Systèmes d'exploitation

Module SysExp

1er cours - Introduction

Qu'est-ce qu'un SE?

Un système d'exploitation est un ensemble de programmes qui coordonnent le fonctionnement des différents composants matériels et logiciels d'un système informatique.

Composants principaux

Noyau

fichiers — processus

périphériques — protocoles réseau

sécurité — mémoire

Bibliothèques

Outils systèmes

Quelques systèmes

UNIX

Windows

95 *98* *me* *seven* *2000* *vista* *xp*

BSD

puma *panther* *tiger* *leopard* *snow leopard*

Mac OS

cheetah *9* *8*

BeOS

Linux

debian *fedora* *mandriva* *knoppix* *suse* *redhat* *ubuntu* *gentoo*

Solaris

Propriétés d'UNIX

- Multi-utilisateurs et Multitâches
- Exécution simultanée (temps partagé) et protégée de processus
- Processus réentrants
- Système de fichiers hiérarchique

Propriétés - suite

- Entrées-Sorties intégrées au système de fichiers => Transparence de l'accès aux périphériques
- Gestion de la mémoire virtuelle
- Interface utilisateur interactive (shell)
- Beaucoup d'utilitaires
- Très portable

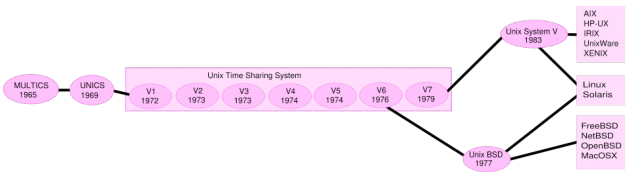
D'où vient UNIX?

septembre 1969

UNICS

http://www.levenez.com/unix/history.html

Évolution résumée



Résumé historique

- 1968 - Ken Thompson (Bell Laboratories) crée le système Multics qui deviendra Unics, puis UNIX.
- Ce système résolvait bon nombre de désavantages des systèmes de l'époque.
- Grâce à sa portabilité le système s'est vite répandu dans les entreprises et universités.
- Des milliers de versions différentes ont vu le jour, dont les branches LINUX, Solaris, BSD et Mac OS X.

Linux

- Système UNIX OpenSource
- Beaucoup de distributions disponibles
- Communauté très active
- De plus en plus simple à aborder

Structure des disques

- Pour être utilisable par un système d'exploitation, un disque doit être structuré selon un standard.
- Aujourd'hui il existe plusieurs standards:
  - MBR (Master Boot Record)
  - GPT (GUID Partition Table)
  - APM (Apple Partition Map)
- Le plus répandu (et aussi le plus dépassé) est le MBR, utilisé par défaut sur Windows et Linux

Master Boot Record

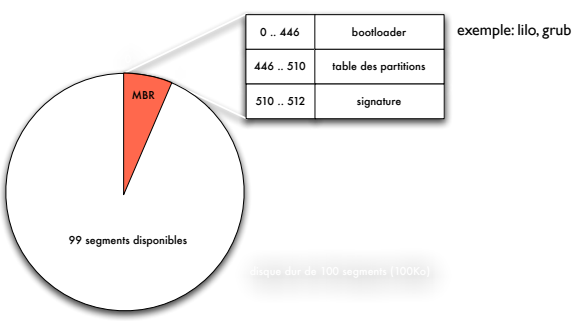
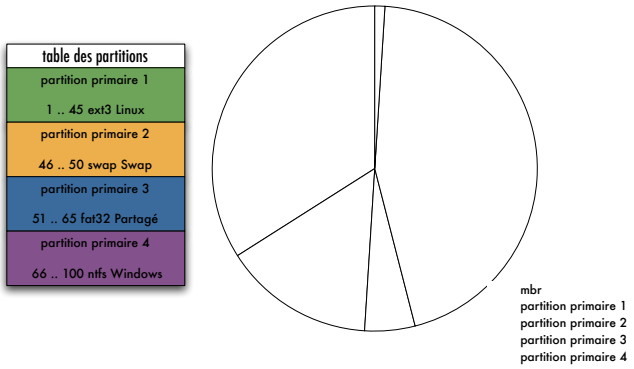


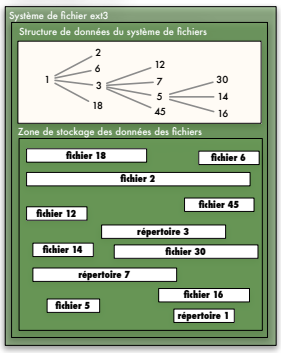
Table des partitions



Limitations du MBR

- Dans sa définition initiale, il ne prévoyait que 4 partitions par disque (les 4 primaires).
- Le concept de partition secondaire a été ajouté pour étendre cela. Ainsi, une partition primaire peut être coupée en plusieurs partitions (ajoutant ainsi une sous-table dans chaque partition primaire)
- Ce format sera vraisemblablement à terme remplacé par le GUID qui est beaucoup plus flexible et générique.

Structure d'une partition



Debian

- Système Linux réputé pour sa stabilité et sécurité
- Mise à jour majeures très peu fréquentes (~2 ans)
- Contrôles très sévère de la qualité du système
- Version actuelle: 6.0 « squeeze »
- http://www.debian.org/CD/netinst/

Ubuntu

- Distribution basée sur Debian
- Mise à jour tous les 6 mois
- Disponible en “Live CD”
- Orientée grand public
- Version actuelle 12.04 LTS (avril 2012)
  « Precise Pangolin »
- http://www.ubuntu.com/server/get-ubuntu/download

expérimentation

Systèmes d'exploitation

Module SysExp

2ème cours - UNIX

Système de fichiers UNIX

- Objectifs principaux
  - assurer la subsistance des données sur un support informatique (aujourd'hui disques durs) en rendant transparent l'accès aux blocs de données
  - hiérarchisation des informations de manière à faciliter l'enregistrement et l'accès aux données
  - système permettant d'assurer la sécurité de l'accès aux données, par l'utilisation de droits d'accès
- Exemples: ufs, ext3, reiserfs, hfs+, zfs

Principes d'un File System

- Le FS est un ensemble de méthodes (fonctions, procédures) qui permettent au système d'exploitation de stocker des fichiers
- Mais c'est surtout une structure de données (table des inodes), stockée au tout début de chaque partition d'un disque dur
- Cette structure est une sorte de carte géographique qui indique ce que contient chaque bloc de la partition du disque dur

Principes d'un File System

- Un File System Unix manipule des inodes
- Un inode permet au FS de faire correspondre un ou plusieurs fichiers (le côté utilisateur) à un ou plusieurs blocs sur le disque dur (le côté matériel)
- Lorsqu'un nouveau fichier est créé dans le système, un numéro de inode libre est réservé et à ce numéro sont associés le/les blocs correspondants

Exemple de table d'inodes

n°	blocs
1	43, 10, 20, 7
2	3
3	11, 13, 45
4	
5	8, 18, 21, 37, 25
6	5, 41
7	
...	

Fichiers sous UNIX

- Fichiers ordinaire
- Répertoires
- Fichiers spéciaux pour l'accès aux périphériques
  - fichiers de type bloc
  - fichiers de type caractères
- Liens symboliques & Pipes nommés (fifo)

Fichier ordinaire

- Contient des données de type caractère, sous forme d'octets
- Ces données sont stockées physiquement sur le disque dans les blocs de données, par "tranches" de 1024 octets (taille d'un bloc)
- Grâce à la structure de données du FS (inodes), le système d'exploitation peut savoir où sur le disque (dans quels blocs) se situe chaque partie du fichier

Répertoire

- Contient une liste associant à chaque ligne un nom avec un numéro d'inode
- C'est donc l'appartenance d'un fichier ordinaire (par son inode) à un répertoire qui permet de définir le nom de ce fichier
- Un même fichier ordinaire peut donc être contenu dans plusieurs répertoires différents (liens hard)
- Un répertoire peut bien évidemment contenir d'autres répertoires (structure hiérarchique)

Exemple de répertoire

n° inode	nom du fichier
24	.
23	..
26	fichier_1.txt
5	fichier_2.xml
26	fichier_5-idem_1.txt
27	image_1.png
18	répertoire_1
35	script_1.sh

Fichiers spéciaux

- Au lieu de faire référence à des blocs sur le disque dur, les fichiers spéciaux (type bloc ou caractères) permettent l'accès direct à des adresses mémoire du système
- Ils permettent d'avoir accès aux gestionnaires de périphériques du système
- Cette méthode permet de garantir le postulat à la base de UNIX: tout est fichier

Liens symboliques & pipes nommés

- Les liens symboliques sont les correspondants des alias (Mac OS) et des raccourcis (Windows)
- Ils permettent de contourner certaines limitation des liens hard
- Les pipes nommés permettent principalement de faire des échanges de données entre des processus par l'intermédiaire de fichiers nommés

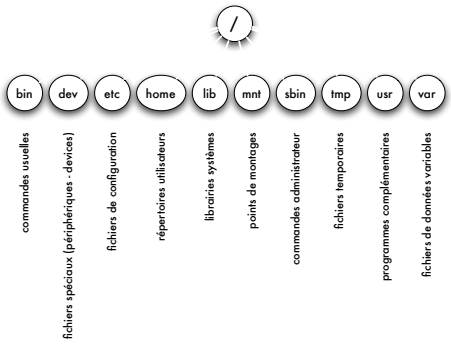
Droits d'accès

- Pour garantir la sécurité des données, il existe les droits d'accès
- Ils sont séparés en trois niveaux distincts:
  - les droits du propriétaire (User)
  - les droits du groupe (Group)
  - les droits de tout le monde (Other)
- Ils peuvent être de 3 types différents:
  - lecture (Read) écriture (Write) et exécution (eXecute)

Droits d'accès

	Fichiers	Répertoires
Lecture	autorise la lecture du fichier	permet de lister le contenu du répertoire
Écriture	permet la modification du fichier	permet la création et la suppression de fichiers du répertoire Attention: cette permission est valable quels que soit les droits des fichiers!
Exécution	autorise l'exécution du fichier (script ou programme)	permet de se positionner dans le répertoire

Hiérarchie standard UNIX

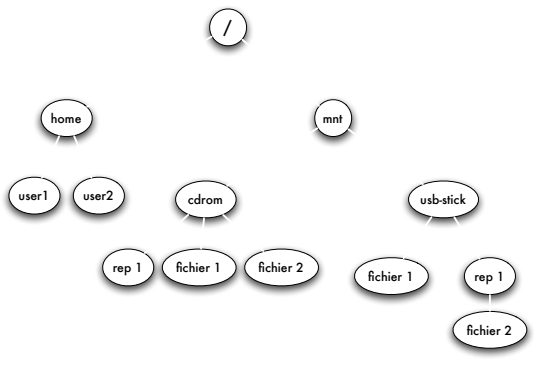


http://www.pathname.com/fhs/pub/fhs-2.3.html

Hiérarchie UNIX

- Sous Unix, il n'y a qu'une seule hiérarchie de fichiers. Tout part de la racine (root): /
- Lorsqu'un volume est monté sur un système Unix, il s'ajoute à la hiérarchie de base, généralement dans /mnt
- Tous les fichiers du système sont donc accessibles par la racine.

Volumes montés



conventions de nommage

- sous UNIX, les noms de fichiers peuvent contenir théoriquement n'importe quel caractère.
- cependant, pour éviter d'être coincé, les espaces et caractères accentués ne sont généralement jamais utilisés.
- un fichier commençant par '.' est considéré comme invisible => pas affiché par la commande ls (il faut utiliser l'option -a)

Comment interagir?

- Par une interface graphique
- Par une ligne de commande (shell)

Interface graphique

- Exemples: KDE, Gnome, etc.
- Avantages:
  - très user-friendly
  - pas besoin de connaître la structure du système (principes génériques communs à tous les OS)
- Désavantages:
  - accès à distance difficiles (vnc)
  - pas beaucoup de liberté de configuration

Ligne de commande (Shell)

- Exemples: sh, bash, ash, zsh, ksh, csh, tcsh,
- Avantages:
  - accès direct au kernel
  - accès à distance très aisé
  - contrôle de toutes les facettes du système
- Désavantages:
  - phase d'apprentissage indispensable

Principes d'un Shell

- Par un login (utilisateur + mdp) permet d'avoir accès au système
- Lors du login, l'utilisateur se retrouve dans son dossier utilisateur, à l'intérieur du système de fichiers
- Basé sur un système de commandes (internes et externes)
- Mise en place d'un dialogue dynamique entre l'utilisateur et la machine

Fonctionnement

- Un shell est un interpréteur de commande
- Une commande est structurée ainsi:

  commande [argument(s)] <retour à la ligne>

- Chaque commande à une syntaxe propre qu'il faut apprendre à connaître

man

- man
  - permet d'obtenir le mode d'emploi d'une commande
- Exemples:
  - man ls => affiche la page de manuel de la commande 'ls'
  - man man => affiche la page de manuel de la commande man

pwd

- affiche la position actuelle dans le système de fichiers
- lors du login, la position actuelle est automatiquement le répertoire home de l'utilisateur loggé
- Exemple: /home/benoit

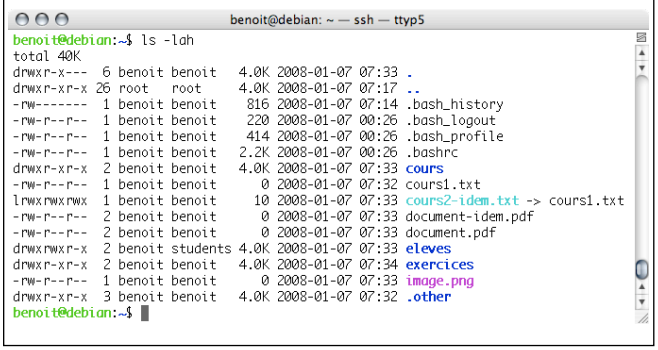
cd

- permet de se déplacer dans l'arborescence
- positionnement relatif / absolu
- Exemples:
  - cd se positionne dans le home
  - cd ~ idem
  - cd - revient à l'endroit précédent
  - cd /home se positionne de manière absolue dans le répertoire "home" situé juste en dessous de la racine
  - cd benoit se positionne de manière relative dans le répertoire "benoit" du répertoire courant (home)

ls

- permet de lister le contenu du répertoire courant
- options importantes:
  - -l affiche sous forme de liste avec toutes les informations des fichiers
  - -a affiche même les fichiers invisibles
  - -i affiche les numéros d'inode des fichiers
  - -h affiche les tailles de fichiers "humainement" (en octets, Ko, Mo, Go, etc.)

listage de fichiers



mkdir

- permet de créer un répertoire
- options utiles:
  - -p crée les répertoires intermédiaires s'ils n'existent pas
- Exemple:
  - mkdir -p test/rep1/rep2

rmdir

- permet de supprimer un répertoire
- cette commande ne fonctionne que si le répertoire est vide (ne contient aucun autre fichier)
- options utiles:
  - -p supprime les répertoires intermédiaires s'ils sont vides
- Exemple:
  - rmdir -p test/rep1/test2

rm

- permet de supprimer un fichier
- options utiles:
  - -r permet de supprimer recursivement des fichiers / répertoires
  - cette option est très puissante et dangereuse car elle force les répertoires non-vides à être supprimés!
- la commande rm -r / executée en tant que root efface tout l'ordinateur!!!

mv

- permet de déplacer ou renommer un fichier
- ne sont autorisées que les manipulations sur des fichiers dont les droits correspondent à ceux de l'utilisateur.

cp

- permet de copier des fichiers ou répertoires d'un endroit à une autre
- option importantes:
  - -r fait une copie récursive (pour les répertoires)
  - -v affiche tous les fichiers copiés sur le shell

ln

- permet de créer un lien d'un fichier
- exemple: ln fichier1.txt fichier2.txt

- option importantes:
  - -s fait un lien symbolique
- exemple: ln -s /etc/passwd utilisateurs

exercices

- login en ssh sur serveur debian
- pour Mac
  - utiliser ssh & scp
- pour Windows:
  - télécharger putty
  - http://www.chiark.greenend.org.uk/~sgtatham/putty
    - PuTTY et PSCP

Systèmes d'exploitation

Module SysExp

3ème cours - UNIX

Commandes supplémentaires

pour toutes les commandes, consultez les pages MAN!

chmod

- permet de changer les droits d'un fichier
- 2 mode possible:
  - numérique
  - symbolique
- options utiles:
  - -R permet de modifier récursivement

chmod numérique

- s'utilise en additionnant, pour chaque niveau de droits, les différents bits:

bit	droit
4	lecture
2	écriture
1	exécution

- Exemples:
  - chmod 777 rep => rwxrwxrwx rep
  - chmod 644 fichier.txt => rw-r--r-- fichier.txt

chmod symbolique

- permet, sur un niveau spécifique, d'ajouter un droit (+), supprimer un droit (-), définir un droit en fonction de l'umask (=).
- les niveaux sont symbolisés par 'ugo' (user, group, other) ou 'a' pour tous.
- Exemples:
  chmod g-rw file.txt => enlève rw au groupe
  chmod ugo+rw file.txt => met tous à rw
  chmod a+rw file.txt => idem
  chmod =rw file.txt => met les droits du fichier d'après les droits par défaut (rw-r--r--)

umask

- permet de consulter / définir, le masque de création des fichiers / répertoires.
- le umask est utilisé par le système lorsqu'un nouveau fichier est créé pour "décider" des droits par défaut.
- il se définit par un nombre de 3 chiffres, chacun de ces chiffres étant le complément à 7 (pour les répertoires) et à 6 (pour les fichiers) du droit désiré.
- Exemple: umask 022 (par défaut) signifie que les fichiers créés auront le droit 644 et les répertoires 755

chown

- permet de changer le propriétaire d'un fichier.
- options utiles:
  - -R va appliquer le changement de manière récursive
- Exemple: chown user fichier.txt
  - => va changer le proriétaire de fichier.txt en mettant l'utilisateur 'user' à la place de l'utilisateur actuel

chgrp

- permet de changer le groupe d'un fichier.
- options utiles:
  - -R va appliquer le changement de manière récursive
- Exemple: chgrp etudiants fichier.txt
  - => va changer le groupe de fichier.txt en mettant le groupe 'etudiants' à la place du groupe actuel

cat

- permet de concaténer et afficher le contenu de fichiers.
- options utiles:
  - -n numérote les lignes
  - -s supprime les lignes vides
- Exemples:
  - cat -n /etc/passwd
    affiche la liste des utilisateurs numérotée
- cat sans paramètres permet d'introduire des données dans la commande

echo

- permet d'introduire et d'afficher des données sur le terminal. Par défaut une retour à la ligne est ajouté à la fin des données.
- options utiles:
  - -n permet de ne pas mettre de retour à la ligne à la fin.
- Exemples:
  - echo "Hello world!"
    affiche Hello world! sur la sortie standard

head & tail

- permet d'afficher respectivement la tête (début) et la queue (fin) d'un fichier donné.
- options utiles:
  - -n permet de spécifier le nombre de lignes à afficher
  - -f (pour tail) permet d'attendre indéfiniment l'ajout de nouvelles lignes
- Exemples:
  - tail -f /var/log/messages.log
    affiche le contenu du fichier de log système

more & less

- permettent d'afficher le contenu d'un fichier, sans fonctionnalités d'édition.
- more permet juste de paginer le contenu alors que less permet en plus de faire des recherches et de naviguer dans le contenu.
- commandes utiles de less:
  - :q quitter
  - ctrl-b / ctrl-f page précédente / page suivante
  - /expr recherche l'expression 'expr' dans le texte
  - :nn aller à la ligne nn
  - :nn% aller à nn% dans le fichier

vi & nano

- sont des éditeurs de texte (indispensables si on souhaite faire des modifications dans des fichiers)
- vi est le plus utilisé mais assez compliqué à utiliser au départ
- nano (ou pico) est beaucoup plus simple d'accès

éditeur vi

- :h page d'aide
- ESC basculer entre le mode d'insertion et le mode d'entrée de commandes
- i insertion avant le pointeur
- A insertion à la fin de la ligne
- o nouvelle ligne
- v mode de sélection de zone de texte (visual)
- y stocke la sélection dans le tampon
- d supprime la sélection
- x efface la lettre/sélection courante et la stocke dans le tampon
- dd efface la ligne courante et la stocke dans le tampon
- yy stocke la ligne courante dans le tampon
- p colle les données présentes dans le tampon
- ctrl-b / ctrl-f une page en arrière / en avant
- u annuler la dernière opération
- b / w un mot en arrière / en avant
- /expr recherche l'expression 'expr' dans le texte (puis / pour continuer...)
- :nn aller à la ligne nn
- :wq sauvegarder et quitter
- :q! quitter sans sauvegarder

éditeur nano (pico)

- ctrl-g page d'aide
- ctrl-y page précédente
- ctrl-v page suivante
- ctrl-k couper la ligne
- ctrl-u coller la ligne
- ctrl-o enregistrer le fichier
- ctrl-x quitter
- ctrl-c affiche des infos sur la position courante

find

- permet de rechercher des fichiers dans le système, en se basant sur différents critères.
- syntaxe: find chemin -critère valeur
- quelques critères:
  - -name cherche sur le nom de fichier
  - -group cherche d'après le groupe du fichier
  - -mmin cherche d'après la dernière date de modification
  - -perm cherche d'après les droits des fichiers (permissions)
  - -print affiche les fichiers trouvés sur le terminal
  - -size cherche d'après la taille du fichier
  - -type cherche d'après le type du fichier
  - -user cherche d'après le possesseur du fichier
  - -exec permet d'exécuter une commande sur chaque élément trouvé
- Exemple:
  - find . -name "*.txt"
    va chercher dans le répertoire courant (.) tous les fichiers se terminant par ".txt"

du

- permet de calculer la taille utilisée par un répertoire.
- options utiles:
  - -h affiche en unités "human-readable" au lieu de blocs
  - -s affiche uniquement le répertoire spécifié
- Exemple:
  - du -hs ~
    affiche la taille du répertoire home

grep

- permet de faire une recherche par contenu dans des fichiers. Retourne les lignes correspondant à l'expression spécifiée.
- syntaxe: grep expression chemin
- options utiles:
  - -i ignore la casse
  - -r effectue une recherche récursive
  - -v retourne les lignes ne correspondant pas à l'expression
- Exemple:
  - grep -r hello ~
    va rechercher dans home toutes les fichiers contenant "hello" et afficher toutes les lignes correspondantes dans ces fichiers

wc

- permet de compter le nombre de lignes, mots, caractères d'un fichier.
- options utiles:
  - -c compte le nombre de caractères
  - -w compte le nombre de mots
  - -l compte le nombre de lignes

cut

- permet de couper des portions de fichiers structurés en champs.
- options:
  - -f la liste des champs à couper
  - -d le délimiteur utilisé dans le fichier
- Exemple:
  - cut -f 1,5 -d : /etc/passwd
    va afficher les 1ères et 5èmes colonnes de /etc/passwd

sort

- permet de trier les lignes des fichiers spécifiés
- options utiles:
  - -b ignorer les espaces au début des lignes
  - -n trie numériquement
  - -r inverse l'ordre de tri
  - -u supprime les doublons
  - -t permet de spécifier le séparateur des champs
  - -k permet de spécifier les champs (keys) à trier
- Exemple:
  - sort -n -k3 -t: /etc/passwd
    va trier /etc/passwd numériquement sur *les 3èmes colonnes*

tar

- permet de gérer des archives de fichiers
- options utiles:
  - -c permet de créer une nouvelle archive
  - -t permet de lister le contenu d'une archive
  - -x permet d'extraire une archive
  - -f permet de spécifier le fichier à traiter (source ou destination)
  - -z permet de dé-/compresser une archive avec gzip
  - -j permet de dé-/compresser une archive avec bzip
- Exemples:
  - tar -czf archive.tar.gz un_repertoire
    => va archiver et compresser le répertoire 'un_repertoire' dans 'archive.tar.gz'
  - tar -xzf archives.tar.gz
    => va désarchiver et décompresser 'archive.tar.gz' dans le répertoire courant.

Plus loin avec les commandes...

auto-completion

- les interpréteurs actuels proposent tous des fonctions d'auto-completion de la ligne de commande.
- cette fonction permet d'éviter de taper systématiquement toutes les lettres d'un path.
- l'auto-completion s'utilise en appuyant sur la touche tabulation. Lorsque la touche est appuyée, le shell affiche à l'écran les noms des fichiers qui correspondent au texte déjà entré.

métacaractères

- dans la plupart des interpréteurs il est possible d'utiliser des métacaractères (wildcards) pour spécifier des paths de fichiers.
- il existe 3 métacaractères fréquemment utilisés:
  - ? remplace un caractère
  - * remplace 0 ou n caractères
  - [...] définit un ensemble de caractères

métacaractère '?'

- permet de remplacer un caractère dans la ligne de commande.
- Exemples:
  - ls /home/benoit/exercices/exercice_?
    affichera tous les fichiers/répertoires de "exercices" qui commencent par "exercice_" et se terminent par un caractère
  - ls /home/benoit/exercices/serie_?_exercice_?.pdf
    affichera tous les fichiers/répertoires de "exercices" qui commencent qui ont numéro de série et un numéro d'exercice à un caractère.

métacaractère '*'

- permet de remplacer 0 ou plusieurs (n) caractères dans la ligne de commande.
- Exemples:
  - ls -l ~/exercices/*.pdf
    affichera tous les fichiers/répertoires de "exercices" qui ont l'extension ".pdf" (par convention, ce ne sera que des fichiers).
  - ls -l ~/cours/cours*
    affichera tous les fichiers/répertoires de "cours" qui ont commencent par "cours".
  - ls -l /home/m*/exercices/*.txt
    affichera tous les fichiers/répertoires de "exercices" qui ont l'extension ".txt" et dont le nom du dossier utilisateur contenant commence par "m".

métacaractère [...]

- permet de spécifier un ensemble de caractères
- il peut être utilisé de deux manières:
  - [aeiouy] énumération de caractères
  - [a-z] intervale de caractères
  - [!a-z] ou [!aeiouy] négation des deux points précédents
- Exemples:
  - ls -l /home/[!aeiouy]*
    affichera les utilisateurs dont le nom ne commence pas par une voyelle minuscule
  - ls -l /home/[m-z]*
    affichera les utilisateurs dont le nom commence par une minuscule supérieure à m

protection des caractères

- selon les commandes, l'utilisation de certains caractères peut poser problème. En effet, ces caractères spéciaux sont réservés:
  ? * [ ] < > $ { } ' " | ` ( ) \ espace
- ils ne peuvent normalement pas être utilisés pour autre chose que leur sens initial. Pour cette raison il peut être utile de protéger ces caractères. Il y a 3 niveaux de protection offerts:
  - le backslash '\', qui permet de protéger un seul caractère en annulant sa fonction spéciale
  - les guillemets "...", qui annulent tous les caractères sauf $ \ `
  - les apostrophes '...', qui annulent l'ensemble des caractères spéciaux

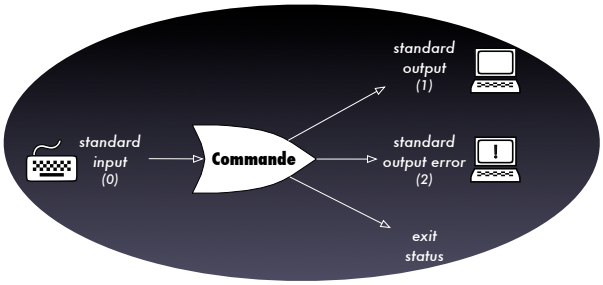
protection des caractères

- Exemples:
  - cat voici\ l\'\ exemple\ de\ texte\ 1.txt
    les espaces, qui normalement séparent les paramètres sont protégés par le backslash. Ainsi la commande ne reçoit qu'un seul paramètre. L'apostrophe est aussi protégé par un backslash
  - cat "voici l'exemple de texte 1.txt"
    tout le paramètre est protégé en une seule fois grâce aux guillemets
  - cat 'voici l\'exemple de texte 1.txt'
    idem que le précédent mais avec les apostrophes (du coup il est tout de même nécessaire de protéger l'apostrophe interne.
- Ces 3 moyens de protections sont tout à fait complémentaires. Aucun d'eux n'est meilleur qu'un autre, leur efficacité dépend entièrement du contexte.

principe des commandes

- une commande a des entrées et des sorties.
- par défaut une commande a une entrée standard, une sortie standard et une sortie d'erreurs.
- ces entrées/sorties sont des périphériques. Comme sous UNIX «tout est fichier», ces entrées / sorties sont en fait des fichiers du système.
- en plus des entrées/sorties, une commande retourne aussi un code de sortie (exit status) qui permet au système de connaître l'état de l'exécution.

entrées / sorties standard



Périphérique	Association par défaut	Descripteur de fichier
standard input buffer	clavier	0
standard output	écran	1
standard output error	écran	2

entrées / sorties manuelles

- il est possible de contrôler manuellement les associations des entrées / sorties d'une commande et redirigeant celles-ci vers d'autre fichiers que les fichiers standards. Cette opération s'effectue par l'intermédiaire des opérateurs de redirection.
- il est aussi possible d'effectuer directement des chaînage entre différentes commandes. Cela consiste à faire des connections entre leurs entrées / sorties respectives pour faire transiter entre elles des données. Cette opération s'effectue en utilisant les pipes.

redirections d'entrée

- < nom_fichier
  prend comme entrée le fichier spécifié
- Exemple:
  - cat < /etc/passwd
- << mot
  prend comme entrée toutes les lignes tapées au clavier jusqu'à celle qui contient mot (uniquement)
- Exemple:
  - more << fin

redirections de sortie

- > nom_fichier
  envoie la sortie dans nom_fichier. Si celui-ci existe, il est écrasé
- Exemple:
  - ls -l ~/exercices > /tmp/listing.txt
- >> nom_fichier
  rajoute la sortie à la fin de nom_fichier (concaténation). Si celui-ci n'existe pas, il est créé
- Exemple:
  - echo salut >> /tmp/listing.txt

chaînage de commandes

- le chaînage s'effectue par l'intermédiaire du caractère '|' appelé pipe.
- comme vu précédemment, il fait office de "tuyau" entre deux commandes
- c'est une fonctionnalité très puissante des systèmes UNIX.
- Exemple:
  - ls -l /etc | cat -n | less
    va lister /etc, numéroter les lignes et afficher le résultat avec less, qui permet d'y naviguer.

séquence de commandes

- il existe plusieurs moyens de faire des séquences de commandes pour les lancer successivement depuis la même ligne de commande
- cmd1; cmd2; ...; cmdN
  permet d'exécuter indépendamment les N commandes
- cmd1 || cmd2 || ... || cmdN
  permet d'exécuter les N commandes sous condition d'erreur (d'après le status de sortie - exit status)
- *cmd1 && cmd2 && ... && cmdN*
  permet d'exécuter les N commandes sous condition de réussite

système multi-tâche

- comme nous l'avions vu, l'un des objectifs d'UNIX est d'offrir un système multi-tâches, permettant l'exécution simultanée de programmes.
- ceci implique des mécanismes de contrôle:
  - ctrl-z suspend l'exécution d'un process
  - ctrl-c stoppe l'exécution d'un process
  - ctrl-d stoppe l'entrée de données ou sort du shell
  - jobs affiche l'état des jobs courants
  - fg job met le job défini en avant-plan
  - bg job met le job défini en arrière-plan
  - ps affiche les processus courants
  - kill pid tue le processus dont le numéro est pid
  - killall nom tue les processus dont le nom est spécifié
  - cmd & exécute la commande en arrière plan

ps

- Cette commande permet de faire des listings de processus en fonction de différents critères.
- Par défaut, uniquement les processus de l'utilisateur courant sont affichés
- Options utiles:
  - -u <username> affiche uniquement les processus de l'utilisateur spécifié
  - -ax affiche tous les processus du système
  - -f / -H affiche les processus sous forme de hiérarchie

top

- cette commande permet de faire du monitoring en temps réel des ressources utilisées par les différents processus du système.
- commandes utiles:
  - f permet de choisir les champs à afficher
  - o permet de choisir l'ordre des champs
  - F / O permet de choisir le critère de tri

history

- va afficher la liste de toutes les dernières commandes exécutée dans la session actuelle.
- elle peut être utilisée pour stocker dans un fichier la liste des commandes effectuées.
- Exemples:
  - history > ~/last_commands.txt
    => met dans 'last_commands.txt', la liste des dernières commandes.

Systèmes d'exploitation

Module SysExp

4ème cours - Scripts

variables d'environnement

- lorsqu'on se connecte à un système UNIX, un certain nombre de variables prédéfinies sont à disposition dans l'environnement d'exécution
- les variables sont identifiées par un nom et par le caractère spécial '$'
- pour afficher une variable, le plus simple est d'utiliser la commande 'echo':
  echo $variable

modifier une variable

- pour modifier une variable, utiliser cette syntaxe:
  <variable>=<valeur>
- Exemples:
  VARIABLE_GLOBALE=345
  variable_locale=678

quelques variables

- Les variables les plus courantes:
  - $SHELL le path du shell courant
  - $HOME le path du home directory
  - $USER le nom de l'utilisateur
  - $PWD le path du répertoire courant
  - $PS1 définit l'invite de commande (benoit@debian:~$)
  - $PATH liste des répertoires de recherche de commandes
- cf: dans le MAN du bash: 'Shell Variables'

variable $PATH

- lorsqu'on utilise un shell, le système utilise la variable $PATH pour savoir où il doit rechercher les commandes
- par défaut, le PATH contient les chemins /bin et /usr/bin. Ensuite se rajoute les paths spécifiques comme /usr/local/bin, /usr/bin
- dans la variable, les paths sont séparés par le caractère ':'
- à l'installation de programmes, il est parfois nécessaire de modifier cette variable.

variable $PS1

- l'invite de commande par défaut est: \h:\w\$
- il peut être modifié à loisir en utilisant ces options (pour toutes les options, chercher 'prompting' dans le man de bash):
  - \h le nom du host
  - \t l'heure au format hh:mm:ss
  - \u l'utilisateur courant
  - \w le répertoire courant
  - \d la date
  - \$ le type d'utilisateur ($ pour user, # pour root)

scripts UNIX

- un script est un fichier contenant des commandes destinées à un interpréteur de commandes
- lorsqu'un script est exécuté, l'interpréteur de commandes va lire chacune des lignes du fichier et les exécuter l'une après l'autre
- un script permet donc en quelques sorte d'automatiser une suite de commandes

création d'un script

- pour qu'un script puisse être exécuté par le système, il faut que sont bit d'exécution soit défini (chmod a+x)
- ceci permet de garantir la sécurité du système (car par défaut les fichiers n'ont pas ce bit défini)
- par convention, les scripts bash se terminent par l'extension ".sh"
- la première ligne du script doit indiquer quel shell utiliser pour l'exécution:

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

commentaires

- dans un script, toutes les lignes qui commencent par le caractère "#" sont considérées par le SHELL comme des commentaires.
- la seule exception est la première ligne, débutant par #! qui indique le shell à utiliser.

exécution d'un script

- pour exécuter un script il faut entrer son chemin. Si le script se trouve dans le répertoire courant, il faut ajouter "./" avant le nom du script
- Exemples:
  - ./test.sh
  - /home/benoit/test.sh
  - ~/test.sh

de l'utilité du PATH

- comme nous l'avons vu, le PATH permet d'indiquer au SHELL dans quels chemins il doit chercher pour trouver les commandes entrées au clavier.
- il peut donc être modifié pour prendre en compte un chemin local au dossier utilisateur. Par exemple:
  - $HOME/bin
- cela permet d'éviter de devoir toujours entrer le chemin des scripts.

paramètres de scripts

- lorsqu'un script est exécuté, un certain nombre de paramètres sont disponibles
- ils permettent de passer des valeurs au script lors de son exécution
- paramètres:
  - $0 contient la ligne commande
  - $# contient le nombre de paramètres
  - $* contient tous les paramètres
  - $n contient le paramètre n
  - $$ contient le numéro du processus
  - $? contient le code de retour de la dernière commande

.bash_profile / .bashrc

- ces 2 scripts sont contenus dans le home directory de chaque utilisateur.
- ils sont exécutés automatiquement par le système lorsque l'utilisateur se connecte.
  - .bash_profile est exécuté à chaque login
  - .bashrc est lancé ensuite par .bash_profile
- ils peuvent être utilisés pour personnaliser l'environnement d'exécution

utilisation des variables

- pour l'affectation, l'utilisation est identique à celle de la ligne de commande:

```
variable=valeur
```

- il est souvent utile de pouvoir isoler les noms de variables dans les lignes de scripts. Pour se faire, il faut utiliser les accolades:

```
echo ${variable}
```

- cela permet d'éviter les confusions:

```
points=18
echo $pointspts serait ambigu
echo ${points}pts est explicite
```


visibilité

- lorsqu'un script est exécuté, toutes les variables de l'environnement courant sont copiées et donc disponibles dans celui-ci.
- lorsque des variables sont crées dans un script, elles ne sont pas forcément transmises aux sous-processus éventuels.
- pour permettre cette transmission, il faut utiliser la commande 'export' qui a pour effet d'exporter la variable dans l'environnement

expansion de variables

- cette fonction très pratique du bash permet de faire des manipulations sur des variables de scripts:
  - ${var:-word} met 'word' dans 'var' si celle-ci est vide
  - ${var:?word} affiche 'word' sur sterr si la variable est vide

```
echo "${variable_inconnue:?Variable non définie}"
```

  - ${var:start:n} retourne les 'n' caractères (optionel) de 'var' en partant du caractère 'start' (à partir de 0).

```
var='Exemple de variable'
echo "${var:8:2}" => retourne 'de'
echo "${var:11}" => retourn 'variable'
```

  - ${#var} retourne le nombre de caractères de 'var'

```
echo "${#var}" => retourne 19
```

expansion de variables

- ${var#pattern} retourne la valeur de 'var' en y enlevant, depuis le début, la plus petite occurrence de 'pattern'.
```
var=/home/benoit/exercices/test1.txt
echo "${var#/*/}" => retourne "benoit/exercices/test1.txt"
```
- ${var##pattern} retourne la valeur de 'var' en y enlevant, depuis le début, la plus grande occurrence de 'pattern'.
```
echo "${var##/*/}" => retourne "test1.txt"
```
- ${var%pattern} retourne la valeur de 'var' en y enlevant, depuis la fin, la plus petite occurrence de 'pattern'.
```
echo "${var%/*}" => retourne "/home/benoit/exercices"
```
- ${var%%pattern} retourne la valeur de 'var' en y enlevant, depuis la fin, la plus grande occurrence de 'pattern'.
```
echo "${var%%/*}" => retourne ""
```


expansion de variables

- ${var/pattern/string} remplace dans 'var' la première occurrence de 'pattern' par 'string'.
```
var="un 2 un 3 un"
echo "${var/un/truc}" => retourne "truc 2 un 3 un"
```
- ${var//pattern/string} remplace dans 'var' toutes les occurrences de 'pattern' par 'string'.
```
echo "${var//un/truc}" => retourne "truc 2 truc 3 truc"
```
- ${!var} va retourner le contenu de la variable dont le nom se trouve dans var (indirection).
- cf: dans le MAN du bash: 'Parameter Expansion'

structure 'if'

- permet de faire des structures conditionnelles
- syntaxe:
```
if [ <expression> ]; then
  <commandes>
elif [ <expression> ]; then
  <commandes>
else
  <commandes>
fi
```
- exemple:
```
if [ -f /home/benoit/test.txt ]; then
  echo "Le fichier test.txt existe"
fi
```


expressions conditionnelles

- Il y a un grand nombre de conditions:
  - -a file: true si 'file' existe
  - -d file: true si 'file' est un répertoire
  - -r file: true si 'file' existe et est lisible
  - -w file: true si 'file' existe et est inscriptible
  - file1 -nt file2: true si 'file1' existe et est plus récent que 'file2'
  - -z var: true si la variable 'var' est vide
  - -n var: true si la variable 'var' a une longueur supérieure à 0
  - str1 == str2: true si str1 est égal à str2
  - -lt -le -gt -ge -ne: correspondent à < <= > >= !=
- cf: dans le MAN du bash: 'Conditional Expressions'

entrées / sorties

- comme nous l'avons vu, toute commande (et donc tout script) a une entrée standard, une sortie standard et une sortie d'erreurs.
- pour récupérer les données de l'entrée standard, il faut utiliser la commande 'read'.
- pour écrire sur la sortie standard, on peut utiliser la commande 'echo'.
- Exemple:

```
read INPUT_DATA
echo $INPUT_DATA
```


fin de script

- il est possible d'interrompre l'exécution d'un script à tout moment en utilisant la commande 'exit'.
- cette commande prend en paramètre une valeur numérique qui correspond au code de retour de la commande (exit status)
- Exemple:

```
if [ -z $1 ]; then
  echo "Paramètre 1 est vide!"
  exit 1
fi
```

Systèmes d'exploitation

Module SysExp

5ème cours - Scripts

fonctions - déclaration

- Il y a 2 syntaxes pour déclarer des fonctions:

```
function nom_de_fonction {
   <instructions>
}

nom_de_fonction() {
   <instructions>
}
```

- Ces 2 syntaxes n'ont pas de différence sémantique.
- Pour terminer une fonction, il existe la commande return <code>. Qui a la même sémantique que la commande exit utilisée pour sortir d'un script.

fonctions - appel

- L'appel d'une fonction se fait par son nom.
- Pour que la fonction soit reconnue, elle doit être déclarée avant son appel.
- Exemple:

```
function nom_de_fonction {
   echo -n "Je suis dans la fonction. Sortir? "
   read sortir
   if [ "$sortir" = "oui" ]; then
       return 1
   fi
   echo "Suite de la fonction"
}

echo "Début du script"
nom_de_fonction
echo "Fin du script"
```


fonctions - paramètres

- Il est possible de passer des paramètres aux fonctions, de la même manière que les paramètres sont passés aux scripts:

```
function ma_fonction {
   echo "1er paramètre: $1"
   echo "2ème paramètre: $2"
}

echo "Début du script"
ma_fonction "1 truc" chose
```

- Les paramètres du script sont cachés par ceux de la fonction. Ils ne sont plus atteignables par les variables $n à l'intérieur des fonctions.
- Pour pouvoir y accéder, il faut les passer en paramètres de la fonction ou les mettre dans d'autres variables avant l'appel de celle-ci.

substitution de commandes

- Il est souvent utile, dans les scripts, de pouvoir récupérer les données de l'exécution d'une commande.
- Pour cela on utilise la substitution de commandes:

```
$(<commande>)
```

- Par exemple, pour mettre dans une variable la liste des fichiers d'un répertoire:

```
variable=$(ls -l ~/cours)
```


expressions mathématiques

- il est possible d'évaluer des expressions mathématiques avec le Bash.
- pour ceci, il faut utiliser cette syntaxe spéciale:

```
$((<expression mathématique>))
```

- cela permet d'effectuer des calculs.
- Exemple:

```
var1=10
var2=$(($var1 * 10 - 50))
```


structure 'case'

- cette structure de contrôle permet de choisir une valeur parmi plusieurs.
- syntaxe:

```
case expression in
   pattern1 )
      <instructions> ;;
   pattern2 )
      <instructions> ;;
   * )
      <instructions> ;;
esac
```


structure 'case'

- Exemple:

```
echo -n "Votre choix: "
read choix
case $choix in
   "oui" )
      echo "vous avez accepté" ;;
   "non" )
      echo "Vous n'avez pas accepté" ;;
   * )
      echo "Vous n'avez rien choisi..." ;;
esac
```

structure ‘for’

- la structure for permet de faire une itération en se basant sur une liste d’éléments.
- syntaxe:

```
for variable in list
do
   <instructions>
done
```

- cette structure aura pour effet de mettre dans `variable` successivement toutes les valeurs contenues dans `list` et d’exécuter les `instructions` à chaque fois.

structure ‘for’

- Exemple:

```
for fichier in $(ls -a ~)
do
   echo “$fichier”
done
```

- ce script aura pour effet de lister le contenu du home directory de l’utilisateur courant.

structure ‘for’

- cette structure peut-être utilisée de la même manière que dans les autres langages, pour faire une itération.
- syntaxe:

```
for ((i=1;i<=10;i+=1))
do
   echo $i
done
```

- cette structure aura pour effet de mettre dans `i` successivement toutes les valeurs de 1 à 10.

structures ‘while’ et ‘until’

- les structures while et until permettent de répéter une suite d’instructions tant qu’une condition est respectée ou, inversément, non-respectée
  - syntaxe:

```
while condition
do
   <instructions>
done

until condition
do
   <instructions>
done
```

structure ‘while’

- Exemple:

```
i=10
while [ $i -gt 0 ]
do
   echo “i: $i”
   i=$(($i - 1))
done
```


structure ‘select’

- cette structure permet de générer un menu de choix parmi une liste d’options.
- syntaxe:

```
select variable in list
do
   <instructions>
done
```

- pour sortir de la structure, il faut utiliser la commande ‘break’.

structure ‘select’

- Exemple:

```
select choix in un deux trois sortir
do
   if [ $choix = “sortir” ]; then
      break
   else
      echo $choix
   fi
done
```

- va générer ce menu:

```
1) un
2) deux
3) trois
4) sortir
#?
```


commande ‘shift’

- cette commande permet de “shifter” les paramètres d’un script.
- elle a pour effet de mettre le paramètre n°2 dans le n°1, le n°3 dans le n°2, etc.
- elle peut-être utilisée pour gérer des paramètres complexes de manière générique.
- en effet, les paramètres d’un script ne sont pas sensés avoir un ordre défini.

commande ‘shift’
exemple

```
while [ ${#} -gt 0 ];
do
  case "${1}" in
    -h)  usage; exit 1;;
    -p1) shift; param1=$1;;
    -p2) shift; param2=$1;;
    -q)  quiet=true;;
    -v)  version; exit 1;;
  esac
  shift
done
```

Systèmes d'exploitation

Module SysExp

6ème cours - Config linux

Principe général

- Comme nous l'avions vu dans les premiers cours, les systèmes UNIX et, par extension, Linux basent la majorité de leurs configurations sur des fichiers textes qui sont contenus dans le répertoire /etc
- Ce principe permet d'avoir une certaine homogénéïté bienvenue dans l'accès aux fichiers de configuration.
- Un système UNIX actuel contient des centaines de fichiers de configuration dont nous allons faire un survol rapide (et non exhaustif!)

/etc/crontab & /etc/anacrontab

- ces fichiers permettent de configurer les système d'exécution automatique du système.
- le système cron permet la configuration d'exécutions chronologiques, en spécifiant les heures précises d'exécution
- le système anacron permet en plus de gérer les cas où la machine n'est pas allumée lorsqu'un script doit être exécuté. Si ce cas se produit, les scripts seront exécutés au prochain allumage.

configuration cron

- le format du crontab est trivial:
  min hour daymonth month dayweek user script
- exemple
  * * * * * debian /home/debian/bin/helloworld
  va exécuter toute les minutes le script 'helloworld' avec l'utilisateur debian.

  15 * * * * ... - va exécuter toutes les heures 15
  * * 10 * * ... - va exécuter toutes les 10 du mois
  0 0 * * 5 ... - va exécuter tous les vendredis à minuit
  0 7 31 12 * ... - va exécuter tous les 31 décembre à 7h
  */4 * * * * ... - va exécuter toutes les 4 minutes
  10,15,20 * * * * ... - va exécuter aux 10, 15, 20

fichiers cron utilisateurs

- Chaque utilisateur possède un fichier cron spécifique.
- Ce fichier a le même format que le fichier principal à part que la colonne 'user' est omise.
- Pour l'éditer, il suffit d'utiliser la commande:
  - crontab -e

répertoires /etc/cron.*

- En plus de la configuration manuelle des exécutions à l'aide de /etc/crontab, les répertoires cron.hourly, cron.daily, cron.weekly, cron.monthly sont utilisés pour des opérations périodiques.
- Les scripts qui se trouvent dans ces différents répertoires sont exécutés automatiquement par cron et anacron par l'intermédiaire de la commande runparts.
- Il est donc possible d'ajouter des scripts dans ces répertoires.

/etc/debian_version

- Contient les informations de version de la distribution Linux. Un fichier similaire se retrouve dans toutes les distributions.
- Exemple: SuSE-release, Redhat_version, redhat-release, mandrake-release, *-release, etc.
- La commande "uname -a", permet aussi d'avoir des informations intéressantes sur le système UNIX actuel (marche aussi sur OS X).

/etc/fstab

- Ce fichier contient la liste de montage des partitions du système.
- C'est dans ce fichier que sont définis les accès aux différentes partitions et périphériques du système
- La syntaxe est la suivante:

  <file system> <mount point> <type> <options> <dump> <pass>

- <file system> représente la source du périphérique, correspond à une entrée du répertoire /dev
- <mount point> représente le répertoire où doit être monté le périphérique
- <type> représente le format du système de fichier

exemple de table de montage

- Exemple de table de montage:

```
/dev/sda1   /                ext3   defaults,errors=remount-ro  0   1
/dev/sdb1   /home            ext3   defaults        0       2
/dev/sdc1   none             swap   sw              0       0
/dev/hdc    /media/cdrom0    udf,iso9660 user,noauto     0       0
/dev/fd0    /media/floppy0   auto   rw,user,noauto  0       0
```

- On retrouve ici les différents périphériques du système
- Le système utilisera cette table de montage au démarrage du système pour mettre en place les différents périphériques

montage manuel

- Il est possible de monter manuellement des systèmes de fichiers à l'aide de la commande "mount".
- Syntaxe:

  mount <file system> <mount point> -t <type>
- Le répertoire "mount point" doit exister avant d'exécuter la commande
- Pour démonter un système de fichiers, utiliser la commande umount <mount point>

état du montage

- la commande 'mount', lorsqu'elle est exécutée sans paramètres, affiche la liste des périphériques actuellement montés sur le système.
- la command 'df' quant à elle permet d'obtenir des statistiques sur l'utilisation des différents disques.
- L'option '-h' de la commande df permet d'afficher les statistiques en unités "humaines" (Kb, Mb, Gb, Tb, etc.)

/etc/group & /etc/passwd

- Ces fichiers contiennent les listes des groupes et des utilisateurs du système.
- Ils sont gérés automatiquement par les commandes useradd, usermod, userdel, groupadd, groupmod, groupdel (cf pages man)
- Dans la plupart des systèmes UNIX actuels, on trouve en plus le fichier /etc/shadow qui contient la liste des mots de passe cryptés des utilisateurs (fichier qui bénéficie d'une protection supérieure à /etc/passwd)

/etc/hostname

- Ce fichier contient le nom de domaine spécifique à la machine.
- C'est ce fichier que consulte la commande 'hostname'.
- Cette information est importante dans un contexte de réseau, le hostname permettant d'identifier les machines autrement que par leurs adresses IP.

/etc/hosts

- Ce fichier contient une liste d'associations de noms de domaines avec des adresses IP
- Il permet de court-circuiter la résolution des noms de domaine du système.
- C'est très pratique pour faire des tests de gestion de noms de domaines, il est effet possible d'associer (en local uniquement) n'importe quelle adresse IP avec n'importe quel nom de de domaine.
- Technique utilisée par certains virus dans les campagnes de Phishing

/etc/hosts

- Ce fichier existe aussi sur OS X et sur Windows (C:\WINDOWS\system32\drivers\etc\hosts)
- Format du fichier:
  <ip address> <domain>
- Exemple:
  192.168.1.1 www.ubs.com
  192.168.1.1 update.microsoft.com
- Dans son fonctionnement général, le système va tout d'abord consulter le fichier /etc/hosts et ensuite, si la correspondance de nom de domaine n'est pas trouvée, il va faire une requête au serveur de nom (DNS).

/etc/resolv.conf

- Ce fichier contient la liste des serveurs de noms que le système doit utiliser pour résoudre les noms de domaines.
- Lors d'une connexion en DHCP, ce fichier est généré automatiquement par le système.
- Format du fichier:
  domain <search domain>
  nameserver <dns_ip_1>
  nameserver <dns_ip_2>
  etc.

répertoires /etc/rc*.d

- Ces répertoires sont les répertoires de démarrage du système.
- Lors du démarrage, le système va consulter le répertoire correspondant à son niveau d'exécution (runlevel) et exécuter les scripts correspondants dans l'ordre défini dans le répertoire
- Ce système permet de définir plusieurs niveaux de démarrage du système
- Par défaut, un système Linux démarre avec le niveau 2

runlevels

- La commande 'runlevel' permet d'afficher le runlevel courant.
- Différents niveaux:
  - le niveau 0 correspond à l'extinction du système
  - le niveau 1 correspond au démarrage en mode single-user (réparation)
  - les niveaux 2-5 correspondent au démarrages standards du système (par défaut 2 ou 3 selon les distributions)
  - le niveau 6 correspond au redémarrage du système

gestion des runlevels

- il est possible de changer à tout moment de niveau d'exécution à l'aide de la commande init <runlevel>.
- Exemple: en tapant init 6, le système va redémarrer
- La configuration des runlevels se fait dans le fichier inittab (cf. man inittab)
- Pour ajouter un script dans un runlevel, chaque distribution a ces propres outils, sous Debian c'est update-rc.d, sous Suse c'est insserv, etc.

/etc/rc2.d

- Ce répertoire contient donc la liste des scripts du runlevel 2:

```
S10sysklogd  S19vmware-tools  S20makedev       S21nfs-common  S99rc.local
S11klogd     S20acpid         S20openbsd-inetd S89atd         S99rmnologin
S18portmap   S20exim4         S20ssh           S89cron        S99stop-bootlogd
```

- La gestion des numéros de scripts se fait automatiquement par les script de gestion update-rc.d ou insserv.
- Ces numéros différenciés permettent de mettre une priorité dans l'exécuation des scripts.

/etc/services

- Ce fichier bien pratique contient la liste de tous les ports TCP et UDP officiellement reconnus.
- Elle permet de connaître le port associé à un service réseau particulier.
- Elle est utilisée par plusieurs outils UNIX pour faire ces associations.
- Elle peut bien évidemment être consultée par l'utilisateur dans ce même but.

/etc/protocols

- Ce fichier contient la liste de tous les protocoles réseau officiellement reconnus.
- Elle permet de connaître le protocole associé à un nom et numéro particuliers.
- Elle est utilisée par plusieurs outils UNIX pour faire ces associations.
- Elle peut bien évidemment être consultée par l'utilisateur dans ce même but.

/var/log/*

- Ce répertoire contient par convention les différents logs des applications et du système.
- Le log système principal est contenu dans le fichier /var/log/messages
- Le log du noyau se trouve dans /var/log/kernel.log
- Pour voir l'évolution d'un fichier log en temps réel, on peut utiliser la commande tail -f <log_file).
- L'option '-f' permet de continuer la lecture du fichier indéfiniment.

/etc/logrotate.d

- Ce répertoire contient les scripts de gestion de la rotation des fichiers logs.
- Ce processus est important car les fichiers logs ont généralement tendance à croître énormément.
- Il est donc nécessaire de faire de l'ordre dans ces fichiers de manière périodique.
- Ce répertoire est utilisé par la commande logrotate, lancée par cron à intervales réguliers.

/etc/sudoers

- Ce fichier est lié à la commande "sudo" qui est très utile pour effectuer des opérations en tant que super-utilisateur (root) à partir d'un utilisateur standard.
- Les droits accordés aux utilisateurs standard, en relation avec la commande sudo, sont configurés dans ce fichier sudoers.
- Le fichier se modifie avec la commande "visudo" qui permet une édition sécurisée.

/etc/lilo.conf

- C'est le fichier de configuration du bootloader lilo.
- Ce fichier défini avec quel kernel, quelle image de démarrage et sur quel périphérique le système doit démarrer.
- Pour appliquer la configuration, utiliser la commande 'lilo' qui a pour effet de parser le fichier et de stocker dans le MBR les données correspondantes.

Opérations d'admin

- La plupart des commandes vues jusqu'à présent nécessitent l'exécution en tant que super-utilisateur.
- Lorsque un système UNIX est installé, l'utilisateur root est créé par défaut, en plus d'un premier utilisateur standard qui a des droits d'administration. Ces droits résidant justement dans l'utilisation possible de la commande sudo.
- Il est mieux d'utiliser cet utilisateur standard dans une utilisation courante et d'utiliser sudo lorsque c'est nécessaire.

Gestion des utilisateurs

- Comme vu précédemment, la commande "useradd" permet d'ajouter un utilisateur.
- Pour changer d'utilisateur, il faut utiliser la commande "su".
- Utilisée sans paramètre, cette commande permet de se logger comme super-utilisateur (root)
- Sinon elle s'utilise ainsi: su <username>

Gestion réseau

- Pour afficher l'état des périphériques réseau du système, il faut utiliser la commande "ifconfig".
- Cette commande affiche la liste des interfaces ainsi que leurs configurations actuelles.
- Normalement, le réseau se lance automatiquement au démarrage, mais il est possible de le contrôler manuellement.
- Pour démarrer ou relancer le réseau, il faut utiliser la commande:

  /etc/init.d/networking start ou restart

Configuration réseau

- La configuration réseau elle-même se trouve dans le répertoire /etc/network.
- Le fichier /etc/network/interfaces contient la liste des interfaces réseau du système.
- Les fichiers /etc/network/if-*.d contiennent des scripts exécutés en fonction des événements réseau (connexion, déconnexion, etc.)

outils réseau

- netstat: permet d'avoir une aperçu de toutes les connexions réseau actives sur le système.
- traceroute: permet d'afficher le "chemin" d'accès à un serveur en détaillant tous les points du réseau par lesquels passent les paquets de données.
- dhclient: permet de lancer le client dhcp et de lancer ainsi une requête d'obtention d'adresse IP au serveur DHCP du réseau courant.

Accès au système

- Une fois que le système est installé, il y a plusieurs moyens pour y accéder.
  - Le plus intuitif est l'interface graphique
  - telnet: accès au shell en connexion non-sécurisé
  - SSH: accès au shell en connexion sécurisée mais nécessite l'installation d'un serveur ssh (sshd), installé par défaut sur la plupart des systèmes.
  - Dans un système UNIX graphique, il est possible à tous moments de passer au mode console en utilisant la combinaison de touches: ctrl-alt-fn
    Pour revenir à l'interface, taper ctr-alt-f7

Installation de logiciels

- Il y a deux grandes manière d'installer des logiciels sous UNIX:
  - par des binaries précompilées
  - par les fichiers sources

Binaries précompilées

- Chaque distribution a son propre format par exemple, *.rpm pour les packages au format "redhat" ou *.deb pour les packages debian.
- Ces fichiers contiennent les fichiers compilés ainsi que des données de contrôle permettant de garantir la bonne installation du logiciel.
- L'avantage de ce système est que l'utilisateur n'a pas besoin de s'occuper de compatibilité ou autre vu qu'il garanti l'utilisation avec les distributions indiquées.

Fichiers sources

- Les fichiers sources (disponibles sous forme d'archive compilée *.tar.gz) sont récupérés et compilés directement sur la machine.
- Cela à l'avantage de garantir une parfaite adéquation avec la configuration effective du système et de permettre éventuellement une paramétrisation précise.

Compil. - Téléchargement

- Télécharger les sources:
  - curl -O <url>
- Désarchiver le fichier:
  - tar -xzf sources.tar.gz
  - cd sources
- Consultation des fichiers informatifs:
  - less README ou less INSTALL

Compil. - Configuration

- Cette étape à pour but d'analyser l'environnement du système et de configurer la compilation en fonction de cet environnement.
- Si des pré-requis (librairies, compilateurs, etc) ne sont pas remplis, le script en notifiera l'utilisateur.
- Elle se fait par l'intermédiaire du script "configure".
  - ./configure
- Une fois la configuration effectuée, un fichier 'Makefile' est généré qui contiendra les différentes instructions de compilation

Compilation - Make

- Cette étape à pour but de compiler effectivement les différents composants du logiciel.
- Elle se basera sur le fichier 'Makefile' généré à l'étape précédente
- Les fichiers seront compilés dans le répertoire courant:
  - make

Compil. - Make install

- La dernière étape consiste à installer effectivement les fichiers compilés aux bons endroits du système (répertoires bin, lib, share, etc).
- Comme cette étape agit sur le système en lui-même, elle doit être effectuée en tant qu'utilisateur root:
  - sudo make install

Compil. - Aide

- Si un problème survient à n'importe quelle étape d'une compilation, vous pouvez généralement partir du principe que vous n'êtes pas le premier à le rencontrer.
- La consultation des forums de discussion apportent généralement les solutions correspondantes.
- Il s'agit juste de poser les bonnes questions, en se basant sur les notifications d'erreurs retournées par ./configure ou make