Séance No 1

- Définition, Historique
- Boucle d'interprétation, commandes simples: forme générale
- Erreurs classiques et aide en ligne
- Systèmes de gestion de fichiers (SGF), fichiers, dossiers, arborescence
- droit d'accès, commandes sur les fichiers/dossiers
- processus
- redirections, enchaînements

Définition

- SHELL: programme en mode texte assurant l'interface entre l'utilisateur et le système unix.
- S'utilise :
  - En interactif depuis une fenêtre terminal (xterm, connexion distante texte, ...) : interpréteur de commande
  - Pour réaliser des scripts (fichiers de commandes) : langage de programmation

Shell: où que je clique ?

- On ne clique pas : ça s'utilise avec une souris à 105 touches et sans boule : un clavier :-)
- L'accès est moins immédiat que celui d'une interface graphique
- Plus de liberté/possibilités qu'avec une interface graphique
- Langage de programmation: possibilité d'exprimer des requètes complexes
- Utilisation interactive ou pour écrire des fichiers de commandes (scripts)

Historique

- Les deux shells des origines sont à l'origine de deux familles de shells aux syntaxes incompatibles :
  - Le shell le plus ancien : sh ou Bourne shell écrit dans les années 70 par Steve Bourne. Tout système système unix a un shell /bin/sh qui est un bourne shell (ou un shell compatible);
  - Le csh: écrit à la même époque par Bill Joy incompatible avec le bourne shell mais offrant quelques fonctionnalités supplémentaires (historique des commandes, aliases, contrôle de tâches, ...

Historique (2)

- Ksh: korn shell (David Korn, 1983) sur la base du bourne sh. Le ksh 88 (ou +) est livré avec tous les unix commerciaux. Base de la norme IEEE Posix 1003.2;
- Tcsh: un shell évolué de la famille csh utilisé dans les années 90 comme shell interactif;
- Bash: Bourne Again sh, le shell de la FSF. Compatible posix 1003.2. Le shell de base des distribution linux.
- Zsh: un shell riche en fonctionnalités. Probablement le meilleur choix actuel en interactif.

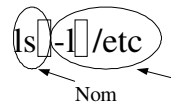
Historique (3)

- POSIX:
- SUS: Single Unix Specification: spécification suivie par les unix commerciaux (et de nombreux non commerciaux) modernes. Proche de la norme POSIX.
- se limiter à SUSv3/POSIX garantit une compatibilité maximale avec les unix utilisés de nos jours
- SUS: http://www.unix.org/what_is_unix/single_unix_specification.html
- De nos jour, il est conseillé d'utiliser un shell compatible posix/sus: ksh, bash et zsh.

Boucle d'interprétation

- Le shell est un programme qui réalise la boucle suivante :
  - Boucle :
    - Lire la ligne de commande
    - Décoder la ligne de commande
    - Exécution de la ligne de commande en créant un processus dans le cas de commandes externes
    - Attendre la fin de l'exécution du processus
    - Retourner en début de boucle

commandes simples: forme générale



- arguments:
  - paramètres optionnels permettant de modifier le comportement de la commande
  - liste des entités auxquelles doit s'appliquer la commande (nom de fichier, processus, utilisateur, ...)
- Exemples:
  - mozilla
  - mozilla -P toto www.univ-evry.fr
  - ls -lrt /etc
  - find ~ -name *.avi -exec rm -f {} \;

quelques commandes simples

- who: liste des utilisateurs ayant une session en cours sur l'ordinateur
- w: idem mais indique aussi ce qu'ils font
- date: date courante
- echo: affiche ses arguments séparés par une espace

Exemples

- Le shell va servir à lancer des commandes internes ou externes
- Exemple de session :

```
#un commentaire commence par #
# lister les fichiers présents
# dans le dossier /etc
ls -l /etc
# liste des utilisateurs connectés
# sur l'ordinateur
who
```

Erreurs

- 5 causes classiques d'erreurs
  1) syntaxe ou chemin incorrect (commande inconnue, ...)
  2)paramètres incorrects (fichier inconnu, ..)
  3)droits d'accès : permission refusée (accès à un fichier, ...)
  4)options invalides (syntaxe des options de la commande)
  5)erreur de conception : le comportement n'est pas celui attendu

•A l'aide

- le manuel
- option « --help » de certaines commandes
- la documentation de votre système d'exploitation ou du programme posant problème
  - souvent /usr/share/doc, /usr/local/share/doc
- recherche sur le WeB: quelqu'un d'autre a forcément déjà eu ce problème

•Le manuel

- dans une version ultérieure de ce document
- les sections du manuel: ràf
- exemples d'utilisation: ràf + exemple dans 2 sections

Systèmes de gestion de fichiers (SGF)

- SGF: mode d'organisation et de stockage des données sur disque;
- Exemples: FAT32, NTFS, ext2fs, ext3fs, reiserfs, UFS, ...
- Les SGF ont des propriétés et fournissent des services variés
- Exemple:
  - les SGF Unix (ext2fs, UFS, ...) : droits sur les fichiers.
  - FAT32: pas de droits d'accès aux fichiers

SGF (suite)

- Les SGF unix fournissent un sous-ensemble commun de fonctionnalités: celui dont nous parlerons.
- Chaque SGF peut fournir plus que ce sous-ensemble
- Fichier unix: fichier disque mais aussi ressource du système (pilote de périphérique, terminal, mémoire, ...)
  - /dev/hda1 : partition 1 du disque 1 (Linux)
  - /dev/kmem: mémoire virtuelle du noyau

Droits d'accès aux fichiers

- 3 types d'accès: lecture (R), écriture (W) et exécution (X)

Objet/Droit	R (lecture)	W (écriture)	X (exécuter)
fichier régulier	lire le contenu	modifier le fichier	exécuter le fichier
dossier	lister le contenu du dossier	modifier le contenu du dossier (y compris destruction de fichier)	utiliser le dossier dans un chemin ou s'y positionner

- 3 classes d'utilisateurs: le propriétaire du fichier, le Groupe du propriétaire du fichier, les Autres utilisateurs.

type fichier	Propriétaire			Groupe du proprio			Autres utilisateurs		
-	R	W	X	R	-	X	R	-	X

- informations dans l'inode, affichage avec « ls »,

Droits d'accès (2): suid, sgid, sticky bit

- 3 autres « droits » spéciaux:
  - bit SUID: le programme s'exécute avec les droits de son propriétaire (au lieu de ceux de l'utilisateur qui le lance)
  - bit SGID: le programme s'exécute avec les droits du groupe propriétaires du fichier
  - sticky bit :
    - sur un fichier exécutable : (obsolète) maintient le fichier en mémoire après l'exécution pour diminuer le temps de la prochaine invocation
    - sur un dossier: seul le propriétaire du fichier a le droit de le supprimer. Exemple: /tmp/

Commandes de base: chmod

- chmod [-R] mode fichier ...
- -R: fichier est un dossier, chmod agit récursivement sur fichier et sur son contenu
- mode:
  - forme numérique: 644
    - pour u: 400 (r), 200 (w) et 100 (x)
    - pour g: 40 (r), 20 (w) et 10 (x)
    - pour o: 4 (r), 2 (w) et 1 (x)
  - forme symbolique: [ugo][+-=][rwxXstguo]

chmod: exemples

- chmod ràf

commande de base: umask

- définit les droits d'accès par défaut d'un fichier
- les droits sont le complément du paramètre d'umask: on laisse tout sauf les droits précisés
- Exemple:
  - umask 002 : mode par défaut: RWXRWXR-X (tout sauf 002)
  - umask 026: mode par défaut: RWXR-X--X (tout sauf 026)
  - umask a=rx,gu+w: mode par défaut: RWXRWXR-X
  - umask -S : affiche le l'état courant sous forme

Commandes de base: chown, chgrp

- chown -R [-H | -L | -P] proprio[:groupe] fichier
- chgrp -R [-H | -L | -P] groupe fichier ...

chown/chgrp: exemples

Commandes de base: ls

Commandes de base: cat

Commande de base: stat

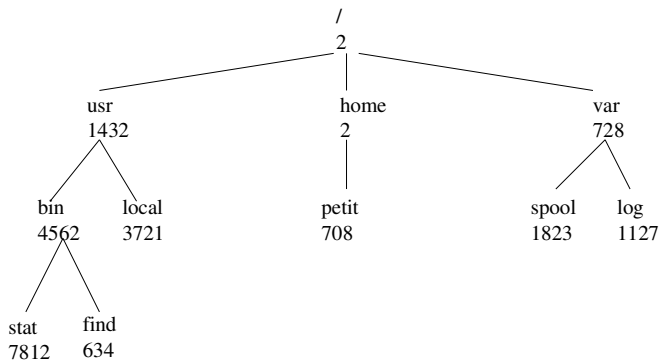
Exemples

- Stat fichier (noter ctime, mtime et atime)
- Cat fichier
- Stat fichier (atime a changé)
- Chmod fichier
- Stat fichier (ctime a changé)
- Modif fichier
- Stat fichier (mtime a changé)

Arborescence

- Sous unix, on a une arborescence unique (donc pas de C:\, D:\, ...comme sous windows)
- Le disque système contient la racine absolue /
- toute l'arborescence est sous cette racine absolue
- Les systèmes de fichiers des autres partitions s'intègrent dans l'arborescence en prenant la place d'un dossier existant
- la racine d'un système de fichier a 2 comme numéro d'inode

arborescence



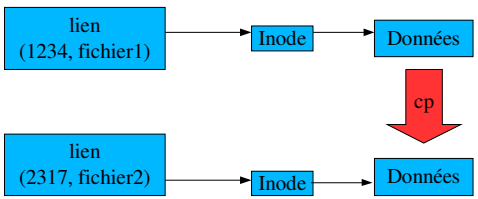
Chemin

- /usr/bin/stat: chemin absolu du fichier stat
- chemin absolu: chemin depuis la racine absolue
- notion de dossier courant
- chemin relatif: chemin depuis le dossier courant

Commandes de base:

- pwd : indique le dossier courant
- cd : changer de dossier courant
- mkdir: pour créer un dossier
- rmdir: détruit les dossiers vides

commande de base: cp



cp fichier1 fichier2

Commandes de base: cp

- copie des données d'un fichier (source) dans un autre (cible) :
  - la cible n'existe pas : création d'un nouvel inode et recopie des données du fichier
  - la cible existe: inode destination inchangée, recopie des données du fichier dans la cible

cp (2)

- gnu cp: en standard sous Linux, installable facilement ailleurs
- fournit des options non standard mais pratiques
- ràf

Commandes de base: rm

- suppression d'un lien d'un fichier ou plusieurs fichiers: `rm [-fiRr] fichier1 ...`
- options:
  - -i: demande de confirmation pour tout fichier à supprimer (aff sur stderr et lecture sur stdin)
  - -f: supprime les messages d'erreur lorsqu'un fichier n'existe pas et supprime la demande d'acquittement si l'utilisateur de rm n'a pas les droits d'écriture sur le fichier à supprimer
  - -R ou -r: supprime récursivement le contenu d'un dossier avant d'appliquer rmdir au dossier.

•rm :exemples

Commandes de base: mv

- mv [-fi] source destination
  - renomme un lien. Source et fichier sont des fichiers réguliers
- mv [-fi] source1 ... destination
  - renomme les sources en les déplaçant **dans** le dossier destination
- la commande fonctionne aussi si sources et destinations sont dans des systèmes de fichiers différents.
- la seconde forme est utilisée si la destination est un dossier existant

mv: exemples

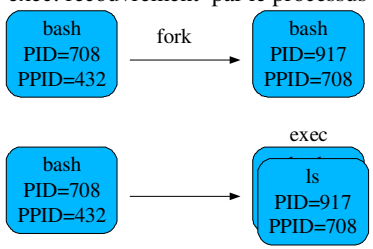
- mv [-fi] source destination
  - renomme un lien. Source et fichier sont des fichiers réguliers
- mv [-fi] source1 ... destination
  - renomme les sources en les déplaçant **dans** le dossier destination
- la commande fonctionne aussi si sources et destinations sont dans des systèmes de fichiers différents.
- la seconde forme est utilisée si la destination est

processus

- un programme: un fichier sur disque
- un processus: un programme en cours d'exécution
  - le code exécutable du programme
  - les données de l'instance en train de s'exécuter
- programme réentrant:
  - deux instances du même programme partagent le même code exécutable
  - elles ont par contre chacune leurs données
- processus système (daemon)/utilisateur

hiérarchie de processus, recouvrement

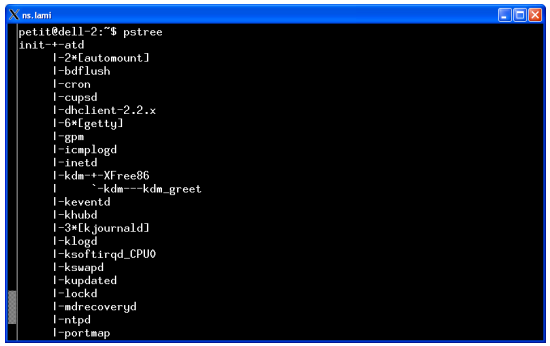
- un processus (processus fils) est toujours créé par un autre processus (processus père):
  - fork: création d'une copie du processus père
  - exec: recouvrement par le processus fils



Hiérarchie de processus

- tout processus a un processus parent sauf le processus initial
- processus initial : init (pid 1)
- arrêter la machine: demander à init d'arrêter tous ses processus fils

pstree



commandes internes/externes

- commande externe: fichier exécutable:
  - création d'un nouveau processus chargé d'exécuter la commande
- commande interne:
  - exécutée par le shell (pas de création de nouveau processus)

type

- type indique si une commande est interne
- options non standard:
  - -a: indique toutes les implémentations
  - -p : indique le chemin de la commande (rien si interne)
- Exemples: testez type sur les commandes suivantes :
  - cd
  - ls
  - pwd

caractéristiques des processus

- statiques
  - PID
  - PPID
  - propriétaire
  - terminal d'attache pour les entreés-sorties
- dynamique
  - priorité
  - nice
  - consommation cpu/mémoire

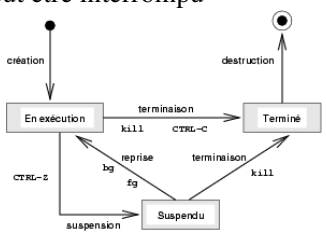
commande ps

- 2 syntaxes:
  - syntaxe Systeme V: option précédées de -
  - syntaxe BSD: options NON précédées de -
  - quelques options SysV:
    - -e ou -A: tous les processus
    - -a: tous les processus associés à un terminal
    - -H: représentation hiérarchique (forêt)
    - -f: format complet;-l: format long (encore plus détaillé)
    - -o: pour modifier le format de sortie (cf manuel)
    - -g, -p, -t, -u: n'affiche que les processus des groupe (-g), processus (-p), terminaux (-t) ou utilisateurs (-u) listés.

commande ps: exemple

Etat d'un processus

- R: exécution
- Z: zombi: il est mort mais son père ne le sait pas
- D: le processus ne peut être interrompu
- S: suspendu
- T: terminé



gestion de processus

- &
- bg
- fg
- jobs
- Ctrl-C
- Ctrl-Z

Signaux

- permettent au système de communiquer avec les processus
- signaux utiles
  - STOP: suspendre
  - CONT: reprendre
  - HUP (1): souvent: relecture configuration
  - KILL(9): tuer sans possibilité de traitement
  - INT(2): équivalent à Ctrl-C: interruption gérable. permet au processus de gérer son interruption
- kill -signal PID

avant plan/arrière plan/détachement

- dans une version future de ce document (ràf)

priorité des processus

- l'exécution des divers processus est gérée par un ordonnanceur (scheduler)
- une priorité est définie dynamiquement
- but: que chaque processus puisse avancer son exécution tout en respectant des priorités
- nice: permet d'influer sur la priorité des processus
  - de 0 à 19 pour un utilisateur
  - de -20 à 19 pour root
  - plus le chiffre est élevé, moins le processus est prioritaire

code de retour

- valeur à laquelle le processus père peut accèder
- 0: terminaison normale
- autre valeur: situation anormale
- commande1 && commande2:la commande2 est exécutée si la commande 1 réussit
- commande1 || commande2: la commande2 est exécutée si la commande 1 échoue
- exemple: commande test
- exemple: construction if/then/else/fi

Entrées-sorties

- Entrées-sorties
  - entrée standard: 0
  - sortie standard: 1
  - sortie d'erreur standard: 2

Héritage

- les descripteurs d'un processus enfant sont initialement les mêmes que ceux du processus père.
- si on ne les modifie pas, en sortie, les affichages s'entrelacent.
- il est possible de changer la valeur de entrée standard et des sorties standard et en erreur
  - pour les rediriger depuis/vers un fichier : <, >, >>, 2>, 2>>
  - pour les rediger depuis/vers un processus : |

redirection des sorties

- >: le contenu du fichier est remplacé par la sortie de la commande
- >>: la sortie s'ajoute à la fin du fichier
- exemples:
  - ls /etc > /tmp/foo.txt
  - cat ls
  - ls /usr/bin >> /tmp/foo.txt
  - du -sk /var/* > /tmp/bar.txt
  - date > /tmp/bar.txt (noter que le No d'inode est inchangé)

redirection de la sortie d'erreur standard

- commande 2> fichier
- commande 2>> fichier
- pratique pour isoler messages d'erreur et sortie
- /dev/null: le trou noir: pour éliminer les messages d'erreur
- du -sk /var/* 2> /dev/null

redirection simultanées

- on peut rediriger plusieurs descripteurs sur une même ligne de commande
- ls > /tmp/f1 2> f2
- les redirections sont traîtées de gauche à droite
- du -sk /var/* > /tmp/resultat 2> /tmp/erreur
- en cas de redirection simultanée avec cette syntase: impérativement vers des fichiers différents

redirection en entrée

- commande < fichier
- exemples:
  - mail petit < texte
  - wc -l < /etc/passwd

redirections avancées

- &1: valeur du descripteur de fichier 1
- &2: valeur du descripteur de fichier 2
- 1>&2: l'entrée standard est redirigée vers le même fichier que la sortie standard
- sert pour des redirection simultanées vers un même fichier
- Exemples:
  - ls > /tmp/test 2>&1 #OK
  - ls 2>&1 >/tmp/test #pas OK: stderr est toujours lié au

<<

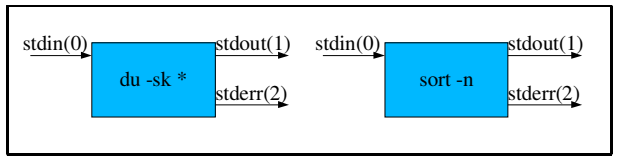
- dans une version ultérieure de ce document

fermeture d'un descripteur

- dans une version ultérieure de ce document

enchaînement de commandes

du -sk * |sort -n



L'ensemble forme une nouvelle commande

Filtres

- commande lisant leurs données sur l'entrée standard et envoyant leur sortie sur la sortie standard
- pratique pour les enchaînements de commandes
- philosophie unix: des commandes simples que l'on combine entre elles

Bilan

- A la fin de cette première séance, vous devez :
  - connaître les notions de systeme de gestion de fichiers, fichiers, dossier, chemin
  - connaître la forme générale d'une commande
  - savoir utiliser le manuel unix
  - savoir vous déplacer dans une arborescence,
  - créer/déplacer/copier des fichiers, des dossiers
  - connaître la notion de processus, d'arborescence de processus, les caractéristiques d'un processus, entrée et sortie standard
  - comprendre la notion d'enchaînement de commandes