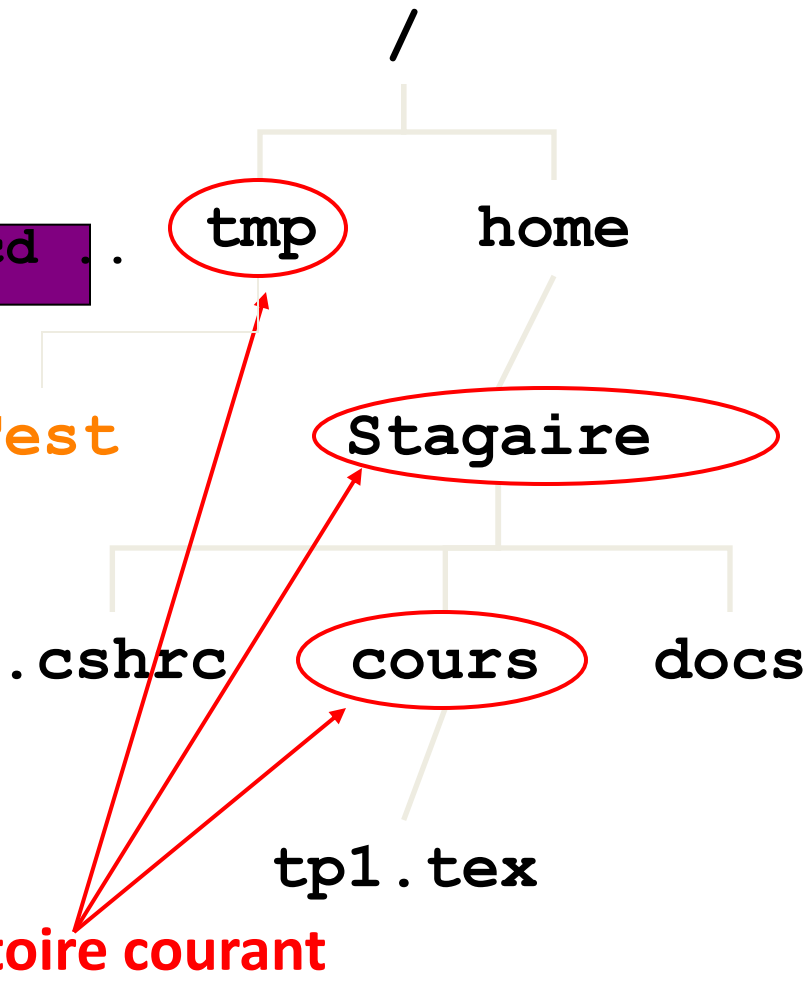
- rmdir repertoire
  - Détruit un répertoire vide
  - Exemple
    - $ *rmdir /home/dupont/temp*

La gestion des fichiers et des répertoires

- **pwd** retourne:
  /home/stagaire/cours
- se déplacer: **cd**

```
[/home/ Stagaire /cours]$ cd ..
[/home/ Stagaire]$
[/home/ Stagaire]$ cd /tmp
[/tmp]$
```

- **chemin relatif**
- **chemin absolu**



- créer un répertoire: mkdir
  [/tmp]$ mkdir Test
- supprimer un répertoire: rmdir
  [/tmp]$ rmdir Test

La gestion des fichiers et des répertoires

- rm fichier
  - Détruit un fichier ou un répertoire non vide
  - Options
    - -r : la commande détruit de manière récursive toute la sous arborescence du répertoire
    - -i : demande la confirmation avant de supprimer le fichier
  - Exemple

$ rm -r doc

rm -i /home/dupont/linux.pdf

La gestion des fichiers et des répertoires

- cat fichier [fichier,...]
  - Concatène et affiche sur la sortie standard le(s) fichier(s) en paramètre
  - Exemple

  $ cat fichier1
- cat fichier [fichier,...]
  - Concatène et affiche sur la sortie standard le(s) fichier(s) en paramètre
- Exemple

$ cat fichier1

$ cat fichier1 fichier2$ cat fichier1 fichier2

La gestion des fichiers et des répertoires

- more fichier
  - Affiche le contenu du fichier page par page
  - Utilisée pour les fichiers longs (contenant plusieurs pages)
    - Q : quitte la commande
    - Return : saute de ligne
    - Espace : saute de page
- Exemple

$ more lettre

La gestion des fichiers et des répertoires

- head [-c nchar -n nline] fichier
  - Affiche le début du fichier
    - Par défaut les dix premières lignes
    - -c nchar : affiche les nchar premiers caractères du fichiers
    - -n nline : affiche les nline premières lignes du fichier
- Exemple

$ head lettre

$head -c 280 lettre

$ head -n 5 lettre

La gestion des fichiers et des répertoires

- tail [-/+c nchar -/+n nline] fichier
  - Affiche la fin du fichier
    - Par défaut les dix dernières lignes
    - -/+c nchar : affiche les derniers caractères du fichier
      + à partir du nchar ème caractère jusqu' à la fin du fichier
      - Les nchar derniers caractères à partir de la fin
    - -/+n nline : affiche les dernières lignes du fichier
      + à partir de la nline ème jusqu'à la fin du fichier
      - Les nline dernières lignes à partir de la fin
  - Exemple

$ tail lettre

$tail -n 6 lettre affiche les 6 dernières lignes

$ tail +n 6 lettre affiche de la ligne 6 jusqu'à la fin du fichier

La gestion des fichiers et des répertoires

- wc [-lwc] fichier
  - Compte le nombre de
    - *-l : lignes*
    - *-w : mots*
    - *-c : caractères*
  - du fichier
    - Par défaut les trois
  - Exemple

    $ wc lettre

La gestion des fichiers et des répertoires

Manipulation des fichiers

- copier : **cp fic1 fic2**
- déplacer/renommer : **mv fic1 fic2**
- effacer : **rm fic**
- afficher le contenu : **cat fic**
- trier le contenu : **sort fic**

La gestion des fichiers et des répertoires

- ln source lien
  - Crée un lien physique sur le fichier source
    - Pas possible pour les répertoires ou fichiers d'autres SGF
  - -s : le lien est symbolique
  - Exemple

$ ln lettre lien_lettre

$ ln lettre -s lien_symbolique

La gestion des fichiers et des répertoires

- Commande tar
  - Permet d'archiver un d archiver ensemble de fichiers dans un seul fichier (d'extension '.tar')
    - Facilite l'organisation (moins d'encombrement dans le SGF)
    - Efficace pour envoyer par mail plusieurs fichiers en attachement
  - Restituer l'ensemble des fichiers à partir du fichier archive (l'opération inverse)
  - Possibilité de compression et de décompression de l'archive en appelant la commande "gzip"

La gestion des fichiers et des répertoires

- Commande tar
  - Syntaxe

    $ tar [options] [fichiers]

Option	Description
-x	Extraire le contenu d'une archive.
-c	Créer une nouvelle archive.
-t	Afficher seulement la liste du contenu de l'archive, sans l'extraire.
-f *Fichier*	Indiquer le nom du fichier archive.
-v	Mode verbeux, affiche le détail des opérations.
-z	Compresser ou décompresser en faisant appel à l'utilitaire gzip.
- j	Compresser ou décompresser avec l'utilitaire bzip2.
-p	Préserver les permissions des fichiers.

La gestion des fichiers et des répertoires

- Commande tar
  - Exemples
    - Créer une archive
      - *$ tar –cvf archive_doc.tar /home/ali/doc*
    - Créer une archive et compression
      - *$ tar –cvzf archive_doc.tar.gz /home/ali/doc*
    - Lister le contenu d'une archive
      - *$ tar –tvf archive_doc.tar*
    - Extraire le contenu d'une archive
      - *$ tar –xvf archive_doc.tar /home/ali/cours*
      - *$ tar –xvzf archive_doc.tar.gz (extraction dans le répertoire courant)*

La gestion des fichiers et des répertoires

- **rpl** *chaine 1 chaine 2 < fic1 > fic2*

  remplace toutes les occurences de chaine1 par chaine2 dans fic1 et émet dans fic2

- ex.:

  rpl " IT " "Italie" < films.cine > films.tele

La gestion des fichiers et des répertoires

- **rpl** *chaine 1 chaine 2 < fic1 > fic2*

  remplace toutes les occurences de chaine1 par chaine2 dans fic1 et émet dans fic2

- ex.:

  rpl " IT " "Italie" < films.cine > films.tele

Les inodes.

- Un fichier contient plusieurs éléments : nom, contenu, longueur, emplacement sur disque, etc.
- Ces informations sont contenues dans des inodes.
- L'inode comporte 3 dates : la date de dernière modification du fichier, la date de dernière
- Affichage des dates :
  - ls –l : date de dernière modification
  - ls –lu : date de dernière modification de l'inode
  - ls –lc : date de dernier accès du fichier
- Les inodes sont les « vrais » fichiers. La hiérarchie des répertoire ne fait que donner de faux noms aux fichiers. Le nom de l'inode est le seul vrai nom

La gestions des Droits

La gestion des droits

- Déterminent les types d'opérations qu'un utilisateur ou une classe d'utilisateurs peuvent effectuées.
  - Chaque fichier peut avoir ses propres droits d'accès
  - Système de protection très puissant
- *Droits d'accès : Trois groupes d'autorisation, l'utilisateur* propriétaire, les personnes appartenant au groupe propriétaire et les autres.

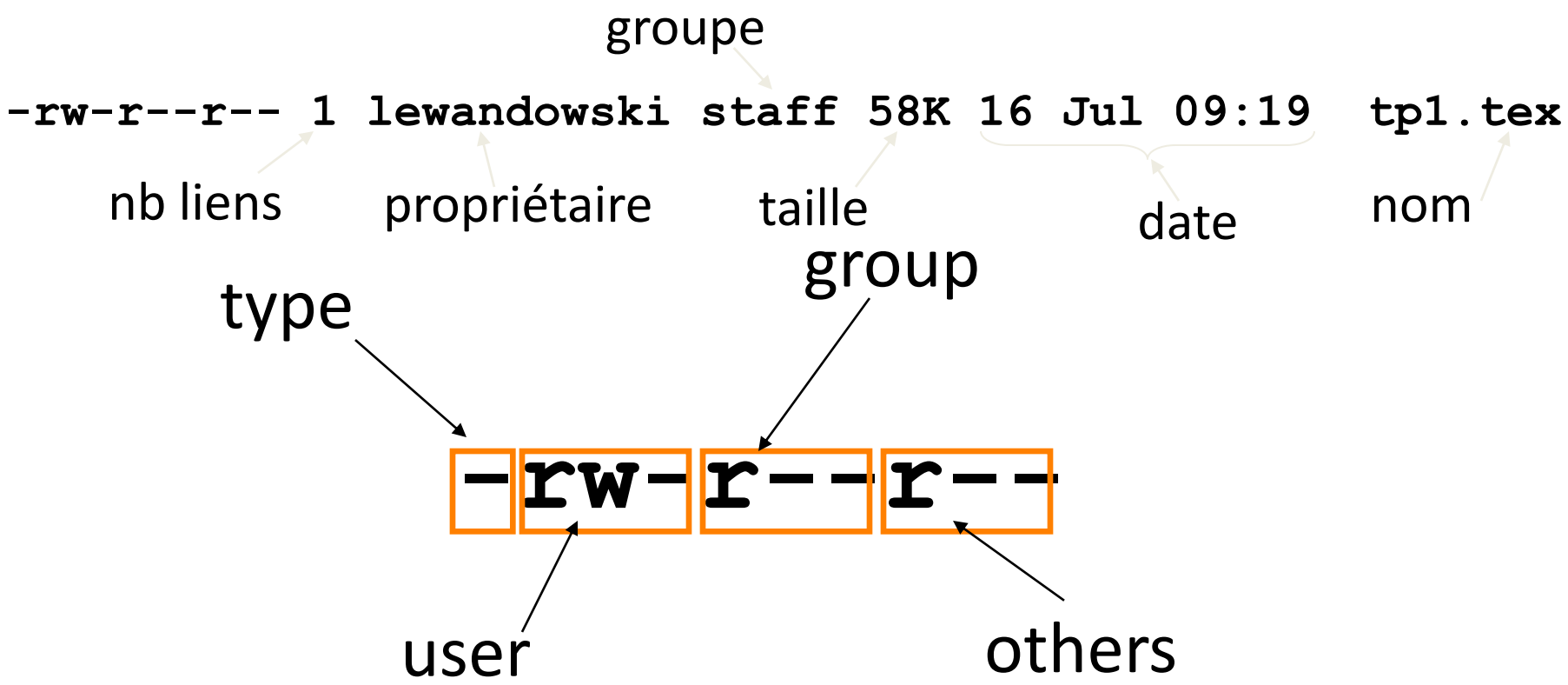
La gestion des droits

- Accès aux fichiers réglementé
  (sauf: tous les droits pour **`root`**)
- 3 types d'utilisateurs:
  - propriétaire (**`user`**)
  - personnes du mm groupe (**`group`**)
  - les autres (**`others`**)
- 3 types de permissions

– lecture (**`r`**)	afficher le contenu	afficher le contenu
– écriture (**`w`**)	modifier	créer/supp fichiers
– exécution (**`x`**)	exécuter	traverser
	fichier	**répertoire**

La gestion des droits

- Affichage des caractéristiques: **ls -l**



La gestion des droits

- Exemple : *d rwx rwx - - - 139 pagnotte profess 352 Nov 25 1999 tp*
  - tp est un répertoire (d)
  - Son propriétaire est *pagnotte*, du groupe *profess*
  - les protections rwx rwx - - - sont à interpréter selon les indications ci-dessus
- *Remarque*
- *Le type du fichier : 'd' pour répertoire, '‘* pour un fichier ordinaire, 'b' ou 'c' pour des fichiers spéciaux (périphériques).

La gestion des droits

- Changer les permissions: **chmod**
  chmod <classe op perm, …>|nnn <fic>
  - classe:
    u : user
    g : group
    o : others
    a : all
  - op:
    = : affectation
    - : suppr.
    + : ajout
  - perm:
    r : lecture
    w : écriture
    x : exécution

  - chaque perm = 1 valeur:

r	4
w	2
x	1
rien	0

  - déf. des permissions (par addition) pour chaque classe

Exemples:

```
chmod u=rwx,g=rx,o=r tp1.tex
chmod a+x script.sh
chmod 755 script.sh
```

La gestion des droits

- chmod mode fichier
  - Mode = utilisateurs/permission
  - Exemple

$chmod u+x fich1

$chmod g-w fich1

$chmod +r fich1

- Mode = chiffres
  - Exemple User Group Other

$ chmod 754 fich1

La commande umask : Modifie le masque des droits de création de fichier. Lorsqu'un programme crée un fichier, il spécifie les droits d'accès. Parmi ceux, certains sont accordés, d'autres refusés, en fonction du masque. Sans argument, donne la valeur actuelle du masque.

-*Syntaxe :* **$ umask [code]**

Pour les répertoires :

droits demandés :	rwx	rwx	rwx	ou encore 777
- masque :	---	-w-	rwx	ou encore 027
droits accordés :	rwx	r-x	---	ou encore 750

Pour les fichiers :

droits demandés :	rw-	rw-	rw-	ou encore 666
- masque :	---	-w-	-w-	ou encore 022
droits accordés :	rwx	r--	r--	ou encore 644

Exemple

```
$ umask
000
$ mkdir foo
$ ls -ld foo
drwxrwxrwx  2  c1  cours  512  Jul 3 16:39  foo
$ rmdir foo
$ umask 022
$ mkdir foo
$ ls -ld foo
drwxr-xr-x  2  c1  cours  512  Jul 3 16:41  foo
$ rmdir foo
$ umask 077
$ mkdir foo
$ ls -ld foo
drwx------  2  c1  cours  512  Jul 3 16:42  foo
$
```

Gestion des Droits

- chown nouveau_util fichier: Change le propriaitaire du fichier
  - Exemple

  $ chown dubois fich1
- chgrp nouveau_grp fichier : Change le groupe du fichier
  - Exemple

  $ chgrp telecom fich1

Permissions : le super-utilisateur

- Afin de permettre l'administration du système, un utilisateur spécial, nommé super utilisateur (ou root), est toujours considéré par le système comme propriétaire de tous les fichiers
- La personne qui gère le système est normalement la seule à connaître son mot de passe. Lui seul peut ajouter de nouveaux utilisateurs au système.

- who
  - Affiche les informations sur les utilisateurs connectés
- who am i
  - Affiche les informations de l'utilisateur courant
- whoami
  - Affiche le login de l'utilisateur courant
- id
  - Affiche l'UID et le GID de l'utilisateur courant

Les mécanismes de redirection et de tube

Les entrées et les sorties

- Il y a trois sortes d'entrées sorties ou flux de données :
  - le premier est l'entrée standard, c'est à dire ce que vous saisissez au clavier,
  - le deuxième est la sortie standard, c'est à dire l'écran, plus précisément le shell,
  - et le troisième est la sortie standard des messages d'erreurs consécutifs à une commande, qui est généralement l'écran.
- Chacun de ces flux de données est identifié par un numéro descripteur, 0 pour l'entrée standard, 1 pour la sortie standard et 2 pour la sortie standard des messages d'erreur.

Les fichiers standard et la redirections d ’E/S

- **Redirection de la sortie standard:**
  - *Remarque:*
    - Pour éviter d’écraser le contenu d’un fichier suite à une redirection de la sortie, on peut utiliser la redirection avec ajout. Dans ce cas le résultat de la commande sera inséré à la fin du fichier.
  - *Syntaxe:* $ Commande >> fichier.
  - *Exemple:* **$ date >> connect**

Les fichiers standard et la redirections d ’E/S

- **Redirection de l’Entrée standard:**
  - Un fichier peut servir comme entrée standard à une commande :
  - Le fichier source contient les arguments de la commande.
  - *Syntaxe :* $ Commande < fichier_source
  - *Exemple :*
    - $ wc -l < connect
    - Permet de compter le nombre de lignes dans le fichier connect.

Les fichiers standard et la redirections d ’E/S

- **Redirection de la sortie erreur standard :**
  - Chaque programme est doté d’un canal de sortie d’erreur séparé dont le descripteur de fichier égal à 2.
  - *Exemple:* $ cc programme.c 2>erreurs.
  - Les erreurs de compilation du fichier programme.c seront redirigées vers le fichier erreurs.
  - On peut utiliser également le fichier /dev/null pour la redirection de la sortie erreur.

Les mécanismes de redirection et de tube

- Redirection des E/S

> : la sortie standard est redirigée vers un fichier (ecrasement de son contenu s'il existe déjà)

< : les entrées de la commande proviennent d'un fichier

>> : la sortie standard est insérée à la fin d'un fichier

2> : la sortie d'erreur est redirigée vers un fichier

2>> : la sortie d'erreur est insérée à la fin d'un fichier

Les tubes de communication

- Définition:
  - Lier les entrées et les sorties de plusieurs commandes dans une même ligne de commande.
- *Syntaxe:* $ Commande1 | Commande2
- Le résultat de la commande1 sera considéré comme argument pour la commande2.
- |: indique un tube.

Les tubes de communication

- *Exemple:*

  $ who | wc - l

  $who : liste de personnes connectés au système.

  $wc -l nom_fichier :Compte le nombre de lignes de nom_fichier.

Les tubes de communication

- **La commande tee :**
- **tee** *[-a] filename* l'affichage de la sortie standard est en même temps dirigé sur *filename.* L'option **-a** signifie >>
  - *Syntaxe:*
    - $ commande1 | tee fichier1 | commande2
  - Redirige le résultat intermédiaire de commande1 vers fichier1. Ce même résultat sera traité par la commande commande2.

Les tubes de communication

- **La commande tee:**
  - *Exemple:*

    $ ls | grep poème | tee fichier1 | wc -l.

    1 2 3 4

    - 1: Liste des fichiers dans le répertoire courant
    - 2: Recherche des noms de fichiers qui contiennent la chaîne de caractères poème.
    - 3: Met le résultat de la commande précédente dans fichier1
    - 4: compte le nombre de lignes ramenés par grep.

Les Filtres

Les Filtres

- *Les commandes ayant la propriété à la fois de lire sur leur entrée standard et* d'écrire sur leur sortie standard sont appelées des filtres. Les commandes cat, wc, sort, grep, cut, tail, head, tr, …. sont des filtres

La commande grep

- Permet de rechercher un certain motif dans un fichier.
- Le motif est décrit par une *expression régulière*.
- ***grep [option] motif fichier*** Affiche les lignes de fichier qui contiennent le motif *motif*.
- -l: n'affiche que le nom des fichiers.
- Exemple:
  - **grep 'define' stdio.h:** recherche le mot "define" dans le fichier stdio.h.
  - **grep 'hello' *:** recherche le mot "hello" dans tous les fichiers du répertoire.

La commande sort

- **sort** *[options] [+n1 -n2] filename1 [-o filename2]*
  trie, selon l'ordre lexicographique du code, les lignes de *filename1*, affiche le résultat ou le redirige sur *filename2*.

Options de la commande sort

- **-b** on ignore les espaces de tête
- **-d** seuls les chiffres, lettres et espaces sont significatifs dans les comparaisons,
- **-f** majuscules et minuscules sont confondues,
- **-i** les caractères dont le code ASCII est extérieur à l'intervalle [32,126] sont ignorés dans les comparaisons,
- **-n** les débuts de lignes numériques sont triés numériquement,
- **-t***c* définit comme *c* le séparateur de champs au lieu de TAB

La commande cut

- Cette commande extrait des colonnes (option -c) ou des champs (option -f) des lignes d'un fichier ou de l'entrée standard. Dans le cas de l'option -f, il est possible de lui spécifier le délimiteur à chercher en utilisant l'option -d. Le délimiteur par défaut est la tabulation

Exemple

Commande	Action
$ cut -f3,7 -d : /etc/passwd	filtre les champs 3 et 7 de chaque ligne de *passwd* en considérant le caractère : comme délimiteur
$ date | cut -c1-3	filtre les caractères 1 à 3

La commande tr

- **tr** *string1 string2* l'entrée standard est copiée sur la sortie standard, mais un caractère ayant une occurence dans *string1* est remplacé par le caractère de même rang dans *string2*. Avec l'option **-d**, les caractères en entrée, présents dans *string1*, sont supprimés en sortie.

Les outils

La commande find

- Permet de chercher dans un répertoire et ses sous-répertoires des fichiers présentant certaines caractéristiques.
- Syntaxe: ***find chemin expression***
- Principales options:
  - -name fich: recherche le fichier fich.
  - -print: écrit le nom du fichier.
  - -type: d: répertoire, f: fichier symbolique.
  - -exec: permet l'exécution d'une commande sur le fichier représenté par {}.

Exemple avec find

- Rechercher tous les fichiers nommés **hello.txt** à partir du répertoire racine.

find / -name hello.txt -print.

- Afficher tous les fichiers **.h** à partir du répertoire courant.

find . -name '*.h' -print

- Affiche à l'écran le contenu de tous les fichiers **.c**.

find . -name '*.c' -exec cat '{}' \;

La comparaison de fichiers

- La commande diff donne les modifications à apporter au premier fichier spécifié pour qu'il ait le même contenu que le second.

Par exemple :

- diff pass.tmp /etc/passwd affichera les modifications à apporter au fichier pass.tmp pour qu'il ait le même contenu que le fichier /etc/passwd.

La comparaison de fichiers

- touch *fichier :* Modifie la date de dernière modification du fichier, celleci devient égale à la date à laquelle la commande a été exécutée. Si le fichier n'existe pas, il sera créé (et de taille nulle) sauf si l'option c est utilisée.

La comparaison de fichiers

- **La commande cmp**
- **cmp** *nom_fichier1 nom_fichier2:* donne le n° de l'octet et de la ligne où se produit la première différence entre *nom_fichier1* et *nom_fichier*.

Expressions régulières

Expressions régulières

- Motif ou pattern
  - Formulation des critères d'identification de chaîne de caractères
- Le motif peut être
  - Simple : une suite de caractères
  - Exemple : "shell"
  - Exprimé par des caractères spéciaux appelés métacaractères :
    - ^ $ | * ...
  - Analogie avec les astérisques *,? du dos
    - *.exe : désigne tous les fichiers ayant l'extension .exe
  - Combinaison des deux

Expressions régulières

- Objectifs
  - Identifier la (les) chaîne(s) de caractères répondant à un certain nombre de critères
  - Exécuter des commandes sommaires Une commande peut remplacer plusieurs commandes

Expressions régulières

- Principales Fonctionnalités
  - Vérification de l'existence d'une séquence de caractères dans une ligne
  - Remplacement d'une séquence par une chaîne de caractère
  - Suppression d'une séquence de caractères
  - Extraction d'une séquence de caractères

Principaux métacaractères

Métacarctères	Signification
*	Répétition 0 ou plusieurs fois
+	Répetition au moins une fois
?	Répétition 0 ou 1 fois
{n}	Répétition exactement n fois
{m,n}	Répétition entre m et n fois
{n,}	Répétition au moins n fois
{,n}	Répétition au plus n fois

Principaux métacaractères

Métacarctères	Signification
.	N'importe quel caractère(sauf \n)
[]	Un des caractères entre les crochets
^	Début de ligne ou négation entre []
$	Fin de ligne
-	Intervalle de ... à entre []
|	Le choix
()	Sous motif

Expressions régulières

- Méta-caractères

  Ces caractères ont une signification particulière et par conséquent ne peuvent être utilisés directement pour une recherche les concernant. Ils doivent donc être précédés par le caractère d'échappement '\'.

  Exemple

  script\.sh

Expressions régulières
Méta-caractères

- **.** **:** Remplace n'importe quel caractère
- [] : Regroupe l'ensemble ou l'intervalle de valeurs que peut prendre un caractère
- Exemples
  - [ACGT] : l'un des quatre caractères
  - [a-z] : n'importe quelle lettre minuscule
  - [a-zA-Z] : n'importe quelle lettre
  - [0-9] : n'importe quel chiffre
- * : Exprime la répétition d'un caractère ou d'un motif
  - Exemples
    - A* : chaîne vide ou contenant plusieurs A (A, AA, AAA, …)
    - [a-z]* : séquence de caractères minuscules
    - [1-9][0-9]* : entier nature

Expressions régulières
Méta-caractères

- ^
  - Début de ligne
  - Exemples
    - ^[A-Z] : La ligne commence par une lettre majuscule
    - ^Bonjour : la ligne commence par « Bonjour »
  - Négation : entre []
  - Exemples
    - [^a-z] : n'est pas un caractère minuscule
- $
  - Fin de ligne
  - Exemples
    - [0-9]$: la ligne se termine par un chiffre
    - (valide)$: la ligne se termine par le mot « valide »
    - \.$: la ligne se termine par '.'

Expressions régulières
Méta-caractères

Caractères spéciaux usuels

Caractère	Description
\n	Saut de ligne
\r	Retour chariot
\t	Tabulation
\f	Saut de page
\e	Echappement

Expressions régulières
Méta-caractères

Motifs prédéfinis

Caractère	Description
\d	Un chiffre : [0-9]
\D	Tout sauf un chiffre : [^0-9]
\w	Un caractère alphanumérique : [0-9a-zA-Z]
\W	Tout sauf Un caractère alphanumérique : [^0-9a-zA-Z]
\s	Un espacement
\S	Tout sauf espacement

L’éditeurs de texte vi

Introduction

- LINUX permet de distinguer les "visualisateurs " de fichiers (afficheur de contenu pour la consultation), des éditeurs de fichiers en texte brut (pour la modification sans mise en page), des formateurs de texte qui permettent de mettre en forme un texte, des filtres et des traitements de texte qui sont plus sophistiqués.
- Il existe de nombreux éditeurs de texte: **ed**
- **joe**
- **vi** (visual)
- **vim** (vi improve)
- **pico**
- **emacs** (editor macros)

- **L'éditeur "vi"** est le premier éditeur "plein écran" d'UNIX. "vi" est rapide mais difficile. L'éditeur "vi" fonctionne dans plusieurs "modes". La version améliorée pour LINUX " **vim**" facilite son apprentissage.

- Pour ouvrir un fichier existant ou le créer:
  vi premier.txt
  Pour passer en mode insertion:
  Saisir "i" Active le mode insertion
  Saisir "a" Active le mode insertion mais un caractère après le curseur
  Pour revenir au mode "commande":
  ECHAP

- Pour quitter "vi" sans enregistrer:
  Saisir ":q! "
- Pour quitter "vi" en enregistrant:
  Saisir "ZZ"
  ECHAP + ":wq!"
- Pour afficher l'aide en ligne:
  :help

Sauvegarde d'un fichier / Sortie de vi

- :w Sauvegarde le fichier
- :e! Ignore les changements et recharge le fichier
- :q! Force vi à se terminer
- :w fichier Sauvegarde le fichier sous le nom "fichier"

- Voir d’autres commandes au TP

- *Déplacements dans vi*
- [CTRL]f Descend d'un écran
- [CTRL]b Remonte d'un écran
- [CTRL]d Descent d'1/2 écran
- [CTRL]u Remonte d'1/2 écran
- :xxx Va à la ligne xxx

- *Réactualisation de l'écran*
- [CTRL]l Réactualise l'écran
- *Positionnement du curseur*
- H En haut de l'écran
- M Au milieu de l'écran

- L En bas de l'écran
- h Décale d'un caractère à gauche
- j Descend d'une ligne
- k Monte d'une ligne
- l Décale d'un caractère à droite
- O Au début de la ligne
- $ À la fin de la ligne
- w Au début du mot suivant
- e À la fin du mot suivant
- b Recule d'un mot

- *Insertion d'un texte*
- I Insère au début de la ligne
- A Insère à la fin de la ligne
- [ESC] Termine le mode insertion

- *Insertion et remplacement d'un texte*
- r* Remplace le caractère à la position du curseur par *
- R Remplace tous les caractères jusqu'à la fin de la ligne ([ESC] pour terminer)
- cw Remplace uniquement le mot à la position du curseur ([ESC] pour terminer)
- cnw Remplace n mots ([ESC] pour terminer)
- C Change le reste de la ligne ([ESC] pour terminer)

- *Suppression d'un texte*
- x Supprime un caractère
- dw Supprime un mot
- dnw Supprime n mots
- dd Supprime une ligne
- ndd Supprime n lignes
- *Copier/Coller*
- Y Copie une ligne
- nY Copie n lignes
- P Colle les lignes avant le curseur
- p Colle les lignes après le curseur

- *Copier/Coller du texte dans un autre fichier*
- ma Marque la position "a" dans le fichier
- mb Marque la position "b" dans le fichier
- :'a,'b fichier Copie le texte de la position "a" à la position "b" dans le fichier "fichier"
- *Insertion du texte d'un autre fichier*
- :r fichier Insère tout le contenu du fichier "fichier"

- *Rechercher/Remplacer du texte*
- /chaine Recherche le texte "chaine" vers le bas
- ?chaine Recherche le texte "chaine" vers le haut
- n Répète la dernière recherche
- N Inverse la recherche précédente
- :g/expr1/s//expr2/g Recherche dans tous le fichier l'expression 1 et la remplace par l'expression 2
- :x,y s/expr1/expr2/g Recherche de la ligne x à la ligne y l'expression 1 et la remplace par l'expression 2

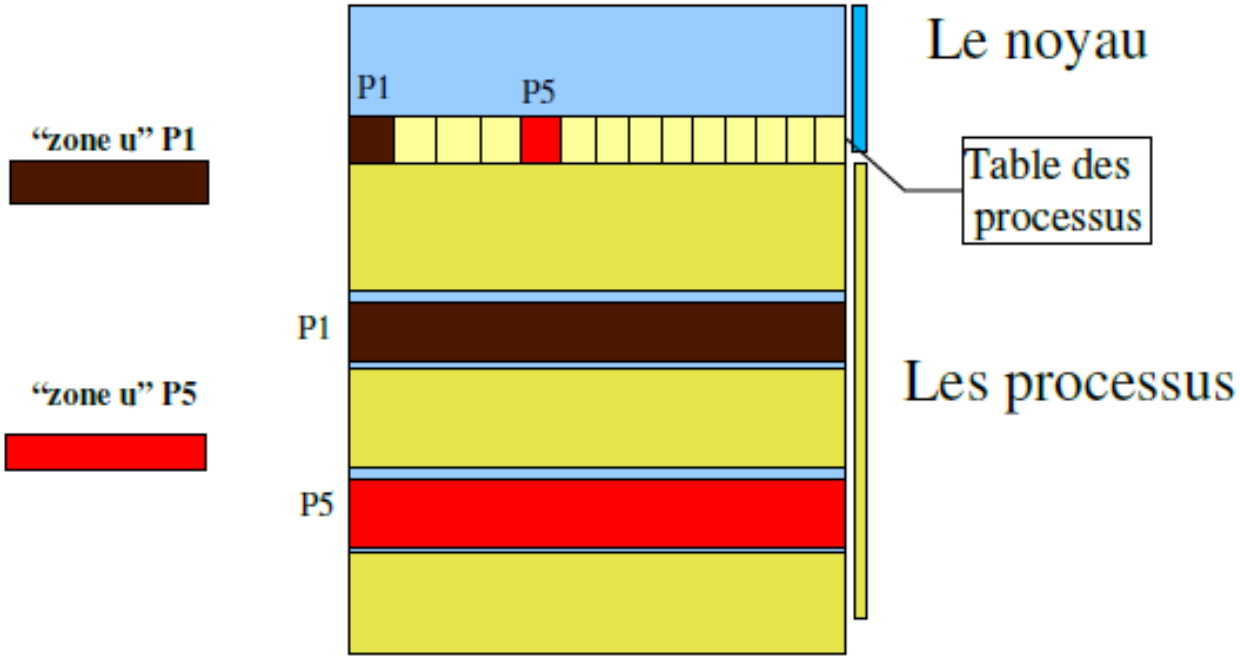
- *Annuler/Répéter les modifications*
- u Annule la dernière modification
- U Annule toutes les modifications effectuées sur la ligne courante
- . Répète les dernières modifications

gestion des processus

- **<u>gestion des processus :</u>** contrôle de la création, de la terminaison, de la synchronisation, du partage de temps (ordonnancement), de la communication entre processus,
- Les processus: Un processus est un programme qui s'exécute, ainsi que ses données, sa pile, son compteur ordinal, son pointeur de pile et les autres contenus de registres nécessaires à son exécution.

- Le noyau maintient une table, appelée « table des processus », pour gérer l'ensemble des processus (ici P1, ..., P5, ...).

  Cette table, interne au noyau, contient la liste de tous les processus avec des informations concernant chaque processus. C'est un tableau de structure « proc » (<sys/proc.h>).



Le nombre des emplacements dans cette table des processus est limité pour chaque système et pour chaque utilisateur.

- Le noyau alloue pour chaque processus une structure appelée « zone u » (<sys/user.h>), qui contient des données privées du processus, uniquement manipulables par le noyau.
- Seule la « zone u » du processus courant est manipulable par le noyau, les autres sont inaccessibles.
- L'adresse de la « zone u » d'un processus est placée dans son mot d'état.
- Le noyau dispose donc d'un tableau de structures (« proc.h ») dans la table des processus et d'un ensemble de structures (« user.h »), une par processus, pour piloter les processus.

- Le contexte d'un processus est l'ensemble des données qui permettent de reprendre l'exécution d'un processus qui a été interrompu:
  - son état (élu, prêt, bloqué, ...)
  - son mot d'état : en particulier
    - la valeur des registres actifs
    - le compteur ordinal
  - les valeurs des variables globales statiques ou dynamiques
  - son entrée dans la table des processus
  - sa « zone u »
  - les piles « user » et « system »
  - les zones de code (texte) et de données

Etats d'un processus

- **Prêt**

  - Le processus est prêt pour l'exécution. Il détient toutes les ressources, sauf le CPU.
  - Plusieurs processus peuvent être à l'état prêt.

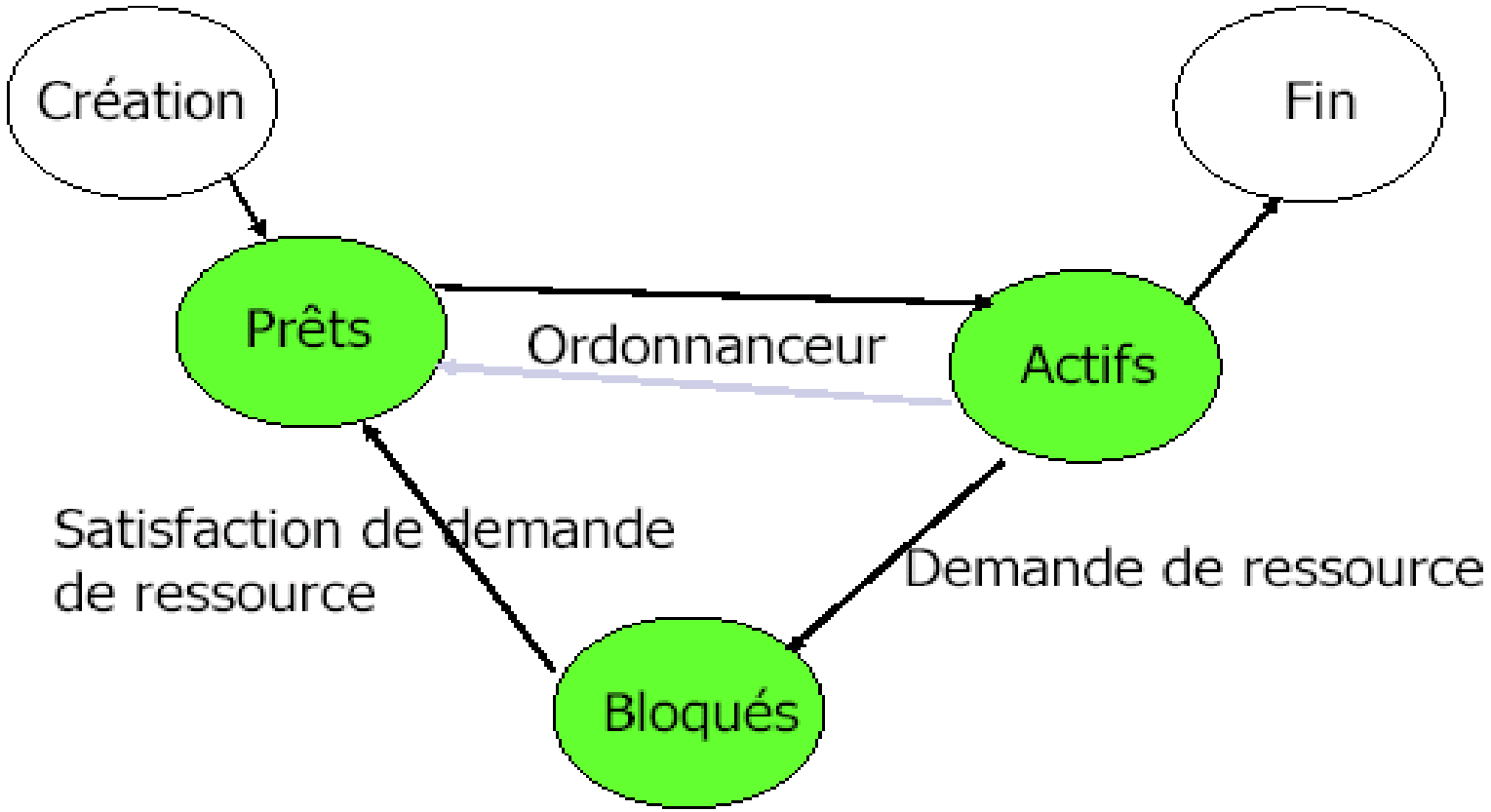
- **Actif**

  - Détient la ressource CPU.
  - Le processus «évolue».
  - Nombre de processus actifs inférieur ou égal au nombre processeurs.

- **Bloqué**

  - Le processus est bloqué sur une demande de ressource : E/S, mémoire centrale, etc.

Etats d'un processus



- L'exécution d'un processus se fait dans son contexte.

- Parmi les informations propres à chaque processus, qui sont contenues dans les structures système (« proc.h » et « user.h ») , on trouve :
  - un numéro d'identification unique appelé PID (Process Identifier), ainsi que celui de son père appelé PPID
  - le numéro d'identification de l'utilisateur qui a lancé ce processus, appelé UID (User IDentifier), et le numéro du groupe auquel appartient cet utilisateur, appelé GID (Group IDentifier) ;
  - le répertoire courant ;
  - les fichiers ouverts par ce processus ;
  - le masque de création de fichier, appelé *umask ;*

– la taille maximale des fichiers que ce processus peut créer, appelée *ulimit ;*
– la priorité ;
– les temps d'exécution ;
– le *terminal de contrôle, c'estàdire* le terminal à partir duquel la commande a été lancée.

Certaines des caractéristiques de l'environnement peuvent être consultées par

diverses commandes. Nous connaissons déjà :

- pwd affiche le chemin du répertoire courant
- tty affiche le terminal de contrôle
- umask affiche le masque de création de fichier
- id consulte l'UID et le GID.
- Exemple:
- $ id
- uid=106(c1) gid=104(cours)
- $

Création de processus

- Pour chaque commande lancée (sauf les commandes internes), le shell crée automatiquement un nouveau processus.
- Il y a donc 2 processus. Le premier, appelé processus *père, exécute le* programme shell, et le deuxième, appelé processus *fils, exécute la commande.*
- Le fils *hérite de tout l'environnement du* père, sauf bien sûr du PID, du PPID et des temps d'exécution.

Pour visualiser les processus que vous avez lancé, tapez la commande «ps» :

- ***La commande ps***
- ***Affiche les informations des processus actifs***
- ***Options***
  - -l : affiche les informations complètes des processus
  - -x : affiche tous les processus actifs (d'autres utilisateurs)
  - -u : affiche les processus d'un utilisateur donné
- Exemple
- $ ps -l
- $ ps -u dupont

- Exemple:

```
$ ps
PID    TTY    TIME    COMMAND
527    ttyp   4 1:70   ksh
536    ttyp   4 0:30   cmd1
559    ttyp  4 0:00    ps
$
```

- PID identifie le processus,
- TTY est le numéro du terminal associé,
- TIME est le temps cumulé d'exécution du processus,
- COMMAND est le nom du fichier correspondant au programme exécuté par le processus.

- Il existe bien d'autres commandes pour gérer les processus, comme par exemple la commande « top ».
- top : cette commande affiche les processus qui consomment le plus de ressources systèmes. Dans les premières lignes, elle affiche des informations globales sur le système (charge, mémoire, nombre de processus, ...).

- Lorsqu'un processus se termine, il retourne toujours une *valeur significative ou statut.*
- Par convention, lorsqu'un processus se termine correctement, il retourne la valeur 0, sinon il retourne une valeur différente de 0 (généralement 1). Ce choix permet de ramener des codes significatifs pour différencier les erreurs.

Le statut d'une commande shell est placé dans la pseudo variable spéciale, nommée « ? ». On peut consulter sa valeur en tapant la commande :

```
$ echo $?
```

Le shell

- Le shell (littéralement coquille autour du noyau d'UNIX) est l'interpréteur de commandes d'UNIX. Tout à la fois :

  - il exécute en mode interactif les commandes émises par l'utilisateur,

  - il propose un langage de programmation interprété permettant de générer de nouvelles commandes ou procédures cataloguées ("scripts shell"), C étant le langage le plus adapté pour construire les nouvelles commandes que le shell ne peut traduire.

- Le shell ne fait pas partie du noyau d'UNIX et n'est pas résident en mémoire principale. Ainsi, on peut disposer facilement de plusieurs interpréteurs de commandes : Bourne-shell, C-shell, Korn-shell, ...

Bourne shell

- **« Bourne shell » la syntaxe des commandes est** proche de celle des premiers UNIX (/bin/sh).
- Il existe plusieurs interpréteurs de commandes. Historiquement, le premier a été écrit par S. R. Bourne. Il est donc souvent nommé Bourne Shell.

Les variables

- leur ***nom*** : une suite de caractères lettres, chiffres
- exemple:

a=paul

chemin=/users/eleves/m-durand99

- leur ***valeur*** : $a ou ${a} désigne la valeur de la variable a et ${a}c désigne la valeur de a suivie de c.
- exemple:

a=paul

b=chou

echo $a $b

On utilise trois ***caractères génériques*** :

- * toute sous-chaîne, même vide,
- ? tout caractère,
- [...] tous les caractères d'un intervalle.
- Toute fin de ligne commençant par # est un commentaire

- ***métacaractères*** : < * ? | & , \ ont un sens spécial.
- ex:

a="bijou * caillou "

b=chou ; c=caillou ; r="$a $b";echo $r

-

  Précédés de \, les métacaractères perdent leur signification particulière

-

- ex.:
- echo * ; echo \\
- echo abc***d
- les ***délimiteurs de chaînes*** :
- dans une chaîne délimitée par des " , les caractères $, \, ', ` sont des caractères spéciaux.
- dans une telle chaîne, un caractère doit être précédé de \
- dans une chaîne délimitée par des ' , tous les caractères perdent leur aspect spécial

Propriétés des variables

- Identificateur
  - Nom composé de caractères
  - Certains caractères sont interdits ($,#,...)
- Types
  - Numérique
  - Chaîne de caractères
- Accès au contenu
  - Précéder l'identificateur par le caractère $

Variables d'environnement

- SHELL : le shell utilisé
- USER : nom de l'utilisateur
- UID : identificateur de l'utilisateur
- PATH : chemin des répertoires contenant les fichiers exécutables
- HOME : chemin du répertoire d'accueil
- PWD : chemin du répertoire courant
- HOSTNAME : nom de la machine

Lecture

- Lecture (read)
- read permet de lire une ou plusieurs variables à partir de l'entrée Standard
- Syntaxe

read var1 [var2, …]

Si plusieurs variables à la fois, le contenu saisi sera réparti sur les variables, dans l'ordre, avec espace comme séparateur.

- Exemples
- read a
- read n a

Affectation

- Affectation : =
- Syntaxe

Ident_variable = <expression>

Exemples

n=10

Nom="Mohamed"

Les structures de contrôle

- Les structures de contrôle permettent de transformer une simple procédure en un programme qui pourra comparer, tester ...

La structure for

- Cette structure de contrôle permet d'exécuter un ensemble de mêmes commandes avec un ensemble d'éléments.

for *nom [in liste ...]*

do

commandes

done

- ***nom*** *est le nom d'une variable, qui prendra* successivement la valeur de chaque mot de la liste exhaustive fournie après **in. L'exécution se termine** lorsqu'il n'y a plus d'élément dans la liste.

La structure if

- Cette construction peut être utilisée comme instruction de branchement générale. Il s'agit d'un aiguillage.

if ***commandes1***

Then ***commandes2***

else ***commandes3***

fi

- La ***commande1 est évaluée. Si elle est vraie (code de*** retour nul), ***commande2 est évaluée à son tour (et commande3 ne le sera pas) et si elle est fausse (code de*** retour non nul), ***commande3 est évaluée (commande2 ne*** l'étant pas). La partie **else de cette instruction est** optionnelle.

La structure case

- Cette structure de contrôle permet de sélectionner des actions suivant la valeur de certains mots. La structure *case ne peut que comparer des chaînes de caractères.*

case *chaîne in*

motif1) **commande1 ;;**

motif2) **commande2 ;;**

...

motifn) **commanden ;;**

esac

Les différents ***motifi sont des expressions reconnues par le*** mécanisme d'expansion des noms de fichiers. De plus, le caractère "**|", lorsqu'il est utilisé dans un motif, permet** l'union des deux expressions entre lesquelles il est placé.

L'interpréteur recherche le plus petit entier ***i inférieur*** à ***n tel que chaîne réponde au motifi. Il exécute alors la commandei (et elle seule). Il est courant d'utiliser * comme*** dernier motif (cas par défaut) d'un aiguillage.

- La structure de contrôle ***while (itération non bornée)*** est certainement la structure que l'on retrouve le plus dans tous les langages de programmation

while ***commandes1***

do ***commandes2***

done

- Dans le cas du ***while, tant que le statut de commandes1 est vrai, commandes2 est exécutée. commandes1 peut être formée d'un ensemble de*** commandes enchaînées par des pipes (**|**).

- **until *commandes1***
- **do *commandes2***
- **Done**
- La structure de contrôle ***until n'est rien d'autre que le*** test inverse du ***while. On pourrait traduire le while par*** "*tant que" et le* ***until par "jusqu'à ce que".***

Commande expr

- Commande expr : Permet d'exécuter les opérations arithmétiques de base
- syntaxe

`expr var1 op var2`

op : +,-,*,/,%

Exemple

n=$(expr $a + $b)

- e1 + e2 retourne le résultat de l'addition
- e1 - e2 retourne le résultat de la soustraction
- e1 * e2 retourne le résultat de la multiplication
- e1 / e2 retourne le résultat de la division
- e1 % e2 retourne le résultat du modulo

- e1 | e2 si ***e1 est égale à 0 retourne e2 sinon retourne e1***
- e1 & e2 si ni ***e1 ni e2 ne sont égales à 0 retourne e1***
- e1 < e2 retourne 1 si ***e1 est plus petit que e2 sinon 0***
- e1 <= e2 retourne 1 si ***e1 est plus petit ou égal à e2*** sinon 0
- e1 = e2 retourne 1 si ***e1 est égal à e2 sinon 0***
- e1 != e2 retourne 1 si ***e1 est différent de e2 sinon 0***
- e1 > e2 retourne 1 si ***e1 est spérieur à e2 sinon 0***
- e1 >= e2 retourne 1 si ***e1 est*** supérieur ou égale à ***e2*** sinon 0

Exemple

a=3

b=$(expr $a + 5)

La variable "**b" va récupérer le résultat du calcul effectué** entre parenthèses, c'est-à-dire "**8". La variable "a" est** inchangée.

Imbrication de if/else

- Syntaxe :
  if condition1
  then
  liste_commandes1
  elif condition2
  then
  liste_commandes2
  else
  liste_commandes3
  fi

- **test *expression*** ou [*expression]*
- **test évalue *expression et retourne le résultat de cette*** évaluation. **test appelé sans argument retourne *faux.***

  Vous devez utiliser une des deux syntaxes, mais pas les deux en même temps ! Il ne faut pas non plus oublier de mettre les caractères séparateurs (blanc, tabulation ...) entre les caractères [et].

Test sur des fichiers et répertoires

- test -**w** ***fichier:***vrai si ***fichier existe et est autorisé en*** **écriture.**
- test -**r** ***fichier vrai si fichier existe et est autorisé en*** **lecture.**
- test -**x** ***fichier vrai si fichier existe et est éxécutable.***
- test -**d** ***fichier vrai si fichier existe et est un r*****épertoire**
- test -**f** ***fichier vrai si fichier existe et n'est pas un*** **répertoire.**
- test -**s** ***fichier vrai si fichier existe et a une taille*** **non nulle.**

Test sur des chaînes

- test -**z *s1: vrai si la chaîne s1 est vide (a une*** longueur de 0 caractère).
- test -**n *s1 : vrai si la chaîne s1 est non vide.***
- test *s1 = s2 :vrai si les chaînes* ***s1 et s2 sont identiques.***
- test *s1 != s2 :vrai si les chaînes* ***s1 et s2 sont*** **différentes.**
- test *s1 : vrai si la chaîne* ***s1 n'est pas la chaîne*** **nulle.**

Test sur des nombres

- test *n1* ***-eq n2 : vrai si l'entier n1 est égal à l'entier n2.***
- test *n1* ***-ne n2: vrai si l'entier n1 est différent de*** l'entier ***n2.***
- test *n1* ***-gt n2 : vrai si l'entier n1 est supérieur à*** l'entier ***n2.***
- test *n1* ***-lt n2: vrai si l'entier n1 est inférieur à l'entier n2.***
- test *n1* ***-ge n2: vrai si l'entier n1 est supérieur ou égal*** à l'entier ***n2.***
- test *n1* ***-le n2 :vrai si l'entier n1 est inférieur ou égal*** à l'entier ***n2.***

On peut combiner toutes ces primitives avec les opérateurs :

- **! négation**
- **-a ET logique**
- **-o OU logique**
- **(*expression) pour regrouper logiquement*** plusieurs tests.

exec

exec réalise des redirections d'E/S

- exec 1 > std.out # redirige la sortie standard vers le fichier std.out
- exec < fic # redirige l'entrée standard sur le fichier fic.....
- exec < /dev/tty # rétablit l'entrée standard

- ***variables prédéfinies*** gérées automatiquement par le shell :
- **$#** nombre de paramètres d'une commande, ceux-ci étant désignés par $1 à $9 ($0 le nom de la commande elle-même).
- **$*** la liste des paramètres $1 $2 ...
- $$ le numéro du processus en cours (très utile dans la suite)
- **$!** le n° du dernier processus lancé en arrière plan