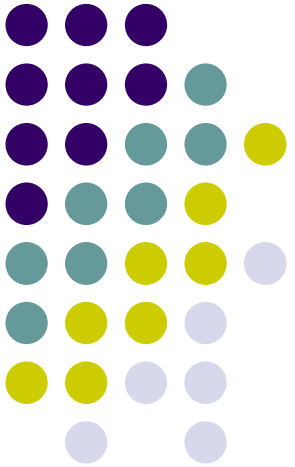
Cours Linux

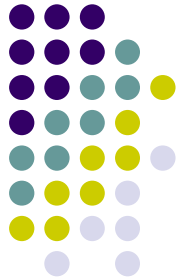
Licence Pro

Université Cheikh Anta Diop

Ould Deye

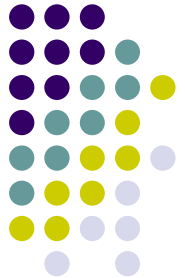


Plan du Cours



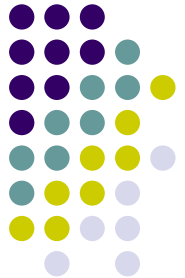
- Présentation et historique
- Ouverture/Fermeture d’une session
- Le système de fichiers
- Commandes de base
- L’éditeur vi
- Les droits d'accès
- Les redirections & pipes
- Contrôle de jobs
- Gestion des processus
- Les filtres
- Les scripts

Présentation et historique



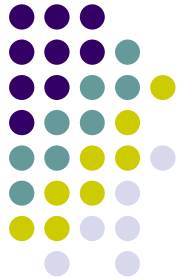
- Un système d'exploitation est l'interface entre l'utilisateur et le matériel. Ses fonctions principales sont :
  - Contrôle des ressources (allocation et gestion du CPU et de la mémoire)
  - Contrôle des processus
  - Contrôle des périphériques
  - …
- Protéger le système et ses usagers de fausses manipulations
- Une base pour le développement et l'exécution de programmes d'application
- Offrir une vue simple, uniforme, et cohérente de la machine et de ses ressources

Présentation et historique



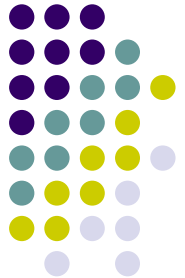
- Linux est une version libre d'UNIX : le code source du système est disponible gratuitement et redistribuable
- Connait actuellement un grand succès, tant chez les utilisateurs particuliers (en tant qu'alternative à Windows) que pour les serveurs Internet/Intranet
- Une distribution Linux comprend le noyau, les pilotes, les bibliothèques,les utilitaires d’installation et de post-installation, ainsi qu’un grand nombre de logiciels
- Les plus répandues sont Red Hat, Suse, Caldera, Debian, Slackware et Mandrake (à l'origine issue de Red Hat), …

Présentation et historique



- **UNIX :**
  - UNIX est un système d'exploitation multi-tâche multi-utilisateurs
  - Multi-tâche préemptif (non coopératif): le système interrompt autoritairement la tâche en cours d'exécution pour passer la main à la suivante
    - Le multitâche coopératif : chaque processus doit explicitement permettre à une autre tâche de s'exécuter.
    - Une tâche peut bloquer l'ensemble du système
  - Multi-utilisateurs : est rendue possible par un mécanisme de *droits d'accès* s'appliquant à toutes les ressources gérées par le système (processus, fichiers, périphériques, etc.)

Présentation et historique

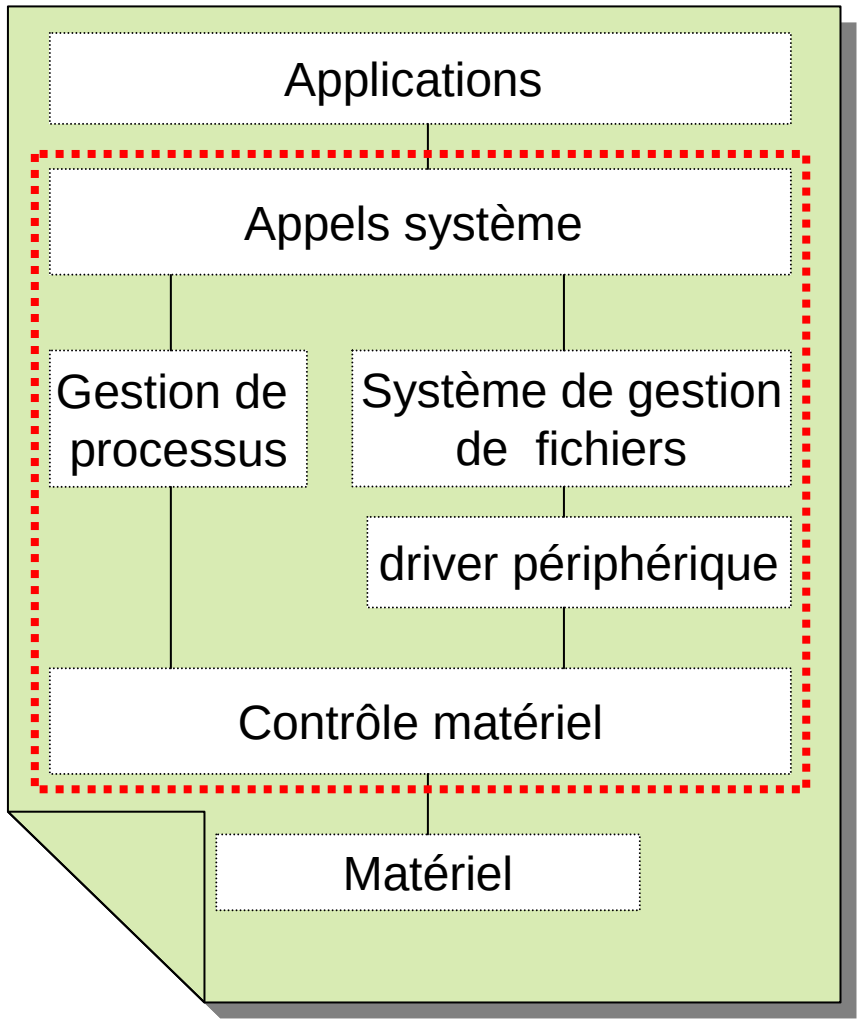


- **Historique :**
  - 1969 aux Bell Labs (laboratoires de recherche en informatique d'A.T.&T.). Écrit en langage C par Ken Thompson et Denis Ritchie (invention de C pour cette occasion et non en assembleur comme il était d'usage de le faire -> grande portabilité)
  - Depuis la fin des années 70, il existe deux grandes familles d'UNIX : UNIX BSD (université de Berkeley (Californie)), UNIX Système V commercialisé par ATT
  - Nombreuses autres versions ont vu le jour, qui sont le plus souvent une adaptation de BSD ou Système V par un fabriquant particulier :
    - AIX IBM, Bull (stations de travail, mainframes)
    - HP/UX Hewlett-Packard (stations)
    - SCO Unix SCO (PC)
    - OSF/1 DEC
    - Solaris Sun MicroSystems (stations Sun et PC)
  - 1991 : GNU/Linux Logiciel libre (et gratuit)

Présentation et historique

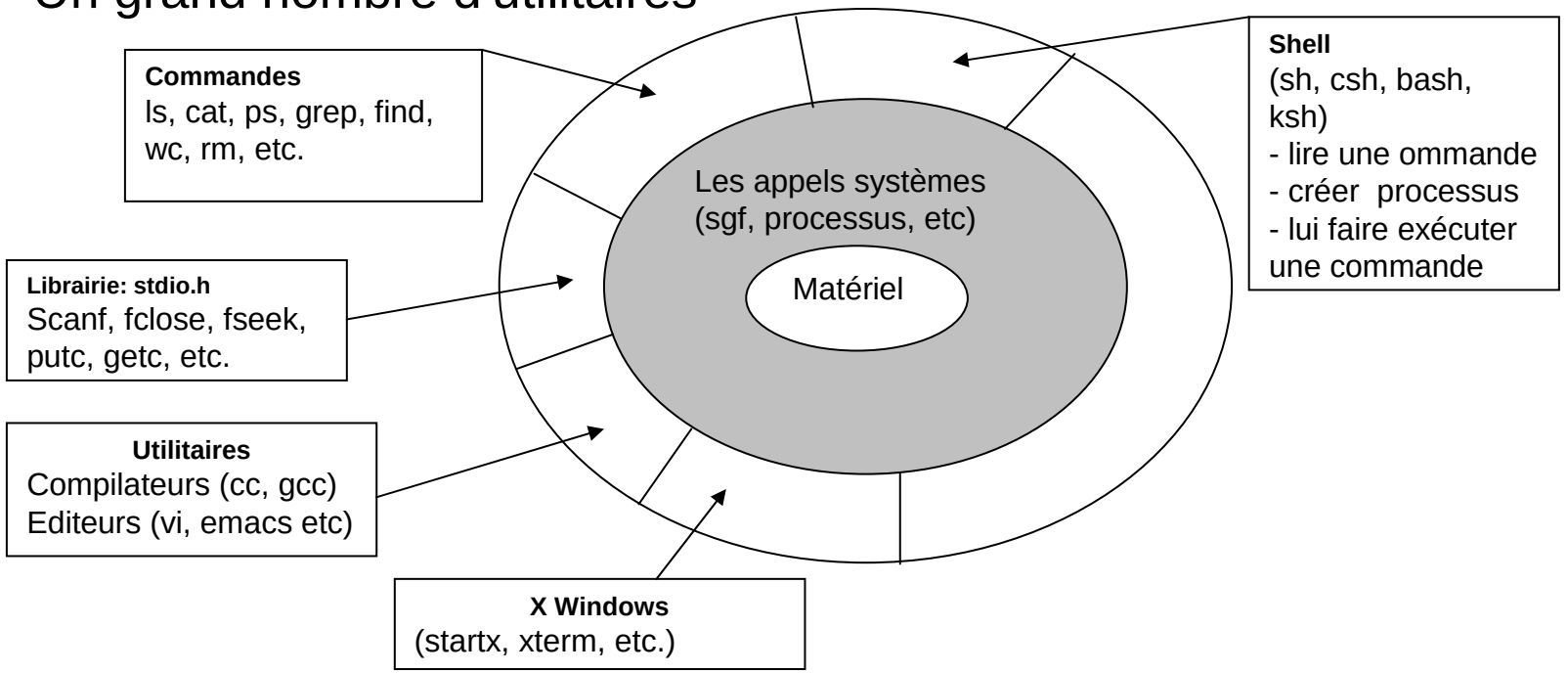
- **Le noyau UNIX :**
  - Le noyau est le programme qui assure la gestion de la mémoire, le partage du processeur entre les différentes tâches à exécuter et les entrées/sorties de bas niveau
  - Il est lancé au démarrage du système (le boot) et s'exécute jusqu'à son arrêt

  - Il est composé :
  - d'un système de gestion de fichiers qui assure l'interface avec les périphériques
  - d'un système de contrôle des processus qui assure l'interface avec l'unité centrale

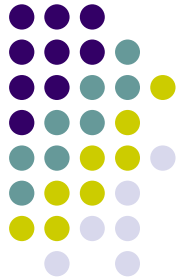


Présentation et historique

- Le kernel sert d'interface au sens large entre le matériel (l'unité centrale et les périphériques) et un environnement système qui comprend :
  - Un ou plusieurs interpréteurs de commandes(shells)
  - Un grand nombre d'utilitaires

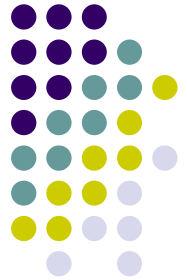


Présentation et historique



- **Shell :**
  - Interface avec l’utilisateur
  - Interprète les commandes de l'utilisateur avant transmission au noyau
  - Couche logicielle bien séparée du noyau
  - Un langage de programmation interprété autorisant la récursivité (shellscripts)

Ouverture/Fermeture d’une session

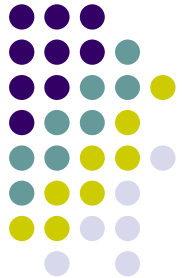


- Travailler sous le système LINUX, même en dehors de tout contexte réseau, implique une connexion au système

- Login:
  - Identification de l’utilisateur: login + mot-de-passe

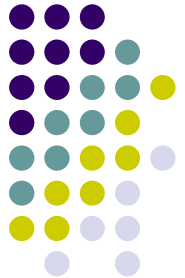
- Après authentification, L'interpréteur de commande par défaut est lancé et a pour répertoire courant le répertoire de connexion de l'utilisateur

Ouverture/Fermeture d'une session



- pwd affiche le répertoire courant
  - pwd [Entrée]
  - /home/ndiaye
- Pour changer de répertoire, utilisez la commande cd
  - cd /usr/local [Entrée]
  - pwd [Entrée]
  - /usr/local
- cd sans arguments, vous permet de revenir à votre répertoire personnel
  - cd [Entrée]
  - pwd [Entrée]
  - /home/ndiaye

Ouverture/Fermeture d'une session



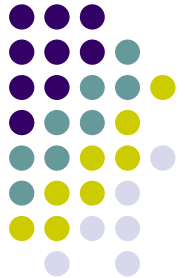
- Pour faciliter les déplacements à travers les répertoires, trois noms de répertoires particuliers sont à retenir:
  - ~ : le répertoire home
  - . : le répertoire courant
  - .. : le répertoire père du répertoire courant

- ls : lister les noms des fichiers
  - ls [Entrée]

- Dans chaque répertoire, on trouve au moins ces deux fichiers : « . » et « .. »

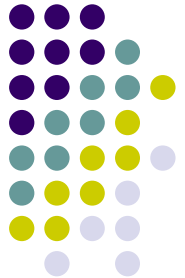
  - cd .. [Entrée]
  - cd ~/test [Entrée] vous conduit au répertoire /home/ndiaye/test

Ouverture/Fermeture d'une session



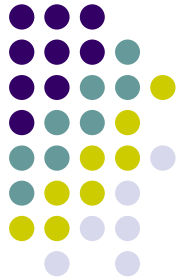
- **L'invite du shell :**
  - [root@markov /root]#
    - Le # indique qu'il s'agit de l'administrateur système
  - [ndiaye@anta /etc]$
    - Le signe $ indique qu'il s'agit d'un utilisateur classique
  - Sa notation symbolique, [\u@\h \W]\$, défini dans /etc/bashrc
    - echo $PS1 [Entrée]
    - \d pour ajouter la date, \t pour ajouter l'heure \w pour ajouter le chemin complet du répertoire courant

Ouverture/Fermeture d'une session



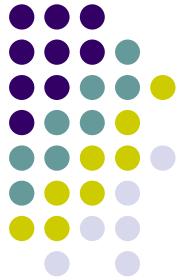
- Pour prendre l'identité d'un autre utilisateur, par exemple moussa :
  - $ su – moussa [Entrée]
- Pour changer votre mot de passe :
  - $ passwd [Entrée]
- Logout:
  - NE PAS ETEINDRE une machine "sauvagement"
  - exit [Entrée]

Commandes de base



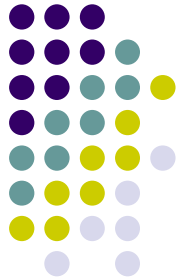
- Il existe plusieurs shells dont les plus utilisés sont :
  - Le Bourne shell, sh /bin/sh : C'est le shell standard d'Unix AT&T
  - Le C-shell, csh, /bin/csh : C'est le shell d'Unix BSD; sa syntaxe rappelle le langage C
  - Le Korn-shell, ksh /bin/ksh : C'est une extension du Bourne shell. On le retrouve maintenant dans la plupart des distributions Unix
  - Le Bash shell (Bourne again shell) /bin/bash : est la version GNU du Bourne-shell. Il incorpore de nombreuses fonctionnalités présentes dans d'autres *shells*, comme le Korn ou le C-shell. C'est le *shell* par défaut de GNU/Linux.
  - Le tcsh (successeur de de csh) /bin/tcsh
- Manipulation : Afficher le shell sur lequel vous travaillez : echo $SHELL

Commandes de base



- **Syntaxe d'une commande**
  - La syntaxe standard d'une commande UNIX est la suivante :
  - Cde [-option(s)] [argument(s)]
  - où Cde indique *ce que doit faire la commande* ; les options précisent *comment le faire* et les paramètres indiquent *sur quoi le faire*
  - Les options varient en fonction de la commande, le nombre des arguments qui suivent dépend aussi de la commande
  - La variable d'environnement PATH propre à chaque user

Commandes de base

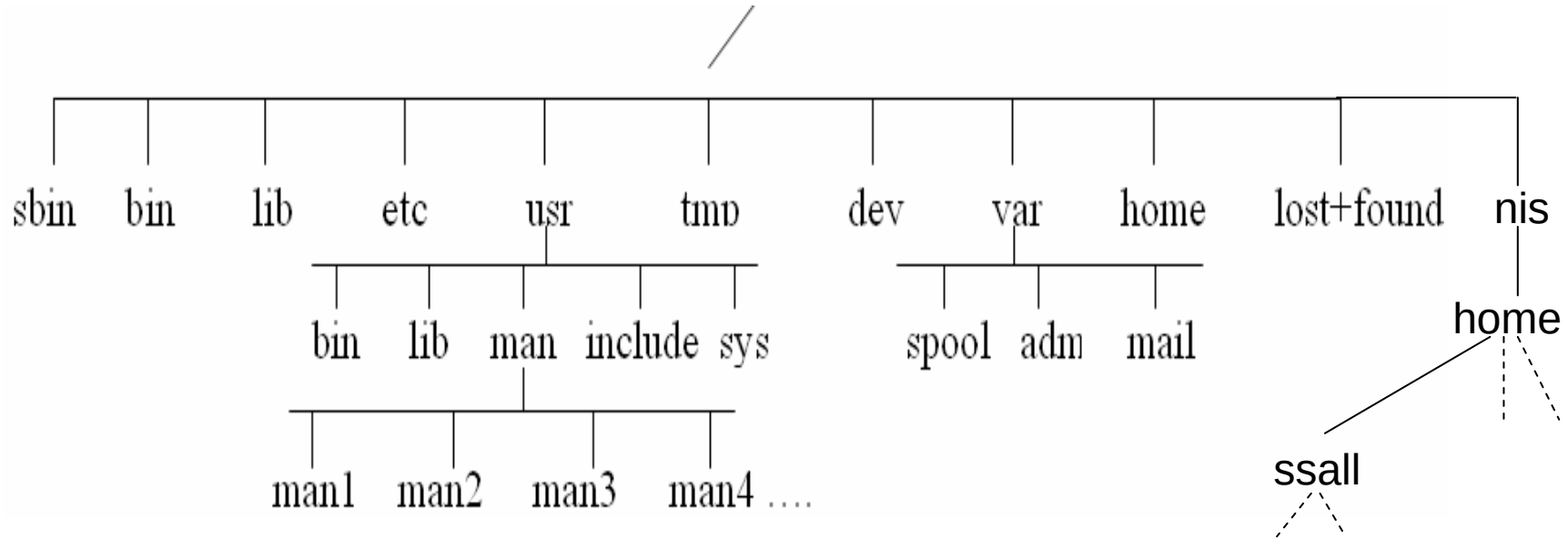


- **Quelques commandes :**
  - Commandes de gestion des répertoires :
    - mkdir nom-de-répertoire Création d'un répertoire
    - rmdir nom-de-répertoire Suppression d'un répertoire vide
    - mv répertoire répertoire-d'accueil déplacement d'un répertoire
    - mv répertoire nouveau-nom Changement de nom d'un répertoire
  - Commandes de gestion des fichiers :
    - touch mon-fichier création d'un fichier vide,
    - more mon-fichier visualisation d'un fichier page à page,
    - rm mon-fichier suppression d'un fichier,
    - mv mon-fichier répertoire d'accueil déplacement d'un fichier,
    - mv mon-fichier nouveau-nom changement de nom d'un fichier,
    - cp nom-fichier répertoire-d'accueil/autre-nom copie de fichier,
    - file mon-fichier permet de connaître la nature d’un fichier mon-fichier

Le système de fichiers

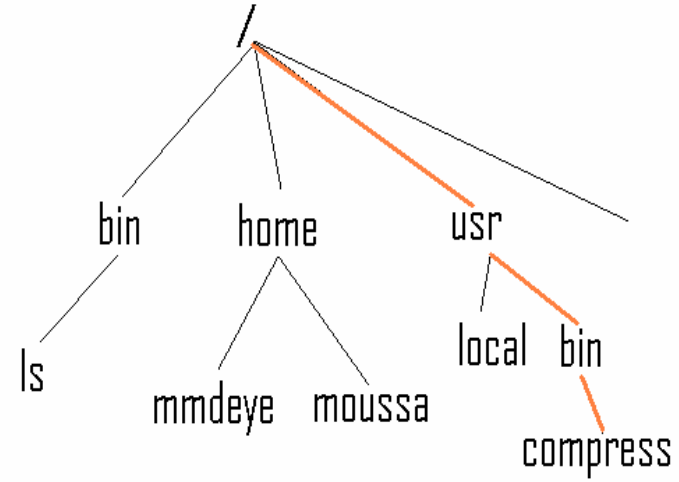
- Sous UNIX TOUT est fichier

- Ces fichiers sont organisés dans une arborescence unique composée:
  - d'une racine (/), des nœuds (les répertoires) et des feuilles (fichiers)



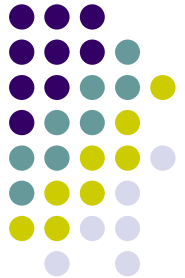
Le système de fichiers

- **Chemins absolus et relatifs :**
  - Un chemin absolu spécifie la suite des répertoires à traverser en partant de la racine, séparés par des caractères « / »
  - Par exemple, le chemin /usr/bin/compress



- Tout chemin qui ne commence pas par un slash « / » est interprété comme un chemin relatif au répertoire courant
- Par exemple, a partir de « /home/moussa », ../../usr/bin/compress

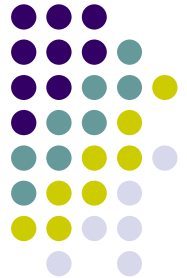
Commandes de base



- ls : lister les noms des fichiers
  - options:
    - -a : tous les fichiers, même cachés
    - -F : les répertoires sont signalés par ’/’ et les fichiers exécutables par ’*’
    - -l : listing détaillé avec type et taille des fichiers
    - -R : affiche récursivement le contenu des sous-répertoires

- Manipulation : Afficher tous les fichiers (y compris ceux des sous-répertoires) de votre répertoire de travail

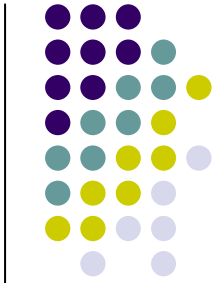
Commandes de base



- Avec la commande ls -l on peut déterminer le type de fichier :
  - d : répertoire
  - - : fichier normal
  - l : lien symbolique
  - b : fichier de type bloc
  - c : fichier de type caractère

```
[mmdeye@mmdeye ~/licpro]$ ls -l
total 8
lrwxrwxrwx    1 mmdeye   mmdeye          2 nov 22 20:15 autrestp -> tp
lrwxrwxrwx    1 mmdeye   mmdeye          7 nov 22 20:20 deuxtp -> tp1.pdf
lrwxrwxrwx    1 mmdeye   mmdeye          7 nov 22 20:20 premtp -> tp1.pdf
drwxrwxr-x    2 mmdeye   mmdeye       4096 nov 22 20:13 tp
-rw-rw-r--    1 mmdeye   mmdeye         32 nov 22 20:24 tp1.pdf
```

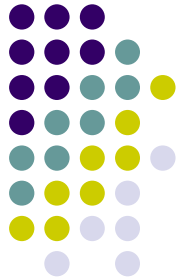
Commandes de base



- **Place occupée sur le disque :**
  - du : calcul de la place occupée par des fichiers ou des répertoires
  - df : place libre sur le disque (avec point de montage)
  - Exemple :

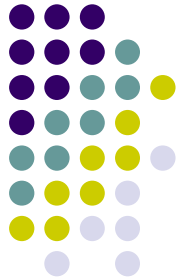
```
[mmdeye@mmdeye ~/licpro]$ ls -F
autrestp@  deuxtp@  lphytp1  premtp@  tp/  tp1.pdf
[mmdeye@mmdeye ~/licpro]$ du -a .
4       ./tp
4       ./tp1.pdf
0       ./premtp
0       ./autrestp
0       ./deuxtp
12      .
[mmdeye@mmdeye ~/licpro]$ df .
SysFichier           1K-blocs    Utilisé Dispo.   Util% Monté sur
/dev/hda6              5328800     83840   4974268   2% /home
[mmdeye@mmdeye ~/licpro]$ df -h .
SysFichier            Tail. Util.Disp. Uti% Monté sur
/dev/hda6             5.1G   82M  4.8G   2% /home
```

Commandes de base



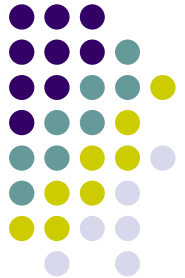
- Les métacaractères permettent de faire une sélection de fichiers suivant certains critères
- Le métacaractère le plus fréquemment utilisé est *, il remplace une chaîne de longueur non définie ls a*
- Le métacaractère ? remplace un caractère unique
- Les métacaractères [ ] représente une série de caractères
  - [acdf] toute lettre parmi la liste {a; c; d; f}
  - [!acdf] tout caractère autre que {a; c; d;f}
  - [a-f] les lettres de a à f

Commandes de base



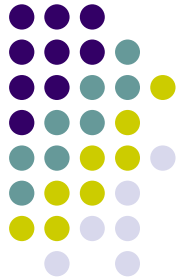
- Le critère **[aA]*** permet la sélection des fichiers dont le nom commence par un a ou A
- Le critère **[a-d]*** fait la sélection des fichiers dont le nom commence par a jusqu'à d
- Le critère ***[de]** fait la sélection des fichiers dont le nom se termine par d ou e
- **ab*** tout fichier commençant par ab
- **a??** tout fichier de 3 caractères commençant par a
- **[a-z]*.[cho]** tout fichier commençant par une lettre minuscule et dont l'extension est .c, .h ou .o

Exercice



- Le répertoire /usr/include contient les fichiers d'entête standards en langage C (stdlib.h,...).
  - Créer un répertoire nommé include dans votre répertoire de connexion (home). En utilisant une seule commande, y copier les fichiers du répertoire /usr/include dont le nom commence par **std** et termine par **.h**
  - Afficher la liste des fichiers de /usr/include dont le nom commence par **a**, **b** ou **c** et termine par **.h**
  - Afficher la liste des fichiers de /usr/include dont le nom comporte **3** caractères suivis de **.h**

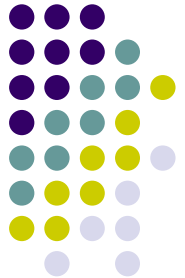
Commandes de base



- **Variables d'environnement**
  - Le système unix défini pour chaque processus une liste de variables d'environnement , qui permettent de définir certains paramètres : répertoires d'installation des utilitaires, type de terminal, etc
  - Chaque programme peut accéder à ces variables pour obtenir des informations sur la configuration du système
  - Les commandes de manipulation des variables sont :
    - printenv : affiche l'ensemble des variables systèmes
    - printenv VAR : affiche la valeur de la variable système VAR
    - export VAR=value : affecte value a la variable système VAR

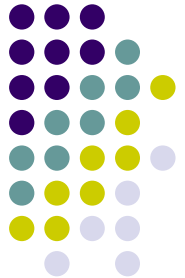
```
[mmdeye@mmdeye ~]$ export LICPRO=2005-2006
[mmdeye@mmdeye ~]$ echo $LICPRO
2005-2006
```

Commandes de base



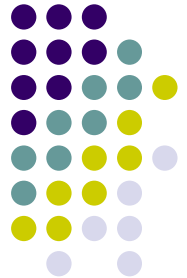
- Les principales variables système sont :
  - HOME : contient le répertoire racine de l'utilisateur
  - PATH : contient l'ensemble des chemins dans lesquels les exécutables sont recherches
  - PS1 : contient la définition du prompt
  - ...

Commandes de base



- **Initialisation d'un shell(**bash**) :**
  - Le script « /etc/profile » communs à tous les users y compris root
  - $HOME/.bash_profile : Un fichier de démarrage personnel et paramétrable
  - $HOME/.bashrc dans lequel il est recommandé de placer toutes les fonctions ou alias personnels (car .bashrc est exécuté dans tout shell)

Commandes de base

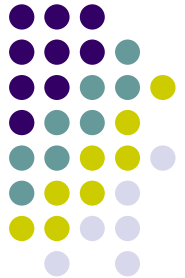


- Alias : les alias sont des raccourcis permettant de donner des noms simples a des commandes complexes :

```
[mmdeye@mmdeye ~/licpro]$ l
bash: l: command not found
[mmdeye@mmdeye ~/licpro]$ alias l='ls -li'
[mmdeye@mmdeye ~/licpro]$ l
total 12
 129056 lrwxrwxrwx    1 mmdeye   mmdeye          2 nov 22 20:15 autrestp -> tp
 129057 lrwxrwxrwx    1 mmdeye   mmdeye          7 nov 22 20:20 deuxtp -> tp1.pdf
 129059 -rw-rw-r--    2 mmdeye   mmdeye         32 nov 22 20:24 lphytp1
 129054 lrwxrwxrwx    1 mmdeye   mmdeye          7 nov 22 20:20 premtp -> tp1.pdf
 483873 drwxrwxr-x    2 mmdeye   mmdeye       4096 nov 22 20:13 tp
 129059 -rw-rw-r--    2 mmdeye   mmdeye         32 nov 22 20:24 tp1.pdf
```

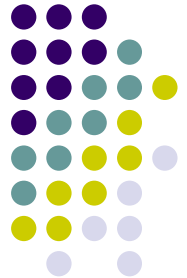
- La commande unalias sert a supprimer un alias

Commandes de base



- **Commandes pour utilisateurs et groupes :**
  - id : affiche les informations d'identification de l'utilisateur
  - whoami : affiche le nom de l'utilisateur
  - users : affiche les noms de tous les utilisateurs connectés sur le système
  - who : comme users mais avec plus d'informations
  - passwd : change le mot de passe actuel
  - quota : affiche les informations de quota (si disponibles)

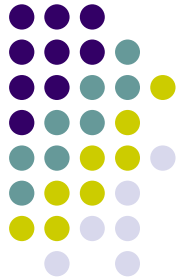
Commandes de base



- groups : affiche les groupes auxquels l'utilisateur appartient
- newgrp groupname : changement de groupe
- su username : changement d'identité de l'utilisateur
- lastlog : date de la dernière connection

```
[mmdeye@mmdeye ~]$ id
uid=500(mmdeye) gid=500(mmdeye) groupes=500(mmdeye)
[mmdeye@mmdeye ~]$ whoami
mmdeye
[mmdeye@mmdeye ~]$ users
mmdeye mmdeye mmdeye
[mmdeye@mmdeye ~]$ who
mmdeye    :0           Nov 22 20:07
mmdeye    pts/0        Nov 22 20:08 (:0.0)
mmdeye    pts/1        Nov 22 21:57 (:0.0)
[mmdeye@mmdeye ~]$ quota
Disk quotas for user mmdeye (uid 500): none
[mmdeye@mmdeye ~]$ groups
mmdeye
```

Commandes de base

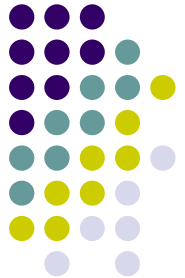


- man : cette commande permet d'afficher le manuel en ligne. Toute la documentation est en ligne sous Unix
  - man ls [Entrée]

- which : permet de savoir quel est le fichier correspondant à une commande
  - $ which csh [Entrée]
  - /bin/csh

- whereis : donne le chemin complet vers une commande, mais aussi sa page man
  - $whereis rm [Entrée]
  - rm: /bin/rm /usr/share/man/man1/rm.1.bz2

L’éditeur vi

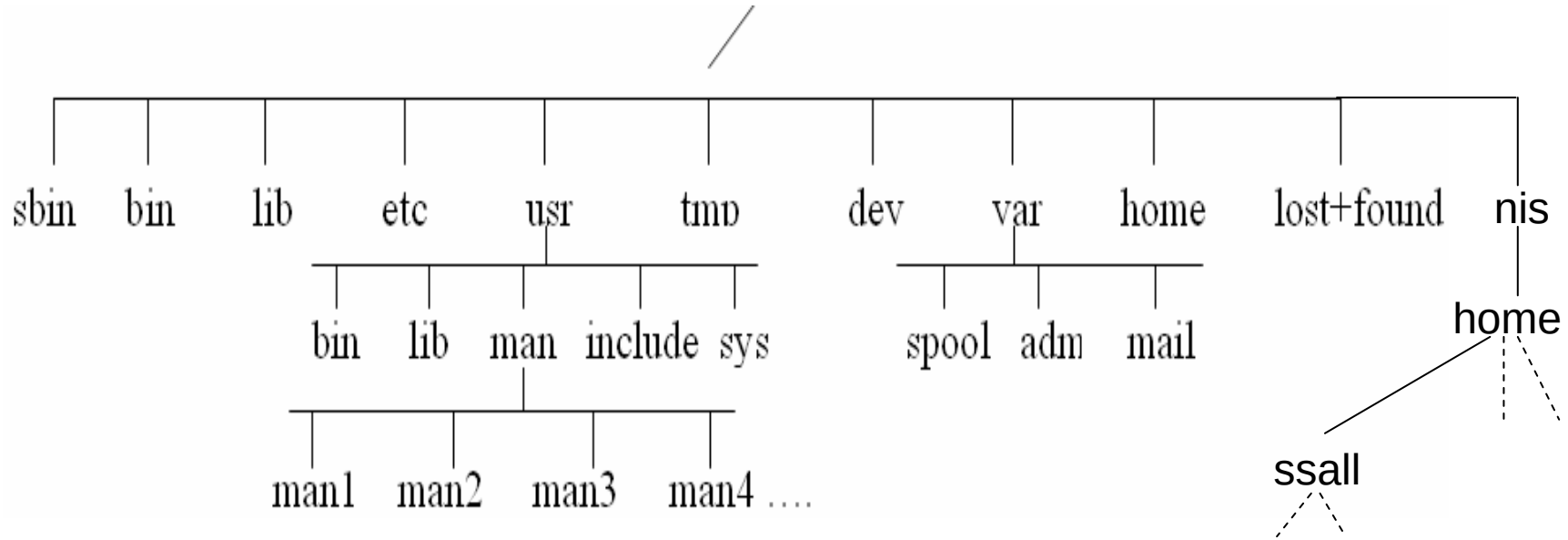


- vi est un éditeur fonctionnant à la fois en mode ligne (ex) et en mode écran. En mode écran, on a soit :
  - le mode commande soit
  - le mode insertion
- On sort du mode insertion avec la touche escape (ESC) et on y entre avec la commande i (insert) ou a (append)
- L’éditeur peut être appelé de plusieurs façons :
  - vi pour le mode écran
  - view pour le mode écran (lecture seule)
  - ex pour le mode ligne
- TP

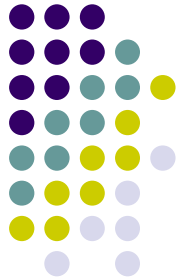
Le système de fichiers

- Sous UNIX TOUT est fichier

- Ces fichiers sont organisés dans une arborescence unique composée:
  - d'une racine (/), des nœuds (les répertoires) et des feuilles (fichiers)

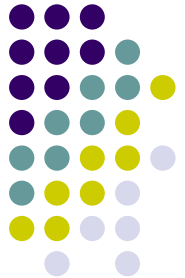


Le système de fichiers



- **Types de fichiers :**
  - les fichiers normaux : un fichier normal (texte, objet, exécutable, etc.) est une séquence d'octets sans aucune organisation interne spécifique
  - les répertoires : qui contiennent une liste de références à d'autres fichiers UNIX
  - les fichiers spéciaux : associés par exemple à des pilotes de périphériques
  - les tubes et sockets : utilisés pour la communication entre processus
  - les liens symboliques (fichiers "pointant" sur un autre fichier)

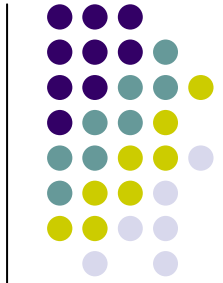
Le système de fichiers



- **Table des i-nodes :**
  - Chaque élément (i-node) de la table décrit entièrement un fichier :
    - type de fichier et permissions d'accès
    - identification du propriétaire et de son groupe
    - nombre de liens sur le fichier
    - taille du fichier en octets
    - les dates de création, dernière modification, dernier accès en lecture
    - tableau de pointeurs sur les blocs de données

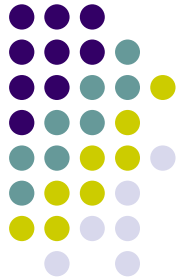
  - Mise à part la dernière (le tableau de pointeurs) ces informations peuvent être visualisées par la commande ls -l par exemple

Le système de fichiers



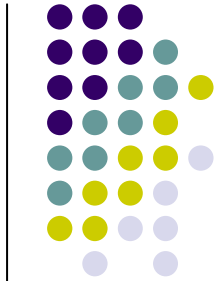
- La liaison entre les noms des fichiers et les informations enregistrées dans le i-node du fichier se fait grâce aux répertoires
- Un fichier répertoire contient le nom et le numéro de i-node des fichiers placés directement sous ce répertoire dans l'arborescence des fichiers
- Les informations concernant un fichier sont donc conservées dans deux endroits différents :
  - le nom du fichier est conservé dans le répertoire contenant le fichier
  - les informations systèmes sont conservées dans le i-node du fichier
- Le contenu du fichier (les données) est enregistré dans les blocs du disque dont les adresses sont dans le i-node

Le système de fichiers



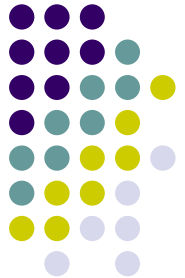
- **Hiérarchie des fichiers sous Linux :**
  - /bin : commandes de base d'Unix (ls, cat, cp, mv, rm, vi etc.)
  - /dev : les fichiers spéciaux représentant les périphériques (devices)
    - /dev/fd0 : lecteur de disquette,
    - /dev/cdrom : périphérique lecteur de CD-ROM,
    - /dev/lp0 : imprimante
    - /dev/hda1 : 1ère partition du 1er disque dur IDE
    - …
  - /etc : fichiers de configuration propres à la machine
    - /etc/passwd contient les mots de passe
    - /etc/fstab contient les systèmes de fichiers à monter lors du lancement du système(fstab : file system table)
    - …

Le système de fichiers



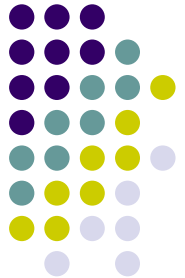
- /home : racine des répertoires des utilisateurs
- /lib : bibliothèques de programmes
- /mnt : contient les points de montage des partitions temporaires (cd-rom, disquette, ...)
- /root : répertoire de l'administrateur root
- /sbin : commandes d’administration
- /tmp : fichiers temporaires du système ou des utilisateurs
- /usr : programmes et utilitaires mis à la disposition des utilisateurs
  - /usr/bin : exécutables des utilitaires : cc, man etc
  - /usr/include : les fichiers d’entête pour développer en C
  - /usr/local : contient les commandes locales
  - /usr/X11R6 : tous les fichiers du système X Window
  - …

Le système de fichiers



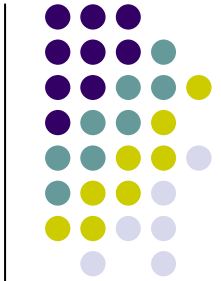
- /var : contient les données qui varient lorsque le système fonctionne normalement
  - /var/spool : répertoires pour les files d'attente des imprimantes
  - /var/log : fichiers traçant l'exécution de différents programmes
  - …
- /proc : Il n'existe pas sur le disque. Il est en mémoire. Il est utilisé pour fournir des informations sur le système
  - /proc/1 : contenant des infos sur le processus numéro 1.Chaque processus possède un répertoire sous /proc portant le nom de son identificateur
  - /proc/filesystems : systèmes de fichiers supportés par le noyau
  - /proc/devices : liste des pilotes de périphériques configurés dans le noyau
  - …
- Ce système de fichiers peut résider sur différentes partitions, différents supports physiques ou sur d'autres machines sur le réseau

Commandes de base



- **Les liens :**
  - Liens physiques : **ln <nom_fic> <nouveau_nom_fic>**
    - permet de donner plusieurs noms à un fichier
    - pas pour les répertoires
    - ne traverse pas les partitions
    - un fic est détruit quand TOUS ses liens physiques sont supprimés (≠ raccourcis)
  - Liens symboliques : **ln -s <nom_fic> <nouveau_nom_fic>**
    - crée un raccourci
    - traverse les partitions
    - fonctionne aussi pour les répertoires
  - Lister les liens d'un fichier: **ls -l <nom_fic>**

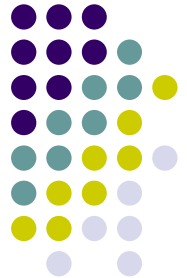
Commandes de base



- Un lien hard n'a pas d'inode propre, il a l'inode du fichier vers lequel il pointe
- Par contre un lien symbolique possède sa propre inode
- Pour connaître le numéro d'inode d'un fichier : ls –i nom_fic

```
[mmdeye@mmdeye ~/licpro]$ ln tp1.pdf lphytp1
[mmdeye@mmdeye ~/licpro]$ ls -il
total 12
 129056 lrwxrwxrwx    1 mmdeye   mmdeye          2 nov 22 20:15 autrestp -> tp
 129057 lrwxrwxrwx    1 mmdeye   mmdeye          7 nov 22 20:20 deuxtp -> tp1.pdf
 129059 -rw-rw-r--    2 mmdeye   mmdeye         32 nov 22 20:24 lphytp1
 129054 lrwxrwxrwx    1 mmdeye   mmdeye          7 nov 22 20:20 premtp -> tp1.pdf
 483873 drwxrwxr-x    2 mmdeye   mmdeye       4096 nov 22 20:13 tp
 129059 -rw-rw-r--    2 mmdeye   mmdeye         32 nov 22 20:24 tp1.pdf
```

Les droits d'accès



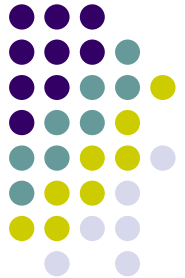
- **Principe** : sous Unix , la sécurité se gère :
  sur trois niveaux et avec trois types d'acces :

① propriétaire (User)	❶ lecture (Read)
② groupe (Group)	❷ écriture (Write)
③ les autres (Others)	❸ exécution (eXecute)

- 3 types de permissions

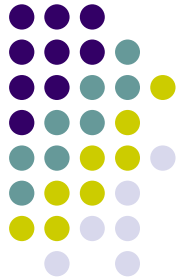
lecture (r)	afficher le contenu	afficher le contenu
écriture (w)	modifier	créer/supp fichiers
exécution (x)	exécuter	traverser
	fichier	**répertoire**

Les droits d'accès



- **Exemple** : sortie de la commande ls –l
  - -rw-r----- 1 mmdeye users 1819 Sep 7 08:32 toto
  - La lecture des 10 premiers caractères est la suivante :
    - \- type : toto est un fichier
    - rw- droits d'accès pour l'utilisateur : lecture et écriture
    - r-- droits d'accès pour le groupe : lecture seule
    - --- droits d'accès pour les autres : aucun

Les droits d'accès

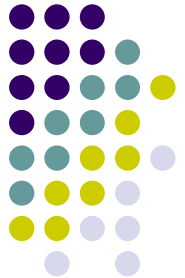


- **Gestion des droits :**
  - La commande chmod permet de modifier les droits d'accès d'un fichier (ou répertoire). Pour pouvoir l'utiliser sur un fichier ou un répertoire, il faut en être le propriétaire
  - La syntaxe est la suivante :

    chmod <classe op perm …> <fic>
    - classe: (u : user, g : group, o : others, a : all)
    - op: (= : affectation, - : suppression, + : ajout)
    - perm: (r : lecture, w : écriture, x : exécution)

  - Exemples :
    - chmod u-w g+r toto
    - chmod g=rx o-rxw /home/mmdeye
    - chmod a=rx /home/mmdeye/.messages

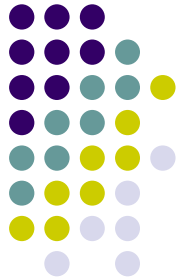
Les droits d'accès



- Exemple :
  - chmod u-w g+r toto
  - chmod g=rx o-rxw /home/pascal
  - chmod a=rx /home/pascal/.messages
  - chmod 700 toto
  - chmod 644 toto

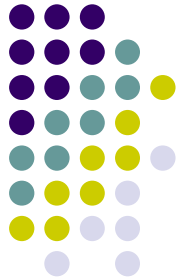
- Pour les deux derniers exemples, chaque chiffre code un niveau d'accès (dans l'ordre utilisateur-groupe-autres) dont le droit d'accès est codé en octal avec : 4=lecture, 2=écriture, 1=exécution, 0=rien

Les droits d'accès



- **Les droits par défaut : umask**
  - Quand vous créer un fichier, par exemple avec la commande touch, ce fichier par défaut possède certains droits. Ce sont 666 pour un fichier (-rw-rw-rw-) et 777 pour un répertoire (drwxrwxrwx),ce sont les droits maximum
  - La commande **`umask`** permet de changer ces paramètres par défaut
    - **`umask 026`**
  - Par défaut les fichiers auront comme droit 640 (-rw-r-----)
  - Par défaut les répertoires auront comme droit 751 (-rwxr-x--x)

Les droits d'accès



- **Changer le propriétaire et le groupe:**
  - Vous pouvez donner un fichier vous appartenant à un autre utilisateur, c'est à dire qu'il deviendra propriétaire du fichier, et que vous n'aurez plus que les droits que le nouveau propriétaire voudra bien vous donner sur le fichier

    ```
    chown nouveau-propriétaire nom-fichier
    ```

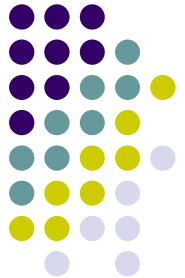
  - Dans le même ordre d'idée vous pouvez changer le groupe

    ```
    chgrp nouveau-groupe nom-fichier
    ```

  - Ces deux commandes ne sont utilisables que si on est propriétaire du fichier

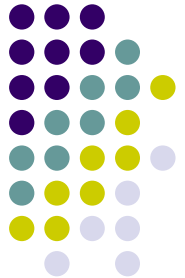
  - NOTA : Sur certains UNIX suivant leur configuration, on peut interdire l'usage de ces commandes pour des raisons de sécurité

Les droits d'accès



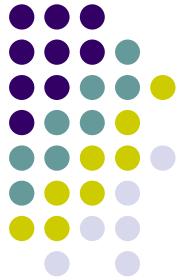
- **Le super-utilisateur**
  - Afin de permettre l'administration du système, un utilisateur spécial, nommé super utilisateur (ou root), est toujours considéré par le système comme propriétaire de tous les fichiers (et des processus)
  - La personne qui gère le système est normalement la seule à connaître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

Les redirections & pipes



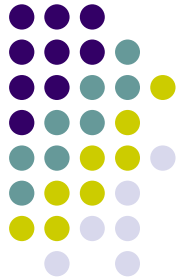
- Chaque processus dispose de 3 fichiers ouverts et affectés par défaut :
  - **`stdin`** est l'entrée standard, *cad* où les informations attendues en entrée sont lues (par défaut le clavier)
  - **`stdout`** est la sortie standard, *cad* où les résultats d'une commande sont transmis (par défaut l'écran)
  - **`stderr`** est la sortie d'erreur, *cad* où les messages d'erreur sont transmis (par défaut l'écran)
- Chacun de ces flux de données est identifié par un numéro descripteur : 0 pour stdin, 1 pour stdout et 2 pour stderr

Les redirections & pipes



- **Redirection de la sortie :**
  - Cela peut être utile, si vous avez une commande qui génère énormément de commentaire, et que vous voulez les récupérer, pour les exploiter par la suite, à la terminaison de la commande
  - Le symbole de redirection est le caractère **`>`** suivi du nom_de_fichier sur lequel on souhaite faire la redirection
  - **`command > file`** : place le résultat de la sortie de **`command`** (stdout) dans le fichier nommé **`file`**. Si **`file`** existe, il est écrasé ; sinon, il est créé
  - **`command >> file`** : ajoute à la fin du fichier **`file`** le résultat de la sortie de **`command`** (stdout). Si le fichier n'existe pas, il est créé

Les redirections & pipes



Redirection des entrées:

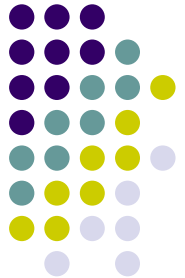
- Le symbole de redirections des entrées est le caractère **<** suivi du nom_du_fichier à partir duquel on souhaite entrer les données

- Exemple :
- **`cat  < premier.txt`**
- Le fichier premier.txt est pris comme entrée au lieu du clavier

- **`sort < mon-fichier`**
- Envoie le contenu du fichier mon-fichier vers la commande sort ,
- celle-ci va donc trier le contenu du fichier

- Il est possible de rediriger l'entrée et la sortie en même temps :
- **`sort < mon-fichier > fichier-trie`**
- Le résultat est sauvegardé dans un fichier(fichier-trie)

Les redirections & pipes



Redirection des erreurs:

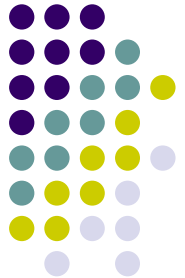
- Il est également possible de rediriger la sortie des erreurs (stderr) en faisant précéder le symbole **>** ou **>>** du chiffre **2**

- **`command 2> file`** : comme ci-dessus, mais concerne la sortie stderr (i.e. les messages d'erreur issus de la commande).

- **`command 2>> file`** : comme ci-dessus, mais concerne la sortie stderr

- Exemple : **`mkdir /root/monrep 2>msgerreur.txt`**

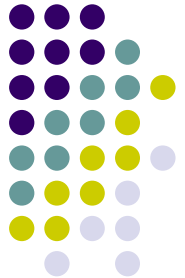
Les redirections & pipes



- **Le fichier virtuel /dev/null :**
  - **`/dev/null`** est un fichier virtuel toujours vide utilisée dans les redirections
  - utilisé en entrée , il n'envoie rien (on l'utilise pour s'assurer qu'une commande ne reçoit rien en entrée)
  - utilisé en sortie , il joue le rôle de poubelle (on redirige dans /dev/null les sorties que l'on ne souhaite pas)

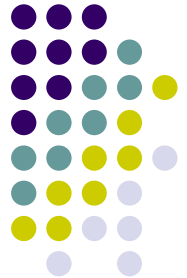
  - Exemple : **`ls -l /root/monrep 2>/dev/null`**

Les redirections & pipes



- **Les tubes (en anglais pipes)**
  - Un tube, symbolisé par une barre verticale (caractère «|»), permet d'affecter la sortie standard d'une commande à l'entrée standard d'une autre
  - **`command1 | command2`** : passe le résultat de la sortie de command1 comme entrée à command2
  - **`command1 | tee file | command2`** : comme ci-dessus, mais la sortie de command1 est en plus placée dans le fichier file
  - **`echo Happy Birthday! | rev`**
  - **`!yadhtriB yppaH`**

Contrôle de jobs



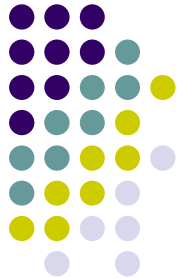
- La plupart de shells proposent un système de contrôle des jobs
- Un job peut être composée de plusieurs processus **`ls |less`**

- Le shell permet avec & placé à la fin d'une ligne de commande de la lancer en arrière-plan
  - **`emacs &`**
  - [1] 2208
- Pour affichera la liste des jobs

```
[mmdeye@mmdeye mmdeye]$ jobs
[1]-  Running                 emacs &
[2]+  Stopped                 xclock
```

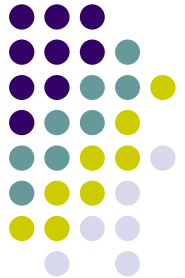
- Le signe + indique le job en cours (le job le plus récemment manipulé) et – indique l'avant dernier job

Contrôle de jobs



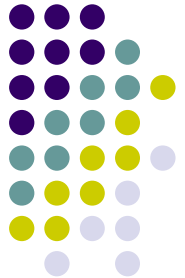
- **`Ctrl+z`** permet de suspendre le job lancé en avant-plan
- **`Ctrl+c`** tue le processus du premier plan
- **La commande `fg`**
  - mettre en avant-plan un job lancé en arrière plan
  - Redémarrer un job suspendu et l'exécuter en avant-plan
  - **`fg %1`** mettra le job 1 en avant-plan
  - **`fg`** sans paramètre agit sur le job en cours
- **La commande `bg`**
  - Redémarrer un job suspendu et l'exécuter en arrière plan
  - **`bg %1`** mettra le job 1 en arrière-plan
  - **`bg`** sans paramètre agit sur le job en cours

Contrôle de jobs



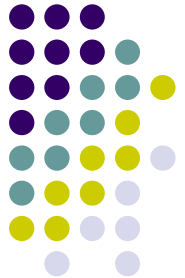
- **Exercice**
  1. Lancer un job par **`man bash`** et le suspend avec **`Ctrl+Z`**
  2. Exécuter **`xclock`** en arrière plan avec `&`
  3. Utiliser **`jobs`** pour lister les jobs lancés en arrière plan et les suspendus
  4. Utiliser **`fg`** pour mettre **`man`** en avant plan, et taper `q` pour quitter
  5. Utiliser **`fg`** pour mettre **`xclock`** en avant plan, et le terminer avec **`Ctrl+C`**
  6. Relancer **`xclock`** et essayer le suspendre avec **`Ctrl+z`**

Contrôle de jobs



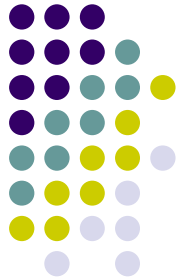
- **Lancement de tâches :**
  - Deux programmes P1 et P2 peuvent s'enchaîner comme suit:
    - **`P1 | P2`** : le résultat de P1 est l'entrée de la commande P2
    - **`P1 && P2`** : P2 s'exécute si P1 est sans erreur
    - **`P1 || P2`** : si erreur sur P1: on exécute P2
    - **`P1 & P2`** : exécute P1 et P2 simultanément (astuce car en fait P1 est passé en background)
    - **`P1 ; P2`** : exécute P1 puis P2

Contrôle de jobs



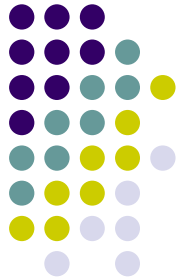
- **Exemples :**
  - **`ls | wc -l`** compte le nombre de fichiers dans le répertoire courant
  - **`date ; pwd`** affiche la date puis le répertoire courant
  - **`test -f fichier && more fichier`** si et seulement si `test` retourne 0 l'enchaînement avec `more` s'exécute et affiche le fichier
  - **`test -f fichier || “fichier inconnu”`** le texte `“fichier inconnu”` s'affiche que si `test` retourne une valeur différente de 0
  - On peut enchaîner plusieurs commandes :
    - **`test -f fichier && more fichier || “Ce fichier n'existe pas dans le répertoire de travail courant $PWD”`**

Gestion des processus



- Le noyau considère chaque programme encours d'exécution comme un processus
- Le noyau identifie chaque processus par un numéro appelé PID (Process Identification)
- Pour chaque processus, il y a un processus père (PPID) – le pid du processus qui l'a créé
- Le seul qui ne suit pas cette règle est le premier processus lancé sur le système "le processus **init**" qui n'a pas de père et qui a pour **PID 1**

Gestion des processus



- Dans un shell, un processus peut être exécute soit en premier plan (on doit attendre la fin de l'exécution pour pouvoir entrer une nouvelle commande), soit en tâche de fond (on récupère la main tout de suite, le processus tournant en parallèle avec le shell)

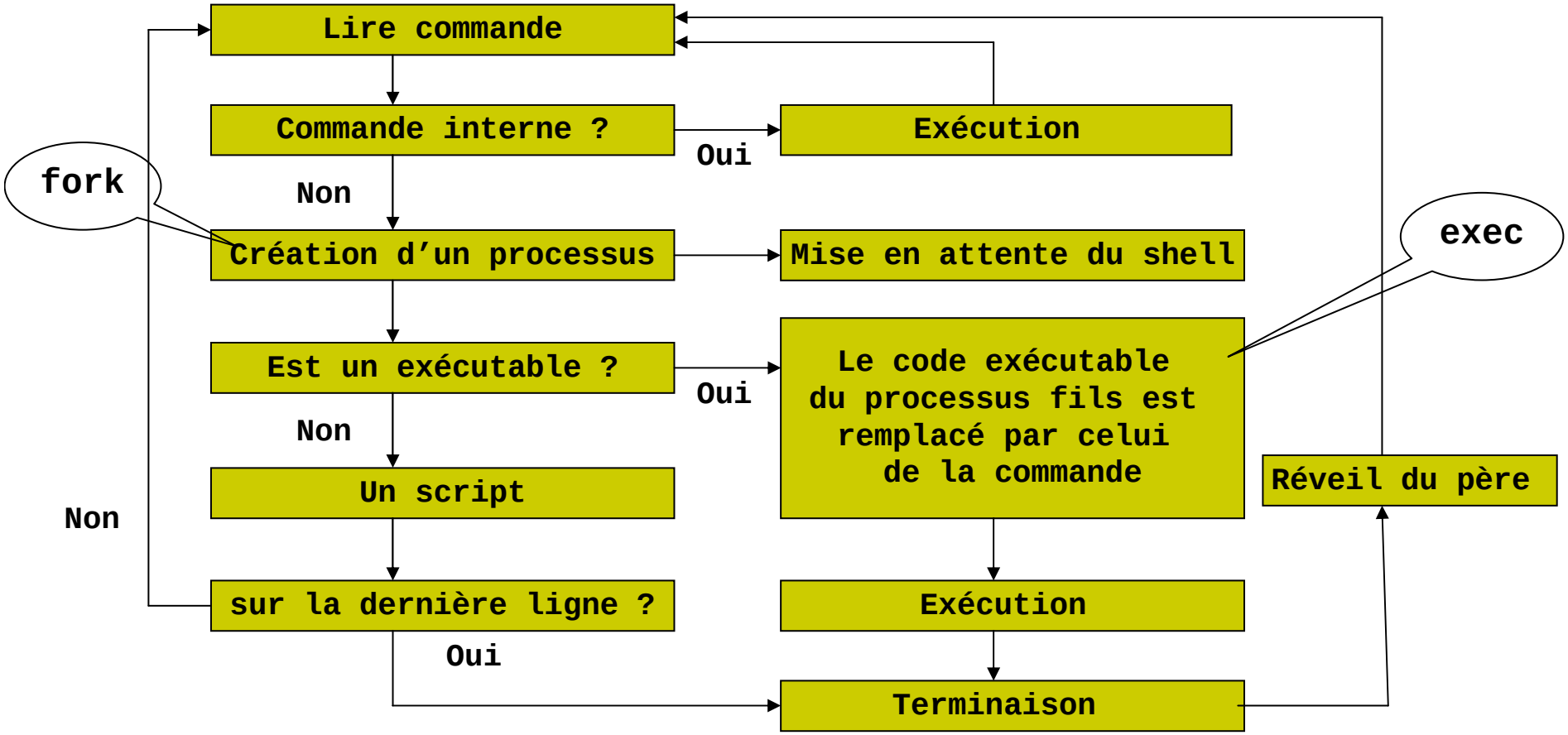
- Un processus peut être stoppé, puis relancé plus tard sans l'affecter, ou bien même tué. Ceci est effectué par l'envoi de signaux au processus

- Seul le propriétaire d'un processus peut le contrôler (exception faite du super-utilisateur)

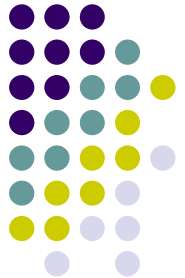
- La fin d'un processus père entraîne la fin de tous ces processus fils

Gestion des processus

- A chaque fois qu'une commande est tapée :

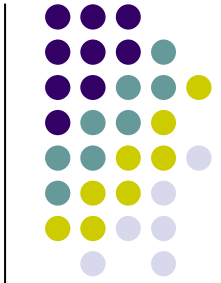


Gestion des processus



- **La gestion d'un processus utilise les propriétés suivantes :**
  - un numéro unique affecté à sa création (PID ou process ID)
  - le numéro du processus parent qui l' a lancé (PPID)
  - l'identité de son propriétaire
  - ses caractéristiques temporelles (date de début, temps CPU utilisé)
  - ses caractéristiques mémoires (mémoire vive et virtuelle utilisées)
  - sa priorité (-20=priorité minimale, 0=priorité normale, 20=priorité maximale)
  - son état (R=exécution, T=stoppé, …)

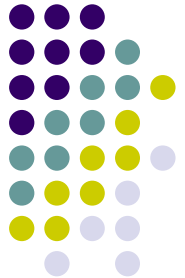
Gestion des processus



- **La commande ps : affiche le statut des processus (par défaut, seulement ceux de l'utilisateur)**

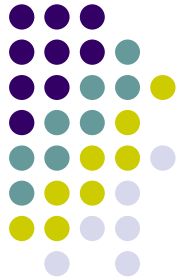
```
[mmdeye@mmdeye mmdeye]$ ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 16:09 ?        00:00:04 init
root         2     1  0 16:09 ?        00:00:00 [keventd]
root         3     1  0 16:09 ?        00:00:00 [kapmd]
root         4     1  0 16:09 ?        00:00:00 [ksoftirqd_CPU0]
root         9     1  0 16:09 ?        00:00:00 [bdflush]
root         5     1  0 16:09 ?        00:00:00 [kswapd]
root         6     1  0 16:09 ?        00:00:00 [kscand/DMA]
root         7     1  0 16:09 ?        00:00:00 [kscand/Normal]
root         8     1  0 16:09 ?        00:00:00 [kscand/HighMem]
```

Gestion des processus



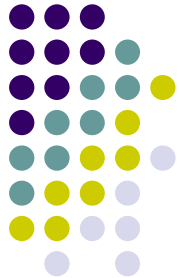
- **La signification des différentes colonnes :**
  - **UID** nom de l'utilisateur qui a lancé le process
  - **PID** correspond au numéro du process
  - **PPID** correspond au numéro du process parent
  - **C** au facteur de priorité : plus la valeur est grande, plus le processus est prioritaire
  - **STIME** correspond à l'heure de lancement du processus
  - **TTY** correspond au nom du terminal
  - **TIME** correspond à la durée de traitement du processus
  - **CMD** correspond au nom du processus

Gestion des processus



- **Contrôle des processus:**
  - **kill -num pid** : envoie le signal num au processus pid. Si -num n'est pas spécifié, le signal par défaut est 15
  - **nohup commande** : indique à la commande qu'elle ne doit pas se terminer si son père meurt
  - **wait** : rend la main lorsque tous les processus en tâche de fond sont terminé
  - **nice renice** : changement de la priorité d'un processus (root)
  - **at batch** : lancement différé de processus

Gestion des processus

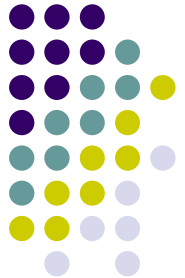


- **Visualiser les processus d'un seul utilisateur :**
  - **ps –u olivier**

- **Changer la priorité d'un processus :**
  - **nice –valeur commande**
  - plus le nombre est grand, plus la priorité est faible
  - **nice -5 man ps**

- **La commande renice permet de changer la priorité d'un processus en cours :**
  - **renice valeur PID**

Gestion des processus



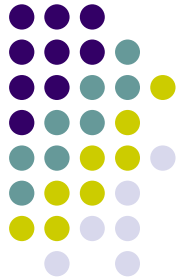
- **Arrêter un processus :**
  - **kill -9 pid**
  - Un utilisateur ne peut arrêter que les processus qui lui appartient
- **Principaux signaux :**
  - **9 SIGKILL** forcer terminaison
  - **15 SIGTERM** terminer (défaut)
  - **17 SIGSTOP** stopper (ou pause)

Gestion des processus

- La commande **pstree** permet de visualiser l'arborescence des processus

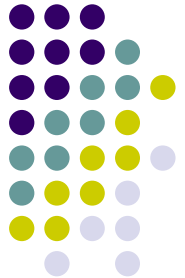
```
[mmdeye@mmdeye mmdeye]$ pstree
init─┬─apmd
     ├─atd
     ├─bdflush
     ├─bonobo-activati
     ├─cannaserver
     ├─crond
     ├─cupsd
     ├─eggcups
     ├─gconfd-2
     ├─gdm-binary───gdm-binary─┬─X
     │                         └─gnome-session───ssh-agent
     ├─gnome-panel
     ├─gnome-settings-
     ├─gnome-terminal─┬─bash───su───bash
     │                ├─bash─┬─emacs
     │                │      └─pstree
     │                └─gnome-pty-helpe
```

Gestion des processus



- La commande **top** affiche les processus utilisant le plus de temps de processeur
- L'affichage est actualisé périodiquement
- En plus des informations sur les processus, top donne des indicateurs sur l'état du système : occupation de la mémoire, de l'unité centrale...
- top montre l'évolution de ces indicateurs en « temps réel »

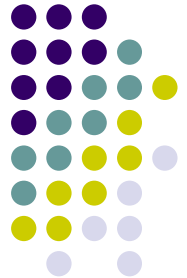
Gestion des processus



- **Commandes interactives de top :**
  - **[q]** pour quitter
  - **[Barre espace]** réactualise immédiatement l'affichage
  - **[h]** afficher l'écran d'aide
  - **[k]** envoyer un signal à un processus
  - **[r]** change la priorité d'un processus
  - ...

- **time** commande : exécute la commande puis affiche son temps d'exécution (real=écoulé, user=en mode user, sys=en mode noyau)

Gestion des processus



- **Exercice :**
  1. Créer le script suivant, nommé forever, dans votre répertoire home:

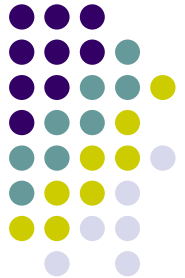
```
#!/bin/sh
while [ 1 ]; do
echo hello... >/dev/null;
done
```

  2. Le rendre exécutable et l'exécuter en arrière plan

```
chmod a+rx forever
./forever &
```

  3. Utiliser ps -l pour contrôler sa priorité

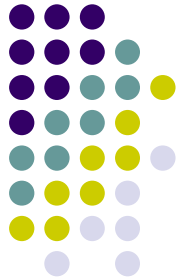
Gestion des processus



4. Lancer le script par nice avec une periorité de 15. essayer de exécuter une autre version avec priorité moins importante et suivre la différence avec top

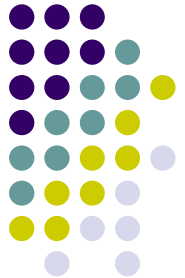
5. Utiliser nice ou renice pour créer un processus avec priorité moins de 0

La commande grep



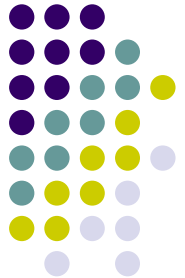
- **La commande grep permet de rechercher une chaîne de caractères dans un fichier. Les options sont les suivantes :**
  - **-v** affiche les lignes ne contenant pas la chaîne
  - **-c** compte le nombre de lignes contenant la chaîne
  - **-n** chaque ligne contenant la chaîne est numérotée
  - **-x** ligne correspondant exactement à la chaîne
  - **-l** affiche le nom des fichiers qui contiennent la chaîne
  - **-i** ignorer la casse (minuscules/majuscules)

La commande grep



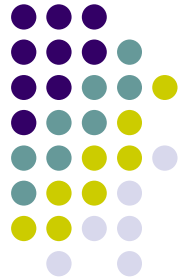
- **grep root /etc/passwd**
  - root:x:0:0:root:/root:/bin/bash
- Une option intéressante de **grep** est l'option **-l** qui permet d'avoir uniquement les noms de fichiers en résultat
- **grep -l root /var/log/* 2>/dev/null**
- **grep** recherche des chaînes de caractères, qui peuvent être un mot complet "terre", une suite de lettres "tre", ou une expression régulière

Les expressions régulières



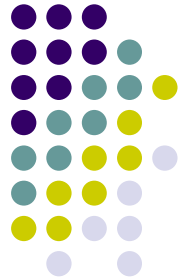
- Les expressions régulières sont des suites de caractères permettant de faire des sélections. Elles fonctionnent avec certaines commandes comme grep
- Les différentes expressions régulières sont :
  - **^** début de ligne
  - **.** un caractère quelconque
  - **$** fin de ligne
  - **x*** zéro ou plus d'occurrences du caractère x
  - **x+** une ou plus occurrences du caractère x
  - **x?** une occurrence unique du caractère x
  - **[...]** plage de caractères permis
  - **[^...]** plage de caractères interdits
  - **\{n\}** pour définir le nombre de répétition n du caractère placé devant

La commande grep



- Chercher toutes les lignes commençant par «a» ou «A»
  - **grep -i '^a' fichier** ou **grep '^[aA]' fichier**
- Chercher toutes les lignes finissant par «rs»
  - **grep 'rs$' fichier**
- Chercher toutes les lignes contenant au moins un chiffre
  - **grep '[0-9]' fichier**
- Afficher toutes les lignes qui commencent par int de tous les fichiers se terminant par .c, avec les numéros de ligne
  - **grep -n '^int' *.c**

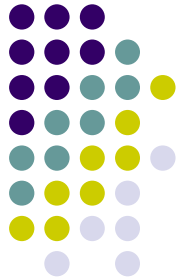
La commande grep



- Afficher toutes les lignes contenant “int” ou “double” ou “float”
  - **grep ’int\|double\|float’ *.c**

- Pour forcer la précédence :
  - **grep -i ’\(public\|private\|protected\) int’ *.java**

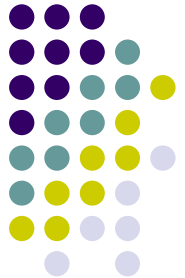
La commande find



- La commande **find** permet de retrouver des fichiers à partir de certains critères. La syntaxe est la suivante :
  - **find <répertoire de recherche> <critères de recherche>**

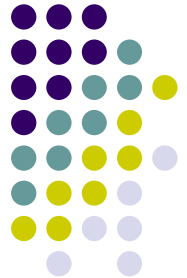
- Les critères de recherche sont les suivants :
  - **-name** recherche sur le nom du fichier,
  - **-perm** recherche sur les droits d'accès du fichier,
  - **-links** recherche sur le nombre de liens du fichier,
  - **-user** recherche sur le propriétaire du fichier,
  - **-group** recherche sur le groupe auquel appartient le fichier,
  - **-type** recherche sur le type (d=répertoire, c=caractère, f=fichier normal),

La commande find



- **-size** recherche sur la taille du fichier en nombre de blocs (1 bloc=512octets),
- **-atime** recherche par date de dernier accès en lecture du fichier,
- **-mtime** recherche par date de dernière modification du fichier,
- **-ctime** recherche par date de création du fichier.

La commande find

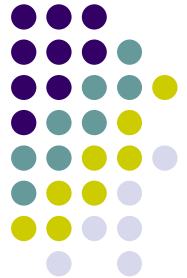


- **On peut combiner les critères avec des opérateurs logiques :**
  - critère1 critère2 ou critère1 -a critère2 correspond au et logique,
  - !critère non logique,
  - \ (critère1 -o critère2 \) ou logique,

- **Recherche par nom de fichier :**
  - chercher un fichier dont le nom contient la chaîne de caractères "toto" à partir du répertoire /usr :
    find /usr –name toto –print

  - Pour rechercher tous les fichiers se terminant par .c dans le répertoire /usr :
    find /usr –name "*.c" –print

La commande find



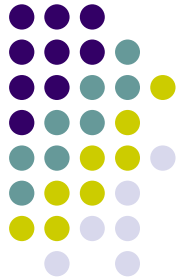
- **Recherche suivant la date de dernière modification :**
  - Pour connaître les derniers fichiers modifiés dans les 3 derniers jours dans toute l'arborescence (/) :

    find / -mtime -3 -print

- **Recherche combinée :**
  - Pour chercher dans toute l'arborescence, les fichiers Ordinaires appartenant à olivier, dont la permission est fixée à 755 :

    find / -type f –user olivier –perm 755 –print

La commande find



- **Recherche en utilisant les opérateurs logiques :**
  - Pour connaître les fichiers n'appartenant pas à l'utilisateur olivier :

    find . ! –user olivier -print

  - Recherche des fichiers qui ont pour nom a.out et des fichiers se terminant par .c :

    find . \(–name a.out –o –name "*.c" \) –print

- La commande **find** peut être couplée à l'utilisation d'une autre commande par l'utilisation de l'option **–exec**

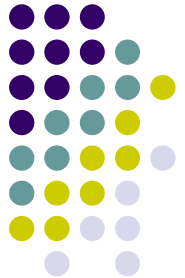
- Effacer tous les fichiers d'extension .o qui se trouvent dans le répertoire '/home/deye' :

  find /home/deye –name '*.o' –exec rm {} \ ;

Les filtres Unix

- Un filtre est une commande ayant les capacités :
  - de lire son entrée sur le canal d'entrée standard ou depuis un fichier ou un ensemble de fichiers passés en paramètre.
  - d’écrire sa sortie sur le canal de sortie standard et ses erreurs sur le canal de sortie d'erreur.

Les filtres Unix



- **Filtres d'affichage :**
  - **cat** : affichage
    - affichage simple : **cat fichier**
    - concaténation de fichier : **cat fic_1...fic_n**
      - Les fichiers **fic_i** sont envoyés dans l'ordre et sans séparation vers stdout.
    - création de fichier :

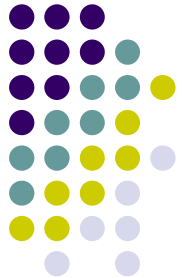
      cat > fichier

      mon texte.

      ^D

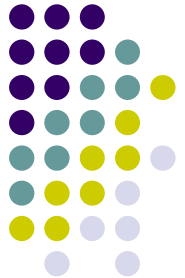
      - Aucun fichier n'est donnée en entrée, c'est donc l'entrée stdin standard (le clavier) qui est utilisée et redirigée vers **fichier**

Les filtres Unix



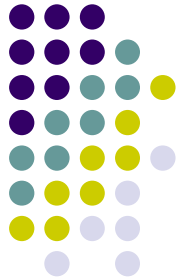
- **more(less) :** affichage avec pause à chaque page
  - consultation d'un fichier : **more fichier**
    Possède des commandes de recherche similaires à celle de vi
  - consultation page à page du résultat d'une commande : **command | more**
  - Cette commande étant interactive, sa sortie n'est généralement ni redirigée ni pipée.

Les filtres Unix



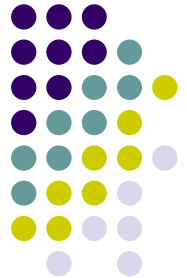
- **Éditer un fichier par la fin : tail**
  - Si vous avez un fichier très long, et que vous voulez visualiser que la fin, vous disposez de la commande tail :
    - **tail +10 mon-fichier** ==> vous obtenez toutes les lignes du fichier de la 10ème jusqu'à la fin
    - **tail -10 mon-fichier** ==> vous obtenez les 10 dernières lignes à partir de la fin
  - Vous pouvez indiquer si votre unité est la ligne (par défaut), le bloc ou le caractère avec l'option –c
    - **tail -10 -c mon-fichier** ==> vous obtenez les 10 derniers caractères du fichier

Les filtres Unix



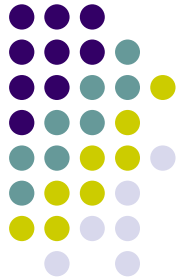
- **Éditer un fichier par le début : head**
  - Si vous avez un fichier très long, et que vous voulez visualiser que le début, vous disposez de la commande head :
    - **head -10 mon-fichier** ==> vous obtenez les 10 premières lignes à partir du début.
  - Vous pouvez indiquer si votre unité est la ligne (par défaut), le bloc ou le caractère avec l'option -c
    - **head -10 –c mon-fichier** ==> Vous obtenez les 10 premiers caractères du fichier

Les filtres Unix



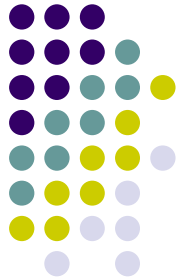
- **Filtre de tri : sort**
  - Tri ligne à ligne par ordre croissant.
  - Principales options :
    - **-n** tri numérique.
    - **-r** inverser l'ordre du tri.
    - **-f** ne pas différencier majuscules et minuscules.
    - **-t c** utiliser le caractère c comme séparateur de champs dans la ligne (par défaut, l'espace).
    - **-k fields** tri en fonction du sélecteur de champ.
  - Principe de sélection des champs (**fields**) :
    - **n,n** tri selon le n^{ième} champ.
    - **n** tri selon la fin de la ligne en commençant à partir du n^{ième} champ.
    - **n.m** tri selon la fin de la ligne en commençant à partir du m^{ième} caractère du nième champ.
    - **n1,n2** tri selon les champs à partir du champ n1 et jusqu'au champ n2.
    - **n1.m1,n2.m2** etc ...

Filtre de tri : sort



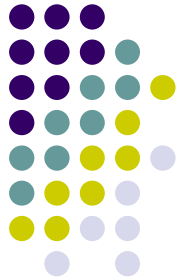
- **Exemples :**
  - **sort –t: +0 -1 /etc/passwd**
  - **sort –t: –k 1,1 /etc/passwd**
  - On peut spécifier la recherche sur un caractère situé à une position particulière, par exemple à la 2eme position du 6eme champ : **sort -t: +5.1 /etc/passwd**
  - Pour plusieurs critères de recherche, il faut spécifier derrière chaque champ le type de tri à mettre en oeuvre pour ce critère. Par exemple :
    - **sort -t: +0d -1 +2nr -3**
    - triera le 1er champ par ordre dictionnaire, et le 3eme champ par ordre numérique inverse.

Gestion de champs



- cut : permet d'extraire certains champs d'un fichier.
  - Options principales :
    - **-c liste** extrait suivant le nombre de caractères
    - **-f liste** extrait suivant le nombre de champs
    - **-d c** utiliser le caractère c comme séparateur de champs (par défaut, la tabulation).
  - Format de la liste :
    - **n** le $n^{ième}$.
    - **n,m** le $n^{ième}$ et le $m^{ième}$.
    - **n-m** du $n^{ième}$ au $m^{ième}$.
    - **n-** du $n^{ième}$ à la fin.
    - **-n** du 1^{er} au $n^{ième}$.
  - Avec la commande cut, contrairement à sort, le premier champ a comme numéro 1

Gestion de champs



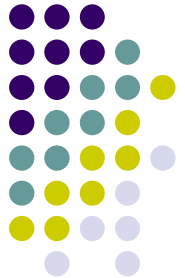
- Soit le fichier carnet-adresse suivant :
  - **cat carnet-adresse**

    maurice:29:0298334432:Crozon
    marcel:13:0466342233:Marseille
    robert:75:0144234452:Paris
    yvonne:92:013344433:Palaiseau
  - **cut –c -10 carnet-adresse** ==> va extraire les 10 premiers caractères de chaque ligne, on obtient :

    maurice:29
    marcel:13:
    robert:75:
    yvonne:92:
  - **cut -d : -f 1,4 carnet adresse** **==>** va extraire le premier et quatrième champ, le : fixant le séparateur de champ. On obtient :

    maurice:Crozon
    marcel:Marseille
    robert:Paris
    yvonne:Palaiseau

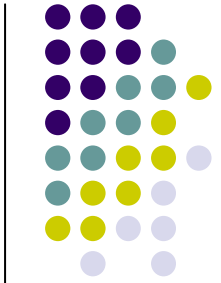
Modification d'un flux



- **tr** : permet de convertir une chaîne de caractère en une autre de taille égale.
  - Options :
    - **table tablefin** transcodage de tous les caractères contenus dans table un à un avec les caractères contenus dans tablefin (de même longueur que table).
    - **-d table** suppression de tous les caractères contenus dans table.
    - **-s table** suppression des occurrences multiples consécutives des caractères contenus dans table.
    - **-c** inversion de table.

  - La commande tr a besoin qu'on lui redirige en entrée un fichier, le résultat de la conversion s'affichant sur la sortie standard.

Modification d'un flux

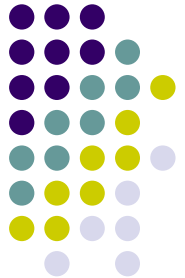


- Table de caractères : les tables de caractères sont à donner entre crochets, avec les facilités suivantes :
  - **A-F** les caractères de A à F.
  - **A*4** le caractère A répété 4 fois.
  - **A*** (dans tablefin seulement) autant de A qu'il faut pour atteindre la longueur de table.
- Exemples :

```
cat toto
aaabbcc cccbbaaa
cat toto | tr [a-c] [A-C]
AAABBCC CCCBBAAA
cat toto | tr -d [ac]
bb bb
```

```
cat toto | tr -s [bc]
aaabc cbaaa
cat toto | tr [a-c] [-*]
------- --------
cat toto | tr -dc [ac]
aaacccccaaa
```

Sed



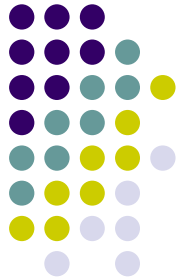
- Éditeur de texte batch (édition automatique d'un texte par une suite d'instructions programmées)
- Les instructions les plus directement utilisables sont la substitution et la suppression :
  - s/expr1/expr2/g *remplace partout expr1 par expr2.*
  - /expr/d *efface toutes les lignes contenant expr.*

- Exemples :

```
cat toto
aaabbcc cccbbaaa

sed s/a//g toto
bbcc cccbb
```

```
cat titi
s/b/bc/g
s/c/de/g
sed -f titi toto
aaabdebdedede dededebdebdeaaa
```

Les shell-scripts sous Unix

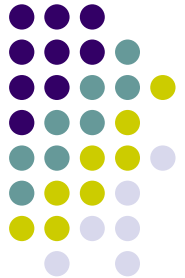


- Le shell Unix est à la fois un interpréteur de commandes interactif et un mini langage procédural autorisant la récursivité

- Ces petits programmes écrits sous shell sont appelés scripts

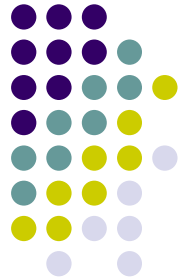
- Les scripts sont des fichiers qui contiennent des commandes que l’on veut exécuter en série dans un shell donné

Les shell-scripts sous Unix



- La connaissance du shell est indispensable au travail de l'administrateur unix :
  - dans de nombreux contextes, on ne dispose pas d'interface graphique
  - le shell permet l'automatisation aisée des tâches répétitives (scripts)
  - de très nombreuses parties du système UNIX sont écrites en shell, il faut être capable de les lire pour comprendre et éventuellement modifier leur fonctionnement

Les shell-scripts sous Unix



- Bash est un logiciel libre, utilisé sur toutes les distributions récentes de Linux et de nombreuses autres variantes d'UNIX

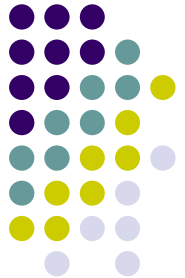
- Le fichier nommé « sauvegarde »:

  - **#!/bin/bash**
  - **BACKDIR=~/`date +%a%d%b%Y`**
  - **echo "Sauvegarde en cours dans $BACKDIR"**
  - **mkdir $BACKDIR**
  - **# Cette ligne est un commentaire.**
  - **cp -v *.xls *.pl Makefile $BACKDIR**

Indique quel est le shell (ou la commande) a exécuter pour interpréter le fichier

- Une fois que le script est écrit, il faut le rendre exécutable avec **chmod +x nom_script**

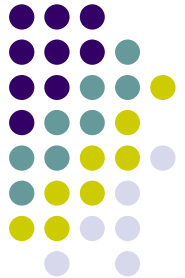
Les shell-scripts sous Unix



- Si le shell se trouve dans une des directories contenues dans la variable PATH, il sera possible de l'exécuter de partout

- **Exemple d'exécution :**
  - **$ sauvegarde**
  - Sauvegarde en cours dans /home/mmdeye/mer25jan2006
  - `result.xls' -> `/home/mmdeye/mer25jan2006/result.xls'
  - `rts.xls' -> `/home/mmdeye/mer25jan2006/rts.xls'
  - `look.pl' -> `/home/mmdeye/mer25jan2006/look.pl'
  - cp: ne peut évaluer `Makefile': Aucun fichier ou répertoire de ce type

Les shell-scripts sous Unix

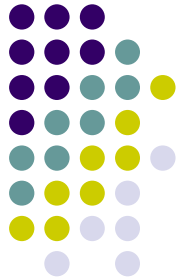


- **Les variables :**
  - **Affectation:** elle se fait en utilisant la syntaxe: **var=valeur**

Attention : pas d'espaces autour du signe égal

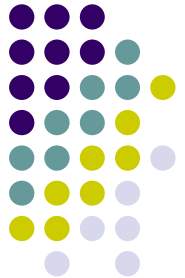
  - **Valeur:** on accède à la valeur de la variable **var** en la faisant précéder de **$** (i.e. avec **$var**).
  - **Exception:** Il peut être nécessaire d'écrire **${var}** afin de différencier les cas suivants:
    - **$variable** contenu de variable
    - **${var}iable** contenu de **var** concaténé avec le texte **iable**

Les shell-scripts sous Unix



- **Évaluation:** ou le bon usage de " , ' et `
  - '**texte**' n'est pas interprété : aucune substitution n'a lieu dans une chaîne délimitée par ce caractère (*apostrophe ou quote*)
  - `**texte**` est évalué et renvoyé : une chaîne délimitée par ce caractère (*accent grave ou backquote*) est interprétée comme une commande et le résultat de la commande lui est substitué
  - "**texte**" n'est pas interprété sauf les parties précédées de **$ \ `**
    - \ (*backslash*) placé devant un caractère spécial lui enlève son interprétation particulière (ignoré devant les autres caractères)

Les shell-scripts sous Unix

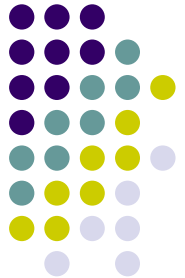


- **Exemples**

```
a=4
echo $a
4

b="pim pam pom"
echo $b
pim pam pom

echo "${a}6 $b"
46 pim pam pom
```

```
c=`date`
echo $c
jeu fév 2 14:27:43 GMT 2006

d='$a > $b'
echo $d
$a > $b
```

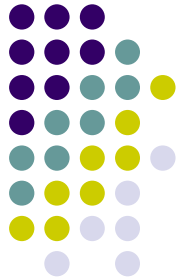
Les shell-scripts sous Unix



- **Opérations arithmétiques :**
  - En shell, toute expression ne contenant que des chiffres peut être utilisée pour du calcul arithmétique. Seules les valeurs entières sont autorisées

  - **Évaluation arithmétique :**
    - **$((expr))** évaluation immédiate de l'expression expr.
      - **a=3**
      - **echo $a + 4**
      - **4 + 3**
      - **echo $((a + 4))**
      - **7**
      - **echo $((a / 2))**
      - **1**

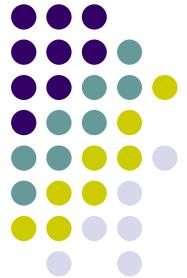
Les shell-scripts sous Unix



- **let var=expr** évaluation suivie d'une affectation.
  - **let a="1+2"**
  - **echo $a**
  - **3**
  - **let b="$a + 4"**
  - **echo $b**
  - **7**
  - **let c="${a}4 + 4"**
  - **echo $c**
  - **38**
  - **let d='a+4'**
  - **echo $d**
  - **7**

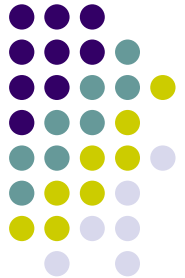
- A noter que si le sens de " ne change pas, le sens de ' est différent.

Les shell-scripts sous Unix



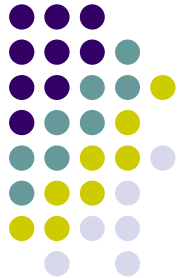
- **Principaux opérateurs**
  - **opérateurs binaires classiques:** `+ - * / % (reste) ** (puissance)`
  - **affectations:** `= += -= /= *= %=`
    - **a=9**
    - **let "a=a+1"**
    - **echo $a**
    - **10**
    - **let "a/=2"**
    - **echo $a**
    - **5**
    - **echo $((a%2))**
    - **1**

Les shell-scripts sous Unix



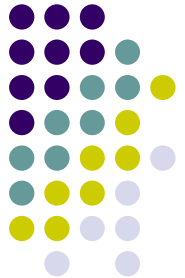
- **Opération sur les chaînes de caractères:**
  - **Opération de manipulation de chaînes**: toutes ces fonctions laissent la chaîne de caractères var inchangée.
    - **${#var}** Renvoie la longueur de la chaîne de caractères var.
    - **${var#exp}** Enlève l'expression exp au début de la chaîne var (plus petit retrait possible avec exp si exp est une expression régulière).
    - **${var##exp}** Idem ${var#exp} mais en prenant le plus grand retrait possible.
    - **${var%exp}** Enlève l'expression exp de la fin de la chaîne var (plus petit retrait possible avec exp si exp est une expression régulière).

Les shell-scripts sous Unix



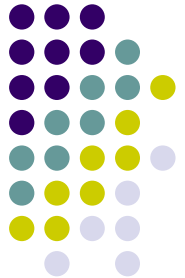
- **${var%%exp}** Idem ${var%exp} mais en prenant le plus grand retrait possible.
- **${var:pos:len}** Extrait de var la portion de chaîne de longueur len commençant à la position pos.
- **${var:pos}** Extrait de var la fin de la chaîne en commençant à la position pos.
- **${var/exp/str}** Retourne la chaîne var en remplaçant la première occurrence de exp par str.
- **${var//exp/str}** Idem mais en remplaçant toutes les occurrences de exp.

Les shell-scripts sous Unix



- **Chaînes de caractères conditionnelles :**
  - **${var :-word}** si var est affecté alors renvoie $var sinon renvoie word
  - **${var :=word}** si var n'est pas affecté alors affecte var à word puis, renvoie $var
  - **${var :+word}** si var est affecté alors renvoie word sinon ne renvoie rien
  - **${var :?word}** si var est affecté alors renvoie $var sinon affiche le message d'erreur word sur stderr et sort du shell-script (exit).

Les shell-scripts sous Unix

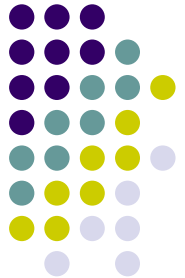


- **Exemples**

```
z="123456789012345"
echo ${#z}
15
echo ${z#123}
456789012345
echo ${z#*2}
3456789012345
echo ${z##*2}
345
```

```
echo ${z%345}
123456789012
echo ${z%3*}
123456789012
echo ${z%%3*}
12
echo ${z:6}
789012345
echo ${z:6:4}
7890
```

Les shell-scripts sous Unix

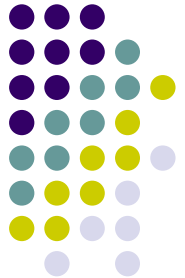


- **Exemples(`z="123456789012345"`)**

```
echo ${z/2/a}
1a3456789012345
echo ${z//2/a}
1a345678901a345
A="abcd"
B=""
echo ${A:-1234}
abcd
echo ${B:-1234}
1234
```

```
echo ${A:+1234}
1234
echo ${B:+1234}

echo ${A:=1234}
abcd
echo ${B:=1234}
1234
echo $B
1234
```

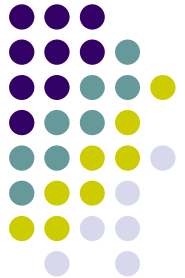
Les shell-scripts sous Unix



- **Exemples**

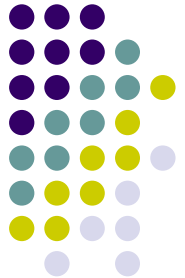
```
C=""
D="/home/mmdeye"
pwd
/
cd ${C:?Invalid directory}
bash: C: Invalid directory
cd ${D:?Invalid directory}
pwd
/home/mmdeye
```

Les shell-scripts sous Unix



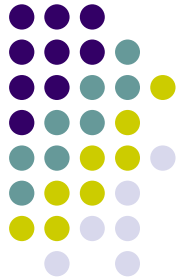
- **Variables prédéfinies dans un shell**
  - Un certain nombre de variables non modifiables sont prédéfinies par défaut dans un shell.
  - **Les variables générales**: parmi celle-ci, on peut citer:
    - **$$** pid du shell.
    - **$PPID** pid du processus père.
    - **$?** valeur de sortie de la commande précédente (0 si terminée avec succès).
    - **$SECONDS** le nombre de secondes écoulés depuis le lancement du script.
    - **$RANDOM** génère un nombre aléatoire entre 0 et 32767 à chaque appel.

Les shell-scripts sous Unix



- **Variables pour le passage d'arguments**
  - Il est possible de passer des paramètres à un shell, et de les exploiter.
  - Pour cela, on utilise les variables spéciales suivantes:
    - **$#** : renvoie le nombre d'arguments.
    - **$@** ou **$*** : l'ensemble des arguments.
    - **$0** : nom du shell (ou de la fonction).
    - **$n** : n^{ième} argument (exemple: $3 est le 3^{ieme} argument).

Les shell-scripts sous Unix



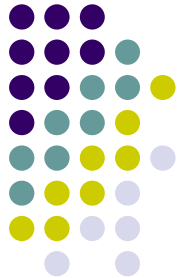
- **Exemples**

```
$ cat testarg
echo "shellscript : $0"
echo "nb arguments: $#"
echo "arguments : $*"
echo "1er argument: $1"
echo "2nd argument: $2"
```

```
$ ./testarg oui 3
shellscript : ./testarg
nb arguments: 2
arguments : oui 3
1er argument: oui
2nd argument: 3
```

- **Pb**: **Écrire** le script "**compiler**" tel que **compiler fichier.c** compile le fichier passé en paramètre, affiche un message de fin de compilation et lance l'exécution du programme.

Les shell-scripts sous Unix

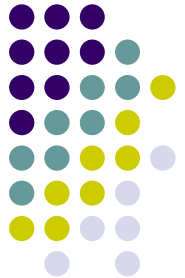


- Par ailleurs, deux fonctions servent à manipuler les arguments:
  - **set** : réaffecte les arguments.
  - **shift p** : décale les arguments vers la gauche (l'argument n devient l'argument n-p, l'argument 0 ne change pas, les arguments de 1 à p sont perdus). Si p n'est pas spécifie, il est égal à 1

```
set `date`
echo $*
jeu fév 2 14:27:43 GMT 2006
echo $6
2006
```

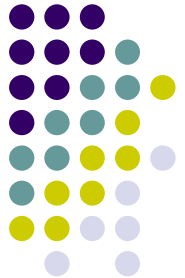
```
shift 3
echo $*
14:27:43 GMT 2006
echo $3
2006
```

Les shell-scripts sous Unix



- **Groupement de commandes:**
  - Grouper des commandes permet de leur affecter un fichier de sortie standard commun
  - On forme un groupe de commandes de deux manières:
    - Exécution dans le shell : **{ commande1;commande2;...; }**
      - la liste de commandes doit se terminer par ';', elle doit être séparée des accolades par des espaces
    - Exécution dans un sous-shell:**(commande1;commande2;...)**
      - Les commandes passées entre parenthèses sont exécutés comme un sous-shell

Les shell-scripts sous Unix



- **Exemples :**

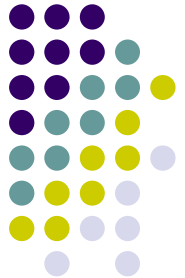
```
pwd;(cd /usr/bin;pwd);pwd
/home/mmdeye
/usr/bin
/home/mmdeye
```

```
pwd;{ cd /usr/bin;pwd; };pwd
/home/mmdeye
/usr/bin
/usr/bin
```

- les variables définies dans le sous-shell disparaissent avec le sous-shell

```
a=3; ( echo $a; a=2; echo $a ); echo $a
3
2
3
```

Les shell-scripts sous Unix

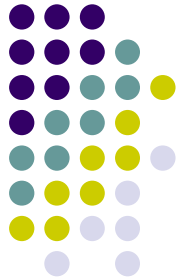


- **Fonctions :**
  - Il est possible dans un shell de définir des fonctions:
  - **syntaxe**: la syntaxe d'une fonction est la suivante:

```
function NOM_FONCTION
{
   CORPS_FONCTION
}
```

  - **arguments:** les arguments ne sont pas spécifiés, mais ceux-ci peuvent être traités en utilisant $#, $* et $n.
  - **portée de la fonction:** elle n'est connue que dans le shell qui la crée (mais ni dans le père, ni dans le fils).
  - **portée des variables**
    - **VAR=valeur** est une variable globale.
    - **typeset VAR=valeur** est une variable locale.

Les shell-scripts sous Unix

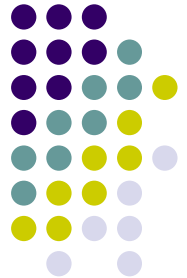


- **Exemples :**

```
$ cat testfun1
function test1
{
echo "Nb arg: $#"
echo "Arg 1 : $1"
echo "Shell : $0"
a=$2
typeset b=$3
echo "in : a=$a b=$b"
}
a=1
b=2
echo "init: a=$a b=$b"
test1 pim pam pom
echo "out : a=$a b=$b"
```

```
$ ./testfun1
init: a=1 b=2
Nb arg: 3
Arg 1 : pim
Shell : ./testfun1
in : a=pam b=pom
out : a=pam b=2
```

Les shell-scripts sous Unix

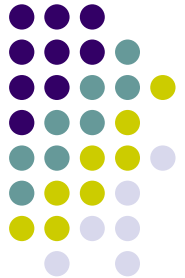


- **Entrée/Sortie**
  - **echo** : affichage sur la sortie standard.
  - **options**:
    - **-n** pas de passage automatique à la ligne.
    - **-e** active l'interprétation des codes suivants (**entre guillemets**).

\a	bip!
\b	backspace
\c	comme -n
\f	saut de page
\n	saut de ligne

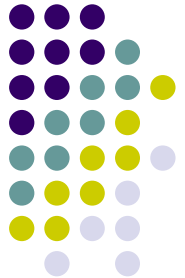
\r	retour chariot
\t	tabulation
\\	caractère \
\nnn	caractère dont le code octal est *nnn*

Les shell-scripts sous Unix



- **read** : lecture à partir de l'entrée standard.
  - si aucun nom de variable n'est spécifié, la valeur lue est mise dans la variable **REPLY**.
  - si plusieurs noms de variables sont spécifiés, l'expression est parsée entre les différentes variables. La dernière variable contient éventuellement la fin complète de l'entrée.
  - read renvoie toujours 0 sauf sur le caractère EOF où 1 est renvoyé. Ceci permet d'utiliser cette fonction pour lire des fichiers.

Les shell-scripts sous Unix



- **Exemples :**

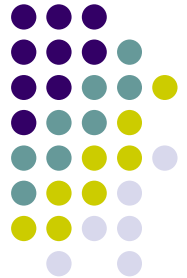
```
$ echo coucou
coucou
$ echo $HOME
/home/pascal
```

```
$ read x
pim pam pom
$ echo $x
pim pam pom
$ read
pom pam pim
$ echo $REPLY
pom pam pim
```

```
$ echo -n 1 ; echo 2
12
$ echo -e "1\t2"
1 2
```

```
$ read x y
pim pam pom
$ echo $x
pim
$ echo $y
pam pom
```

Les shell-scripts sous Unix

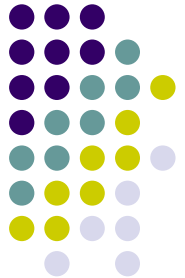


- **Les tests**
  - Il existe en shell trois principales catégories de test:
    - Les tests sur les chaînes de caractères.
    - Les tests sur les entiers.
    - Les tests sur les fichiers.
  - Un test renvoie 0 si le test est VRAI, et une valeur différente de 0 si le test est faux.
  - L'écriture canonique du test d'une expression expr s'écrit:

    `test expr` ou **`[ expr ]`**

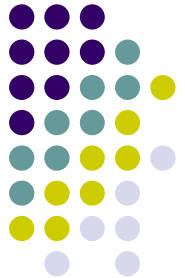
Attention : les espaces entre les crochets et les opérandes et autour de l'opérateur sont **requis**

Les shell-scripts sous Unix



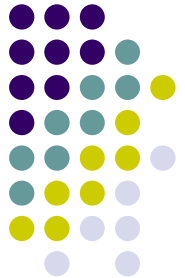
- **Comparaison de chaînes de caractères**
  - **[str1 == str2]** vrai si str1 est égale à str2
  - **[str1 != str2]** vrai si str1 est différente de str2
  - **[str1 \< str2]** vrai si str1 est inférieure à str2 (ordre alphabétique ASCII)
  - **[str1 \> str2]** vrai si str1 est supérieure à str2 (ordre alphabétique ASCII)
  - **[-z str]** vrai si str est nulle.
  - **[-n str]** vrai si str est non nulle.

Les shell-scripts sous Unix



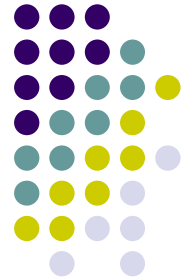
- **Comparaison d'entiers**
  - **[num1 -eq num2]** vrai si num1 est égal à num2
  - **[num1 -ne num2]** vrai si num1 est different de num2
  - **[num1 -lt num2]** vrai si num1 est inférieur à num2
  - **[num1 -le num2]** vrai si num1 est inférieur ou égal à num2
  - **[num1 -gt num2]** vrai si num1 est supérieur à num2
  - **[num1 -ge num2]** vrai si num1 est supérieur ou égal à num2

Les shell-scripts sous Unix



- **Tests sur les fichiers**
  - **test -e *file*** vrai si ***file*** existe
  - **test -f *file*** vrai si ***file*** est un fichier
  - **test -s *file*** vrai si ***file*** est un fichier de taille non nulle
  - **test -d *file*** vrai si ***file*** est un répertoire
  - **test -L *file*** vrai si ***file*** est un lien symbolique
  - **test -r *file*** vrai si l'utilisateur a le droit r sur ***file***
  - **test -w *file*** vrai si l'utilisateur a le droit w sur ***file***
  - **test -x *file*** vrai si l'utilisateur a le droit x sur ***file***

Les shell-scripts sous Unix



- **Combinaisons logiques**

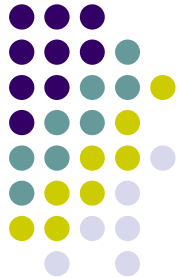
	forme avec [ ]	forme avec test
expr1 ET *expr2*	[*expr1* && *expr2*]	test *expr1* -a *expr2*
expr1 OU *expr2*	[*expr1* || *expr2*]	test *expr1* -o *expr2*

- La négation s'exprime avec le symbole !.

```
$ A=8
$ [ "$A" -gt 5 ] && echo "vrai"
vrai
$ [ ! "$A" -ge 3 ] && echo "vrai"
vrai
```

```
$ B="ab"
$ [ "$B" != "abc" ] && echo "vrai"
vrai
$ [ -n "$B" ] && echo "vrai"
vrai
```

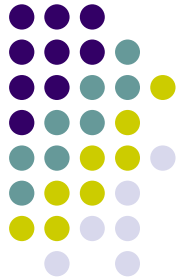
Les shell-scripts sous Unix



- **Exemples :**

```
$ ls -l
-rw-r--r-- 1 Pascal users 1 Sep 7 12:07 toto.c
$ test -f toto.c && echo "vrai"
vrai
$ test ! -x toto.c && echo vrai
vrai
$ ! test -x toto.c && echo vrai
vrai
$ [ "aa" \< "$B" ] && [ "$B" \< "ac" ] && echo "vrai"
vrai
$ [ 3 -le "$A" ] && [ "$A" -ge 10 ] && echo "vrai"
vrai
$ test -f toto.c -o -r toto.c && echo "vrai"
vrai
```

Les shell-scripts sous Unix



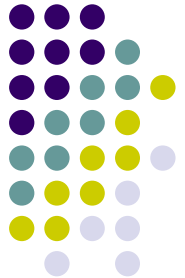
- **Structure de contrôle conditionnelle :**
  - Les structures de contrôle conditionnelles et itératives sont communes à tous les langages de programmation, mais leur syntaxe diffère. En bash :

```
if tst
then
   cmd1
else
   cmd2
fi
```

```
if tst
then
   cmd
fi
```

```
if tst1
then
   cmd1
elif tst2
   then
      cmd2
   else
      cmd3
fi
```

Les shell-scripts sous Unix

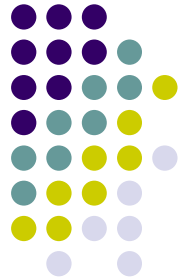


- **Exemples :**

```
$ cat if1
#!/bin/bash
echo -n "$1 est un "
if ! (test -a $1)
then echo "introuvable"
elif test -L $1
then echo "lien symboli"
elif test -f $1
then echo "fichier"
elif test -d $1
then echo "repertoire"
else echo "autre type"
fi
```

```
$ ls -l
total 2
drwxr-xr-x …… Sep 26 00:24 tata
lrwxrwxrwx …… Sep 26 00:24 titi -> toto
-rw-r--r-- …… Sep 26 00:23 toto
$ if1 tata
tata est un repertoire
$ if1 titi
titi est un lien symboli
$ if1 toto
toto est un fichier
$ if1 tutu
tutu est un introuvable
```

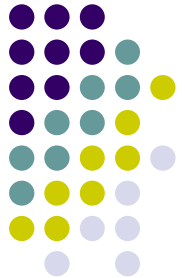
Les shell-scripts sous Unix



- **Exemples :**

```
$ if cd; then echo "succes"; else echo "echec"; fi; pwd
succes
/home/pascal

$ if cd /bachibouzouk; then echo "succes"; else echo "echec"; fi; pwd
bash: cd: /bachibouzouk: No such file or directory
echec
/home/pascal
```

Les shell-scripts sous Unix

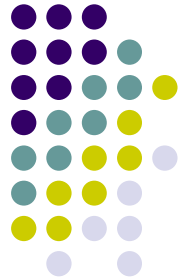


- **case** :

```
case mot in
cas1) cmd1;;
cas2) cmd2;;
...
casn) cmdn;;
esac
```

- Avec
  - **mot** contenu d'une variable, expression arithmétique ou logique, résultat de l'évaluation d'une commande.
  - **casi** expression régulière constante de type nom de fichier. Il est possible de donner plusieurs expressions régulières en les séparant par le caractère **|** (ou). Le cas **default** du C est obtenu en prenant * pour valeur **casn**.
  - **cmdi** commande ou suite de commandes à exécuter si le résultat de **mot** correspond à **casi**.

Les shell-scripts sous Unix

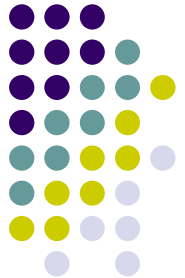


- **Exemples :**

```
$ cat dus
#!/bin/bash
case $# in
0) dir=`pwd`;;
1) dir=$1;;
*) echo "Syntaxe: dus [repertoire]"
exit;;
esac
du -s $dir
```

```
$ dus
143 home/pascal/Universite/StageUnix
$ dus /home/pascal
557392 /home/pascal
$ dus /home/pascal /mnt/cdrom
Syntaxe: dus [repertoire]
```

Les shell-scripts sous Unix



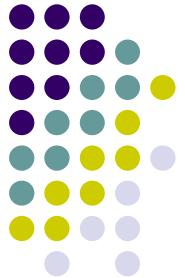
- **Structures de contrôle itératives**
  - **for**
    - syntaxe

```
for var in liste
do
commandes
done
```

    - exemples de listes valides

1 2 3 4 chiffres de 1 à 4
***.c** fichiers C du répertoire courant
$* arguments
`**users**` utilisateurs

Les shell-scripts sous Unix



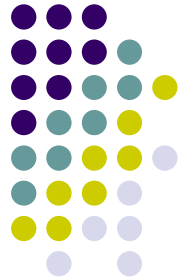
- **Structures de contrôle itératives**
  - **while ou until**
    - syntaxe

```
while COMMAND
do
commandes
done
```

```
until COMMAND
do
commandes
done
```

    - condition d'arrêt
      - COMMAND est une commande renvoyant 0 pour vrai (voir la fonction test).

Les shell-scripts sous Unix

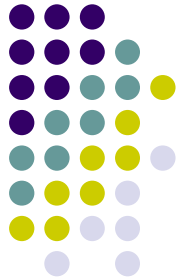


- **Exemples :**

```
$ cat listarg
#!/bin/bash
for u in $*
do
echo $u
done
$ listarg pim pam pom
pim
pam
Pom
```

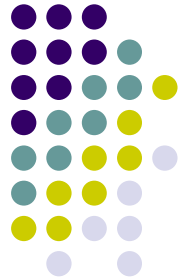
```
$ cat boucle
#!/bin/bash
i=0
while test $i -lt 3
do
echo $i
let i=i+1
done
$ boucle
0
1
2
```

Les shell-scripts sous Unix



- **Contrôle d'exécution**
  - **exit *n*** :
    - elle permet une sortie immédiate du shell, avec l'erreur n.
    - si n n'est pas fourni, n = 0 (sortie normale).
  - **break** :
    - fonction de contrôle de boucle permettant une sortie immédiate de la boucle en cours.
  - **continue** :
    - fonction de contrôle de boucle permettant un passage immédiat à l'itération suivante.

Les shell-scripts sous Unix

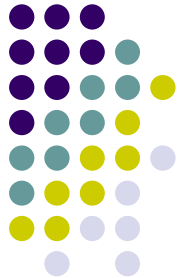


- **Exemples :**

```
$ cat testcont
#!/bin/bash
i=0
while test $i -lt 8
do
let i=i+1
if test $((i%2)) -eq 0
then continue
fi
echo $i
done
$ testcont
1
3
5
7
```

```
$ cat testbreak
#!/bin/bash
i=0
while test $i -lt 8
do
let i=i+1
if test $((i%3)) -eq 0
then break
fi
echo $i
done
echo "Fin“
$ testbreak
1
2
Fin
```

Les shell-scripts sous Unix

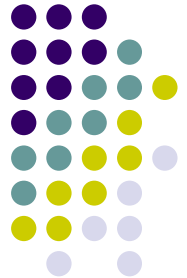


- **Redirections et boucles**
  - Il est possible de conjuguer les redirections < et > pour effectuer des opérations sur des fichiers.

```
$ cat testfor1
#!/bin/bash
for u in 1 2 3
do
echo $u > flist1
done

$ testfor1
$ cat flist1
3
```

```
$ cat testfor2
#!/bin/bash
for u in 1 2 3
do
echo $u
done > flist2

$ testfor2
$ cat flist2
1
2
3
```

Les shell-scripts sous Unix



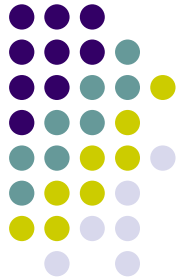
- **Exemples :**

```
$ cat litfic
#!/bin/bash
while read u
do
echo $u
done < flist2

$ litfic
1
2
3
```

```
$ cat litficbad
#!/bin/bash
while read u < flist2
do
echo $u
done

$ litficbad
1
1
(boucle infinie)
```

- Notes
  - Attention à la position des redirections dans la boucle.
  - Pour forcer le **read** à utiliser l'entrée du clavier, utiliser: **read var < /dev/tty**

Les shell-scripts sous Unix



Choix interactif: select

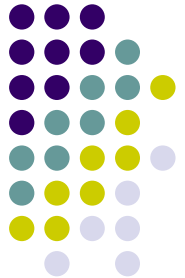
- Cette fonction permet d'effectuer un choix interactif parmi une liste proposée.
- **Syntaxe**

  select *var* **in** *liste*
  do
  commandes
  done

- **Fonctionnement**

  1. La *liste* des choix est affichée sur stderr.
  2. La variable *var* est affectée à la valeur choisie seulement si le choix est valide (non affectée sinon).
  3. Puis, *commandes* est exécutée.
  4. Le choix est reproposé (retour en 1) jusqu'à ce que l'instruction *break* soit rencontrée dans *commandes*.

Les shell-scripts sous Unix



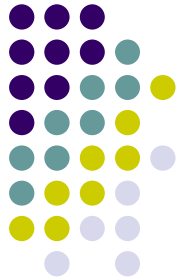
- **Exemples :**

```
$ cat select1
#!/bin/bash
select u in A B C
do
[ -n "$u" ] && break;
done
echo "Le choix est: $u"
$
```

```
$ ./select1
1) A
2) B
3) C
#? zorglub
#? 2
Le choix est: B
$
$
```

- **Option**
  - La variable **PS3** permet de régler la question posée (voir exemples).

Les shell-scripts sous Unix



- **Exemples :**

```
$ cat ./select2
#!/bin/bash
PS3="Votre choix:"
select v in "Ou suis-je?" Quitter
do
case $v in
"Ou suis-je?") pwd;;
"Quitter") break;;
*) echo "Taper 1 ou 2"
esac
done
$
```

```
$ ./select2
1) Ou suis-je?
2) Quitter
Votre choix:1
/home/pascal
Votre choix:3
Taper 1 ou 2
Votre choix:2
$
```