Présentation

- Pascal PETIT
- 01 60 87 39 03 (tel prof., aléatoire)
- Email: pascal.petit@info.univ-evry.fr
- http://www.ibisc.fr/~petit
- Objectif: être à l'aise sur les machines du dept
- tant sous unix que sous windows

Plan du cours:

- présentation du parc du département
- quelques rappels réseau
- mise en route: unix/windows
- unix: outils et tâches usuelles (X11, ssh, impressions, ...)
- windows: gestion des droits d'accès
- unix: gestion des droits d'accès
- vmware: notions et utilisation

Plan du cours (suite)

- unix:
  - généralités, historique
  - système de fichiers
  - droits d'accès
  - interpréteur de commande en mode interactif
  - éditeurs de texte classiques
  - interpréteur de commandes, commandes usuelles
  - gestion des processus, redirections, filtres
  - programmation shell
  - impressions
  - sauvegardes
  - réseau

Parc du département:

- 6 salles d'informatique pédagogique équipées de matériel performant (Core2Duo, 4Go RAM, écran LCD 19 pouces, ...)
- salles réservées aux étudiants de L3, M1 et M2 du département informatique.
- double boot Unix (linux) – windows
- de nombreux logiciels sont disponibles
- certaines salles sont équipées d'imprimantes (n'imprimez pas vos cours dessus !)
- documentation sur
  - http://dept.lami.univ-evry.fr/

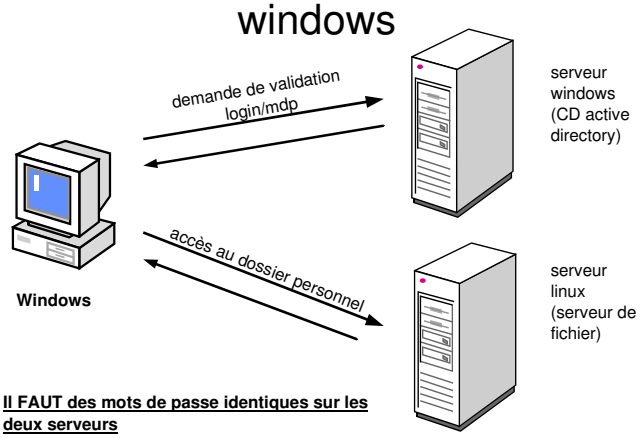
Compte personnel

- compte personnel
  - ouverture de session sous unix et sous windows
  - dossier personnel : 150Mo max
  - navigation WeB via proxy (login/mot de passe géré par la Division informatique)
  - compte mèl géré par la DI
  - accès à vos fichiers depuis chez vous via sftp/scp/rsync/unison

MSDNA

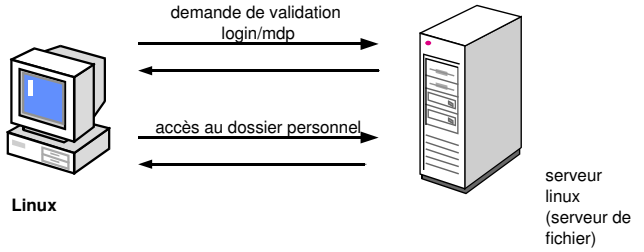
- accès gratuit à de nombreux logiciels microsoft (mais pas ms-office)
- licence personnelle
- à télécharger chez microsoft
- modalité d'inscription
  - le délégué de chaque filière établit une liste comprenant pour chaque étudiant :
    - ses nom et prénom
    - son No d'étudiant (ne peuvent y avoir droit que les étudiants inscrits)
    - son adresse mèl

ouverture d'une session sous windows

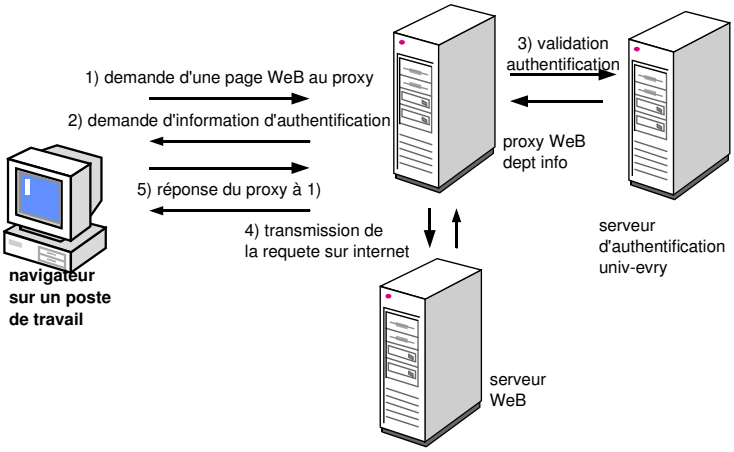


Il FAUT des mots de passe identiques sur les deux serveurs

ouverture d'une session sous Linux



Accès au WeB



Accès au Web : paramètrage

- utilisation du WeB sous linux et sous windows
- dans votre navigateur, indiquer l'adresse du mandataire cache (proxy) dans les préférences du navigateur :
  - proxy: proxy-www.miage.info.univ-evry.fr sur le port 3128

services proposés par le service informatique central

- tout étudiant inscrit a un compte utilisateur dans les salles non spécialisées gérées par le SI central
  - login: No d'étudiant
  - mot de passe à définir avant la première utilisation sur http://pedago.ens.univ-evry.fr/
- services
  - utilisation des salles du 2e étage du bât. 1er cycle
  - adresse mèl
  - accès à l'application dokeos
  - cf http://pedago.ens.univ-evry.fr pour plus

SI central : courriel

- tout étudiant a un compte mèl prenom.nom@ens.univ-evry.fr
  - consultable via webmail depuis l'intérieur ou l'extérieur de l'université
  - possibilité de rediriger les courriers vers votre véritable adresse : **FAITES LE**
  - vous devez lire les courriers envoyé à l'adresse prenom.nom@ens.univ-evry.fr. L'administration l'utilisera pour communiquer avec vous.

accès fichiers depuis l'extérieur de l'université ou en WiFi :

- protocoles supportés: sftp, rsync, unison, ...
- Paramètres pour sftp:
  - hôte: ns.info.univ-evry.fr
  - port: 60022
  - compte: votreLogin au dept informatique
- plus d'informations:
  - http://dept.lami.univ-evry.fr/acces-sftp.html

accès fichiers: outils permettant l'accès via sftp ou autre protocole supporté :

- Filezila (graphique, windows)
- winscp (graphique, windows)
- gftp (graphique, unix)
- sftp (outil en ligne de commande): sftp -oPort=60022 votreLogin@ns.info.univ-evry.fr
- fugu (graphique, maxOSX) : site d'origine , didacticiel (non testé)
- rsync (ligne de commande, tous systèmes) : particulièrement optimisé pour les synchronisation d'arborescences
- grsync (graphique, linux) : non testé, protocole

utiliser son portable au département informatique

- accès en salle C123 (libre service pour portable)
- Wifi: l'ensemble des salles du département est couvert par des bornes WiFi
- des prises de courant spécifiques pour les portables ont été installées dans toutes les salles
- NE PAS DEBRANCHER LES POSTES INFORMATIQUES DES SALLES sous peine de sanction

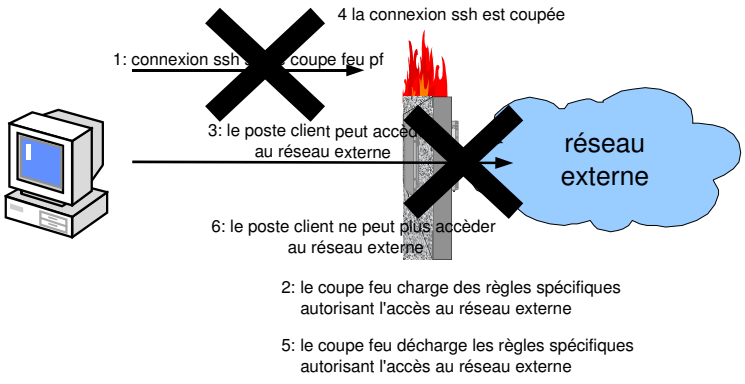
salle libre service pour portables

- opérationnel courant 09/2007
- pour plus d'information : http://dept.lami.univ-evry.fr/libre-service-portable.html
- salle équipée :
  - prises de courant
  - prises réseau ethernet
  - l'accès au réseau nécessite une authentification

salle libre service pour portables : mode opératoire

- démarrer le portable
- réseau configuré en récupération automatique d'adresse IP et de dns (DHCP)
- se connecter via ssh (ssh sous linux, putty sous windows) à la machine nommée « wifi » qui est le routeur de la salle
- tant que la connexion ssh est active, le portable a accès au réseau (ssh, WeB, récup. mail, ...).

•NAC: authPF



accès WiFi

- l'accès utilise les bornes déployées par le service informatique
- Mode opératoire :
  - connexion sur le ssid « étudiants » (pas de chiffrement)
  - aller sur un site WeB quelconque avec un navigateur supportant javascript
  - une redirection mène vers la page d'authentification
  - fournir identifiant (No d'étudiant) et mot de passe du service informatique
  - tant que la fenêtre reste ouverte, l'accès est ouvert

accès WiFi : services disponibles

- accès au WeB (http, https)
- récupération de courrier via pop3 et pop3S
- envoi de courrier via smtp (utiliser smtp.univ-evry.fr comme serveur smtp sortant)
- accès ssh
- ràf: tester ftp et https

Mise en route

- distribution des chartes informatiques
- distribution des mots de passe
  - login/mot de passe commun unix et windows
  - ne pas le changer
- création du mot de passe à la DI:
  - http://pedago.ens.univ-evry.fr/
- utilisation du WeB sous linux et sous windows
  - proxy: proxy-www.miage.info.univ-evry.fr sur le port 3128

Systeme d'exploitation

- un OS (operating System) assure les services :
  - interface entre utilisateur et machine
    - offre des services logicicels (appels système, protection)
    - offre des services d'adminitration (gestion des ressources)
    - permet un accès uniforme à une classe de machines
  - la gestion des matériels composante la machine: périphériques, réseau
  - comparaison DOS/Windows sur les aspects gestion des pilotes

Critères de choix d'un OS

- critères économiques
  - plus répandu = moins chers (bof :-)), recrutement facile, peu de frais de formation
  - perennité (quid de l'existence du vendeur dans 2 ans, est-on lié à un vendeur pour les corrections/évolutions (logiciel libre/propriétaire))
  - logithèque
- critères technologiques
  - charge attendue, domaine d'utilisation, simple ou multi utilisateur, fiabilité, sécurité, ...
- Le choix existe de plus en plus (ms-windows vs MacOS vs Unix vs ...)

Quelques OS existants

- domaine de recherche actif
- système centralisés
  - monoutilisateurs
    - MacOS, windows, Dos, ...
  - Multi-utilisateurs:
    - **Unix** (Linux, *BSD, Solaris, AIX, ...) , qnx, mach, NT, Mac OS X, ...
- systèmes distribués
  - amoeba, plan9, chorus, OSF-1, Sprite...
- ainsi que de nombreux systèmes d'exploitation propriétaires de niche:

Les caractéristiques d'Unix

- inspiré de MULTICS (1969 - GE 645)
- système de gestion de fichier (SGF) hiérarchisé
- compatibilité des entrées/sorties
- création dynamique de processus
- communication inter processus (pipe, socket)
- langage de commande extensible
- système portable (noyau, fichiers, applications)
- système multi-tâches, multi-utilisateurs
- simple, nombreux outils de base disponibles
- rustique mais robuste (sans sécurité)

•Historique d'Unix

- **UN**iplex **I**nformation and **C**omputing **S**ystem

1969 lab. AT&T, lab. Bell, sur PDP-7 en assembleur
(à partir de MULTICS du MIT, par K. Thomson, D. Ritchie et B.W. Kernighan)

1974 AT&T le diffuse *gratuitement* auprès des Univ.
(UNIX V5 - à 90% en *c*)

1975 Unix V6 (interne à Bell + Universités)

1978 Unix V7

1984 AT&T à le droit de commercialiser Unix

- de nos jours, de nombreuses versions ou clones (solaris, Linux, *BSD, AIX, ...)

Architecture générale (vue de l'utilisateur)

- Éditeur de programmes (sed, ed, vi, emacs)
- Outils de développement (cc, as, ld)
- Debuggers (adb, sdb, ddd)
- Gérants de bibliothèques de prog. (M4, make, sccs, rcs)
- Construction de compilateurs (lex, yacc)
- Outils de documentation (nroff, troff, eqn, tbl)
- Outils de communication (mail, uucp, telnet, rlogin)

•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
  - centre d'aide ou ~ de votre gestionnaire de bureau
- 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:

	No section	
thème	SysV	BSD li-nux
commandes utilisateur	1	1
commande systeme	1m	8
référence du programmeur: noyau	2	2
référence du programmeur: biblio. std	3	3
format des fichiers de config.	4	5
informations diverses	5	7
jeux	6	6

•Le manuel: consultation

- via la commande man
  - « man entrée »
  - paramètres de la commande man:
    - « man No_section entrée» pour les entrées présente dans plusieurs section (ex.: printf)
    - « man -f entrée » ou « whatis -r entrée »: affiche la liste des pages de nom « entrée » dans toutes les sections
    - « man -k motClef » ou « apropos motClef »: affiche la liste des pages contenant le mot motclef
    - si whatis et apropos ne fonctionne pas, c'est que leur base de donnée n'est pas à jour. cf « man whatis ».
- via les outils graphiques: xman ,centre d'aide

•Démonstration: manuel

- man passwd
- man -k passwd
- man -f passwd
- idem avec printf

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

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

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

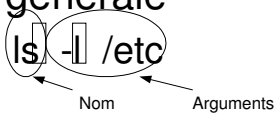
Historique (3)

- 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

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

impressions

- CUPS gère les impressions sur le parc du département informatique
- les outils usuels de Linux (lpr & Co) sont compatibles avec CUPS
- Exemples:
  - a2ps -2 -Pc121 monFichierTexteaMoi.txt : imprime le fichier sur l'imprimante c121 en 2 pages par pages
  - lpstat -a : donne la liste des imprimantes
  - lpq -Pc121: liste des travaux en attente d'envoi à l'imprimante
  - lprm -Pc121 idTravail : annule le travail IdTravail (-:

impressions

- xpp: équivalent à la commande lpr, fait apparaître une tableau de commande graphique permettant de sélectionner graphiquement l'imprimante et ses options
- xpp fait appel à cups
- utilisation classique :
  - a2ps -2 -o- monFichierTexteaMoi.txt | xpp : imprime le fichier en 2 pages par pages via xpp
  - avec les outils qui permettent de préciser le programme utilisé pour l'impression (xpdf par exemple)

sauvegardes

- archivage : tar
- compression:
  - compress (historique),
  - gzip: le standard (géré aussi par les outils windows)
  - bzip2: plus récent, compresse en général mieux que gzip
- combiner les deux :
  - tar cvf - fichiersAArchiver | gzip -9 > archive.tar.gz
  - tar czvf archive.tar.gz fichiersAArchiver (compression avec gzip)
  - tar cjvf archive.tar.gz fichiersAArchiver (compression avec bzip2)

Sauvegarde (suite)

- zip/unzip: outil en ligne de commande pour générer des fichiers zip
- ark: outil graphique pour générer des archives compressés. supporte de nombreux formats
- explorateur de fichier de KDE :
  - support natif de nombreux formats
  - pour créer des archives, s'y déplacer, les désarchiver
- le support (décompression) du format rar suppose l'installation de l'outil unrar (il l'est)

réseau

- présentation faite directement au tableau
- On a rappelé
  - la notion de couches
  - les couches 2, 3 et 4
  - les notions d'adresses ip et de ports
  - le routage IP

X window

- X window ou X11: gestion du clavier/souris/écran (désigné par la variable DISPLAY)
- peut en gérer plusieurs sur un même poste
- ne gère pas la décoration => nécessité d'un gestionnaire de fenêtre
- capable d'afficher des applications graphiques distantes
- gestion minimale de la sécurité
- client/serveur: le serveur (de ressources graphiques) est la machine où s'affiche

X window & ssh -X

- ssh -X fait transiter X 11 via le tunnel sécurisé de ssh
- méthode sure et simple (ssh fait tout). ssh positionne :
  - le MIT-magic-cookie
  - la variable DISPLAY
- l'option -X doit être autorisée :
  - sur le serveur distant (configuration du serveur sshd éalisée par l'administrateur du poste: activée au dept informatique)

ssh

- secure shell
  - un protocole : le protocole ssh (version courante: version 2.0)
  - des produits ou commandes implantant ce protocole
- pour répondre aux problème de sécurité des outils traditionnels (telnet, rsh, rcp, ...)
  - clients et serveur s'authentifient (pas de mim ou d'usurpation)
  - login, mot de passe et données passent dans un tunnel chiffré (imperméable à l'utilisation d'analyseur de trames, de sniffers)
  - ssh permet de relier deux machines sûres à travers

ssh (suite)

- une suite d'outils s'appuyant sur un procole securisé: ssh, scp, sftp (v2)
- des outils de gestion de clefs: ssh-keygen, ssh-add, ssh-agent
- des fonctionnalités
  - redirection de ports
    - rediriger un port de la machine locale vers un port d'une machine distante : permet l'accès à des applications/ressources de machines distantes via le tunnel ssh
    - idem avec un port de la machine à laquelle on est connecté vers un port local: permet l'accès à des ressources locales depuis des machines distantes via le tunnel ssh

clef privées/clefs publiques

- rappel sur les notions
  - chiffrement symétrique
  - de clefs privée/publiques
  - de l'impact de tout ça
- fait en live au tableau

algorithme de chiffrement

- chiffrement symétrique/asymétrique
  - symétrique:
    - les algo classiques sont rapides
    - la même clef sert au chiffrement et au déchiffrement
    - souvent utilisé via une clef de session
      - clef de session: transmise via algo asymétrique (on parle d'enveloppe digitale)
      - session: chiffrée par un algo symétrique et la clef transmise
  - asymétrique:
    - les algo classiques sont lents
    - couple de clef publique/clef privée
      - clef publique: peut être connue de tous
      - clef privée: tenue cachées
      - ce qui est chiffré avec l'une ne peut être déchiffré qu'avec l'autre

algorithmes classiques

- symétriques:
  - DES (1976): standard américain (1977), clef de 56 bits sur des blocs de 64 bits. dépassé de nos jours.
  - triple DES (1978): variante via une triple application de DES permettant d'avoir des clefs entre 128 et 192 bits sur des blocs de 64 bits.
  - RC2, RC4, RC5 (1994) et RC6:
  - IDEA (1992): clef 128 bits sur des blocs de 64 bits
  - blowfish: clef 32 à 448 bits sur des blocs de 64 bits. Algo très analysé,considéré comme solide. utilisation libre.
  - AES (1998): clefs 128, 192 ou 256 bits sur blocs de 128 bits. standard américain. utilisation libre.

algorithmes classiques

- asymétriques:
  - RSA s'appuyant sur la factorisation de nombres premiers
  - Diffie-Hellman et El Gamal s'appuyant sur le calcul des logarithmiques discrets
  - des algorithmes nouveaux s'appuyant sur les courbes elliptiques

cryptographie et ssh

- clefs d'hôtes: couple clefs privée/publique, sert à garantir que l'on dialogue avec la bonne machine
- clef de session: clef calculée par ssh pour chiffrer la session. algorithme symétrique (une seule clef)
- clef d'authentification utilisateur: méthode alternative d'authentification. Remplace la fourniture du mot de passe de connexion

authentifier les hôtes

- principe général :
  - couple clefs publique/clef privée
    - la clef publique est diffusée par un canal sur aux machines B1, B2, ...
    - en mode non paranoïaque, lors de la première connexion, la machine B1 récupère la clef publique de A
  - la clef privée de A est gardée en lieu sur A
  - à l'aide de la clef publique de A, les machines B1, B2, ... peuvent authentifier la machine A
    - si A fait une connexion ssh vers B1
    - si B1 fait une connexion ssh vers A
  - lors d'une connexion ssh de A vers B :
    - A doit avoir la clef publique de B pour l'authentifier
    - B doit avoir la clef publique de A pour l'authentifier

l'échange de clef: première phase de la connexion

Clef de session

- deuxième phase du procole
- la première phase a permis l'échange d'informations chiffrées entre les machines et la négociation des protocoles utilisés (chiffrement, compression, ...)
- Clef de session : une clef pour algorithme symétrique, chiffre l'ensemble des données de la session
- Clef de session: change à chaque session
- Les algorithmes de chiffrement symétriques sont plus rapides

Authentification de l'utilisateur

- 3 méthodes: le serveur indique au client les méthodes qu'il accepte. Le client essaie dans l'ordre fourni par le serveur celles qu'il supporte
  - par login/mot de passe (transitent dans un tunnel chiffré)
  - par clefs privée/publique
    - possibilité de protéger la clef privée par une passphrase
    - possibilité de limiter l'accès avec une clef depuis certains hôtes (cf « man sshd »)
  - par reconnaissance d'hôtes
    - on définit (fichier .shosts) des machines dont les utilisateurs ayant un compte local pourront ouvrir une session localement sans mot de passe

Authentification des utilisateurs par clefs privée/publique

- permet des connexions sans mot de passe
  - utilise un couple clefs privée/publique
  - clef privée sur le poste client
  - clef publique de l'utilisateur sur le serveur dans ~/.ssh/authorized_keys
  - la clef privée permet à l'utilisateur de s'authentifier
  - possibilité de restreindre les postes depuis lesquels la clef est utilisable (cf man sshd)
  - la clef privée est protégée par une « passphrase »
  - se génère avec « ssh-keygen -t dsa » (par exemple)
  - il existe des outils de gestion de clefs (pour éviter

Redirection de ports

- dans une version ultérieure de ce document

Outils windows pour faire du ssh:

- putty/pscp: équivalent graphique de ssh
- filezila: pour réaliser des transferts par sftp (déconseillé en cas d'utilisation de clefs privée/ publiques)
- winscp (conseillé): outil graphique permettant de faire du transfert de fichier via sftp/ssh
- doc sur l'accès fichier au département: http://dept.lami.univ-evry.fr/

TP

- connexion ssh sur une machine distante de la salle et lancement d'une application graphique (xeyes)
- idem mais en enchaînant deux connexion ssh de suite A->B->C
- on souhaite pouvoir se connecter sans mot de passe d'une machine du dept à une autre
  - citer les étapes pour y parvenir en indiquant sur quelle machine est à réaliser quelle action
  - mettre en application (on mettra une « passphrase à la clef)

TP

- à quoi sert la commande ssh-add
- utilisez là pour ne fournir la « passphrase » qu'une fois et vous connecter ensuite sans la fournir

rsync

- outil de transfert de fichier optimisé:
  - rsync ne transfère que les fichiers modifiés ou nouveaux
  - il ne transfère que la partie modifiée des fichiers modifiés
- s'appuie sur ssh
- compatible avec l'accès extérieur à vos fichiers du département informatique
- plus d'info:
  - http://samba.anu.edu.au/rsync/tech_report/ : rapport technique sur l'algorithme utilisé par rsync

Rsync: options utiles

- -r : copie récursive
- -a: archive mode, préserve autant que possible les propriétés des fichiers copiés (sauf les liens physiques, option -H). équivaut à -rlptgoD (cf man)
- -v: affichage verbeux (notamment la liste des fichiers transférés)
- -vv, -vvv: encore plus verbeux
- -z: compresse les fichiers lors du transfert
- --progress: affiche une barre de progression

Rsync: exemples

- rsync -av --progress petit@asr270-23 :/home/petit/Test Old/ : transfère le dossier /home/petit/Test situé sur la machine asr270-23 dans le dossier local Old. La connexion ssh se fera en tant qu'utilisateur petit. Le dossier Old contiendra un sous-dossier Test
- rsync -av --progress petit@asr270-23 :/home/petit/Test/ Old/ : idem mais le contenu de Test est copié directement dans Old. Aucun sous-dossier Old/Test n'est créé

Rsync: TP

- on utilisera les options --stats, --progress et -v pour avoir des informations sur le comportement de rsync
- recopier l'arborescence /home/petit/Bazar/test-rsync situé sur la machine asr270-23 dans le dossier /tmp de votre machine locale
- modifier le fichier local gros.txt et relancer la copie
  - quels fichiers ont été transférés ?
  - vérifier que gros.txt est correct (les modifications ont disparu)

Séance No 2

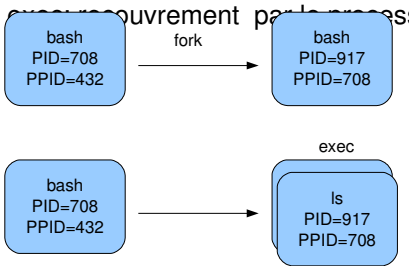
- processus sous unix
- retour sur ssh-agent
- réseau: routage (algo, DGW, ...)
- application à la configuration IP d'un poste
- introduction à vmware
- pratique de vmware
  - bases
  - réseaux virtuels
- processus sous unix. application: ssh-agent

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

hiérarchie de processus, recouvrement

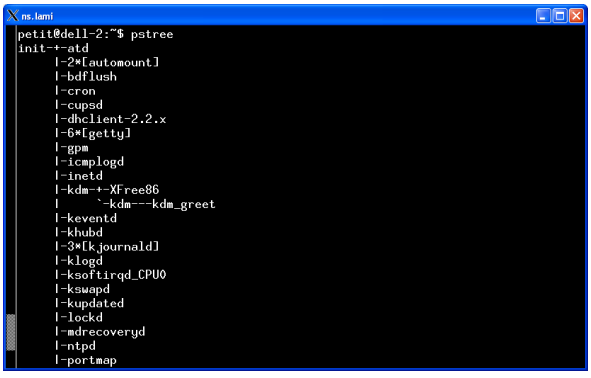
- un processus (processus fils) est toujours créé par un autre processus (processus père):
  - fork: création d'une copie du processus père
  - exec: recouvrement par le processus fils



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
  - file
  - echo

caractéristiques des processus

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

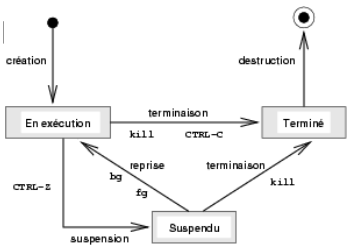
commande ps

- 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)

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
- 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

trap

- dans une version future de ce document

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

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

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

<<

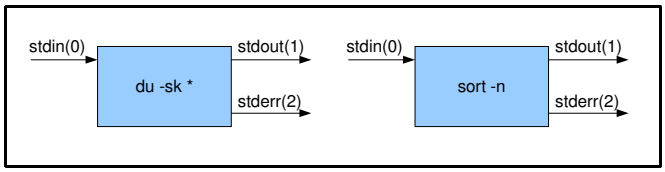
- dans une version ultérieure de ce document

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

commande test

- réalise des tests simple, le code de retour indique que le test est positif ou négatif
- test -d /var/tmp : teste si /var/tmp est un dossier
- test -x /bin/ls: teste si /bin/ls est un exécutable
- test 1 = 2: teste l'égalité de deux chaînes
- forme alternative :
  - test -d /var/tmp

structure de contrôle if

- syntaxe:

```
if commande1
then
    commande2
[elif commande3
then commande4]
...
[else
    commande5]
fi
```

- si commande1 retourne 0, on exécute commande2 sinon, si commande3 retourne 0, on exécute commande4 ... sinon

sort

- selon SUSv3, sort a trois fonctions sur son entrée standard ou des fichiers textes constituées de lignes contenant un ou plusieurs champs :
  - trier les données (par défaut)
  - fusionner des fichiers triées en une sortie globale triée (option -m)
  - vérifier que les données sont triées (option -c)

Sort: options courantes:

- -t car_sep: permet de préciser le caractère qui sépare les champs du fichier
- -k : précise les champs sur lesquels portent le tri
- -o: indique un fichier de sortie (par défaut: sortie standard)
- -d: supprime les doublons
- -c : vérifie si un fichier est trié. Le résultat est indiqué uniquement par le code de retour: 0 si trié, 1 sinon.
- -m : fusionne des fichiers supposés déjà triés

uniq

- supprime les doublons d'une liste triée
- exemple: cat /tmp/test.txt |sort|uniq
- voir manuel pour les autres options
-

tail/head

- queue/tête d'un fichier

wc

- compte le nombre de lignes, de mots et de caractères
- wc -l : nombre de lignes
- wc -w : nombre de mots
- wc -c : nombres de caractères
- ls | wc -l : donne le nombre de fichier du dossier courant

grep, egrep & Co

- grep chaine: sélectionne les lignes qui contiennent la chaine
- grep petit /etc/passwd: sélectionne les lignes de /etc/passwd contenant la chaîne petit
- caractères spéciaux de la commande egrep:
  - ^: début de ligne
  - $: fin de ligne
- grep '^petit:' /etc/passwd: sélectionne les lignes commençant par petit:

cut

- sélectionner certaines colonnes

tr

more/less

commande find

- find permet de chercher récursivement les fichiers vérifiant une ou plusieurs conditions
- outre les expression simples, l'expression que doit vérifier un fichier peut être de la forme (par priorité décroissante) :
  - (expression)
  - ! expression
  - expression1 -a expression2 : ET logique
  - expression1 expression2: ET logique
  - expression1 -o expression2: OU logique

commande find

- expressions élémentaires à argument numérique:
  - +n: toutes les valeurs supérieures ou égales à n
  - -n: toutes les valeurs inférieures ou égales à n
  - n: n exactement
  - par la suite, partout où on verra un argument numérique n, on pourra utiliser +n, n ou -n
  - exemples:
    - -size 1024k: les fichiers de taille égale à 1024 Ko
    - -size -1024k: les fichiers de taille inférieure égale à 1024 Ko
    - -size +1024k: les fichiers de taille supérieur ou égale

commande find: quelques expressions élémentaires

- quelques expressions élémentaires:
  - -name motifProtégé: les fichiers vérifiant le motif
  - -size n: les fichiers de taille n
  - -mtime n, -ctime n, -atime n
  - -perm p avec p ayant la forme numérique ou symbolique des arguments de chmod
  - -type c avec c=b,c,d (dossier),l (lien symbolique),p,f (fichier ordinaire),s
  - -user u
  - -group g
  - -link n : nombre de liens physique sur le fichier
  - -print: provoque l'affichage des noms des fichiers vérifiant l'expression (par défaut sur le Gnu find)

commande find: -exec

- -exec commande;
  - pour chaque fichier trouvé, la commande est exécutée.
  - si {}apparaît parmi les arguments de la commande, il est remplacé par le nom du fichier trouvé
- -exec commande arguments {} +
  - syntaxe POSIX/SUSv3 (standard mais pas disponible sur toutes les plateformes)
  - les noms des fichiers trouvés sont accumulés dans une liste l
  - la commande est exécutée une seule fois, à la fin de la recherche et {}est remplacé par la liste

commande find : exemples

- fichiers ordinaire nommés core de plus de 1024Ko
  - find -size +1024k -type f -name core -print
- fichier ordinaires de l'utilisateur petit ou fichiers ordinaires de taille supérieure à 1024 Ko et de nom core
  - find -type f \(-user petit -o \(-size +1024k -name core \) \) -print
- fichier dont le nom commence par C
  - find -name c* -print

commande find: -exec et les espaces (&Co)

- rm est une commande qui ne lit pas sur son entrée standard. 3 méthodes pour effacer un ensemble de fichiers sélectionnés par find :
  - on lance un rm par fichier (lourd)
    - find -name *.bak -exec rm -f {} \;
  - syntaxe POSIX: un seul rm global est lancé :
    - find -name *.bak -exec rm -f {} \+
  - une solution avec les options spécifiques de Gnu find:
    - find -name *.bak -print0 |xargs -0 rm -f
  - ne marche pas avec les noms de fichiers contenant des caractères à problème (espace, saut de ligne , ' etc):

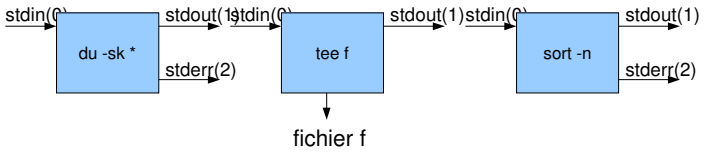
commande xargs

- xargs [options] commande
- xargs rassemble ce qu'elle reçoit sur son entrée standard dans une liste l et exécute « commande l »
- xargs s'utilise avec des commandes qui n'acceptent pas de données sur leur entrée standard
- Exemple:
  - grep -l perso * | xargs chmod 700

xargs : options utiles

- -p: prompt mode. une confirmation est demandée à l'utilisateur pour chaque invocation de la commande
- si le nombre ou la taille des arguments transmis via l'entrée standard est important, il est possible d'indiquer à xargs d'exécuter plusieurs fois la commande avec une liste limitée:
  - -n nombre: la commande est invoquée plusieurs fois et chaque invocation a au plus nombre arguments
  - -s taille: la commande est invoquée plusieurs

commande tee



- la commande tee envoie simultanément son entrée standard vers un fichier et vers sa sortie standard.
- options:
  - -a: ajoute au fichier
  - -i : ignore le signal

commandes qui ne lisent pas leur entrée standard

- ls, who, find
- chmod, cp, mv, rm, ln, mkdir
- date
- kill
- file, type
- echo

Séance No 3

- système de fichiers unix
- droits d'accès
- commandes de base pour manipuler les fichiers sous unix
- premier contact avec les scripts shell

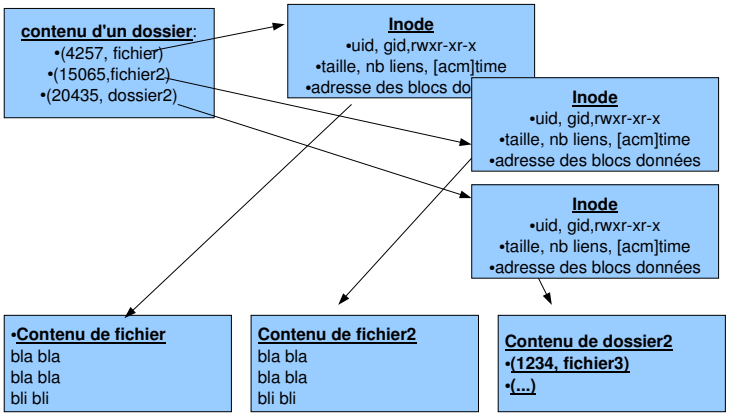
Systèmes de gestion de fichiers (SGF)

- 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

Fichiers



SGF : inode

- Inode: attributs + localisation des blocs contenant les données du fichier
- Inode:
  - Id. du disque logique où est le fichier,
  - numéro du fichier sur ce disque
  - Type de fichier et droits d'accès
  - Nombre de liens physiques
  - Propriétaire, groupe propriétaire
  - Taille
  - Dates :
    - De dernier accès (y compris en lecture): atime
    - Date de dernière modification des données: mtime
    - Date de dernier modification de l'inode: ctime

Dossier/répertoire

- Deux grandes classes de fichiers :
  - Fichier ayant un contenu sur disque : fichiers réguliers, dossiers, liens symboliques, tubes
  - Ressources : Fichiers spéciaux (pilotes de périphériques, ràf, ...)
- Dossiers: listes de couples (nom, No inode)
- Un couple est appelé « lien physique » (hardlink)
- Du point de vue de l'utilisateur, un dossier contient des fichiers (fichiers réguliers,

Inodes/Nom: conséquences

- Créer/détruire un fichier: ajouter/retirer un couple dans le dossier
- opération nécessitant un droit au niveau du dossier pas du fichier
- Le système travaille avec des No d'inode, l'utilisateur avec les noms de fichiers :
  - Ce sont les dossiers qui permettent de faire le lien entre les deux
  - On trouve le coupe (nom, inode) du dossier où est le fichier
  - Pour trouver ce dossier, on applique le même principe (pour Unix, un dossier est aussi un fichier)

Fichiers: résumé:

- ce que l'utilisateur perçoit comme un fichier identifié par un nom peut se décomposer en trois notions sous unix :
  - un inode: informations (taille, dates, uid, gid, droits) et localisation des données sur disque
  - le contenu du fichier: les données qui y sont stockées
  - un lien physique: associe un nom à un inode. Un même inode peut avoir plusieurs lien.

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 positionnner

- 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 », changement avec chmod, chgrp et

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

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 symbolique : u=rwx,g=rwx,o=rw dans notre exemple.

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

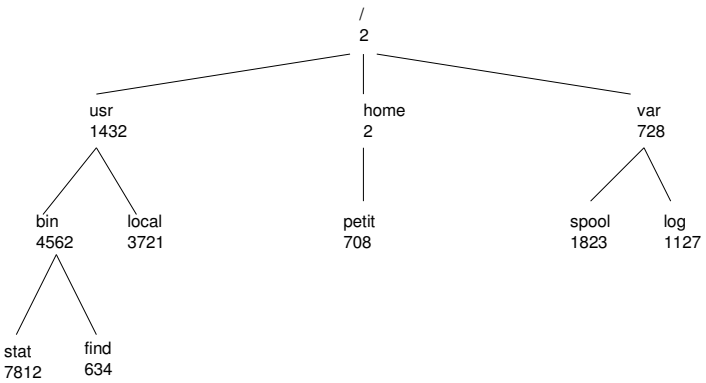
Exemples

- 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

arborescence



algo de recherche

- /usr/bin/stat
- algo de localication:
  - examiner le contenu du dossier d'inode 2 pour trouver le No d'inode du dossier usr : 1432 par exemple.
  - examiner le contenu de dossier d'inode 1432 pour trouver le No d'inode du dossier bin. 4562 par exemple
  - examiner le contenu de dossier d'inode 4562 pour trouver le No d'inode du fichier stat. 7812 par exemple

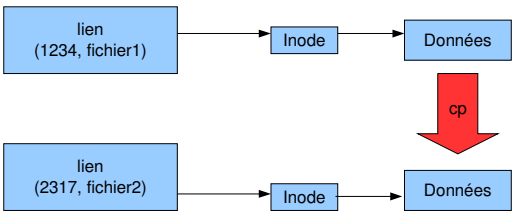
Chemin

- /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

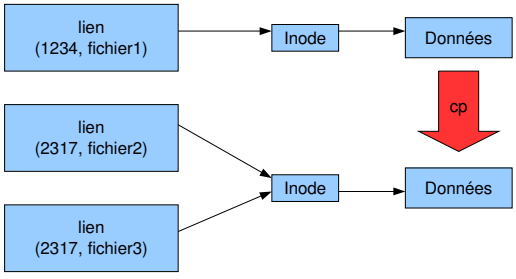
Commandes de base: cp

- 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

cp et liens physiques



cp fichier1 fichier2

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

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

Commandes de base: ln

- ln fichier nouveau_lien_physique
- ln -s fichier lien_symbolique
- options:
  - -s: crée un lien symbolique
  - -f: force la création même si la destination existe déjà
  - --: fin des options (pour permettre le traitement d'un fichier dont le nom commence par « « - »

Scripts shell

- fichier texte contenant des commandes
- nouvelle commande

lancement d'un script shell

- méthode nécessitant une accès en lecture
  - sh nomScript
  - sh < nomScript (peu utilisée)
- Méthodes nécessitant un accès en exécution et en lecture
  - nomScript ou chemin/nomScript ou ./nomScript
  - la première ligne du script qui précise le choix de l'interpréteur est de la forme:

    #! /bin/bash

•Exemple:

- considérons le script /tmp/foo.sh:

```
#! /bin/bash
pwd
cd /etc
pwd
echo coucou
```

- Utiliser les trois méthodes proposées pour le lancer
- exécuter la commande pwd après chaque lancement
- ces 3 méthodes exécutent le script dans un processus shell enfant. Comment le vérifier ?

Exemple (suite)

- ràf: une animation qui montre l'exécution du script et la création du processus
- dans une version ultérieure de ce document

Exécution par le shell courant

- on utilise la commande interne « . »
- . foo.sh
- conséquences :
  - le script est exécuté par le shell courant
  - il peut modifier les données de ce shell courant (variables, dossier de travail, ...)
- Exemple: exécuter « . foo.sh »

Erreurs classiques

- le dossier courant n'est pas dans le PATH. Indiquer le chemin absolu ou relatif. Par ex.: ./foo.sh

```
cd /tmp;foo.sh
commande non trouvée
```

- le fichier n'est pas exécutable. Utilisez chmod pour lui ajouter le droit x:

```
cd /tmp;./foo.sh
permission non accordé
```

Code de retour

- exit: termine le script en retournant un code de retour
- Rappel:
  - 0 : exécution normale
  - autre valeur: situation anormale
- sans utilisation de exit: le code de retour est celui de la dernière commande du shell

commande test

- réalise des tests simple, le code de retour indique que le test est positif ou négatif
- test -d /var/tmp : teste si /var/tmp est un dossier
- test -x /bin/ls: teste si /bin/ls est un exécutable
- test 1 = 2: teste l'égalité de deux chaînes
- forme alternative :
  - test -d /var/tmp

structure de contrôle if

- syntaxe:

```
if commande1
then
    commande2
[elif commande3
then commande4]
...
[else
    commande5]
fi
```

- si commande1 retourne 0, on exécute commande2 sinon, si commande3 retourne 0, on exécute commande4 ... sinon

Variables

- variables d'environnement
  - exportée ou non exportées
- variables utilisateurs
- variables réservées du shell

Variables d'environnement

- variables au shell et/ou au commandes lancées à partir de lui
- set : pour obtenir la liste des variables d'environnement
- obtenir la valeur d'une variable: $nomVariable ou ${nomVariable}
- changement de valeur: nomVariable=valeur
- variables exportées:
  - variables communiquées aux commandes lancées
  - env: pour obtenir la liste des variables exportées

Exemples de variables d'environnement

Nom	rôle
PATH	liste des dossiers où le shell cherche les exécutables
HOME	dossier personnel de l'utilisateur
PWD	dossier courant
TERM	type de terminal
SHELL	nom du shell en cours d'exécution
USER	
LANG	lié aux locale et à la gestion des langues
DISPLAY	« display » X Window (clavier/écran/souris)

Quelques variables d'environnement définies sous bash

variables utilisateur

- définie par l'utilisateur pour ses besoins
- nom des variables
  - premier caractère [a-zA-Z]
  - caractères suivants [a-zA-Z0-9_]
- définition: nom=mot
- valeur: $nom ou ${nom}

variables utilisateur (2)

- variables indéfinie:
  - une variable jamais initialisée est vide
  - unset permet d'annuler la définition d'une variable qui devient alors indéfinie

```
$test=hello
$echo $test
hello
$unset hello
$echo $hello
```

variables utilisateurs (3)

- espace dans les valeurs: " ou \
  - test1="m1 m2 m3"
  - test1=m1\ m2\ m3
- Attention à bien isoler le nom d'une variable:
  - f= toto;g=titi;h=$titi_$toto # $titi_ est vide
  - f= toto;g=titi;h=${titi}_$toto

Substitution de commandes

- remplacer une commande par l'affichage de son résultat d'exécution
- ancienne syntaxe à éviter : `commande`
- syntaxe moderne: $(commande)
- Exemples:

```
$echo je suis $(whoami)
je suis petit
$nom=$(whoami)
$echo je suis $nom
```

caractères de protection

- apostrophe : protège tous les caractères spéciaux sauf elle
  - echo '$HOME' affiche '$HOME'
  - echo $HOME affiche la valeur de la variable HOME
- \: protège le caractère qui suit
- guillemets : protège tout sauf lui-même, $, $() et \

variables réservées du shell

- paramètres positionnels
  - $#: nombre d'arguments
  - $0: nom du script
  - $1: valeur du premier argument
  - $2: valeur du deuxième argument
  - ...
  - $9: valeur du neuvième argument
  - ${10}: avec des shells modernes
  - $* et $@: liste de tous les arguments

variables réservées du shell (2)

- shift [n]: décale de n positions la liste des arguments vers la gauche
  - par défaut, n vaut 1
  - shift: remplace $1 par $2, $2 par $3, ...
- $?: code de retour de la dernière commande exécutée
- $$: pid du shell
- $!: pid du dernier processus lancé en arrière plan

Séance No 4

- fonctions du shell
- substitutions de noms de fichiers
- substitutions de commandes
- substitutions d'expression arithmétiques
- structure de contrôles for, while, case

fonctions du shell

- permet d'isoler un traitement particulier constitué de plusieurs commandes
- une fonction est identifiée par son nom
- elle doit être définie avant d'être utilisées. Une fois définie, une fonction est considérée comme une commande interne.
- une fonction s'utilise comme une commande: « nom liste_de_paramètres »
- la définition d'une fonction n'entraîne l'exécution d'aucune commande. C'est

définition d'une fonction

- syntaxe 1 (conseillée car la plus compatible):

```
mafonction () {
    commande1
    commande2
    ...
}
```

- syntaxe 2 (shells récents):

```
function maFonction {
    commande1
    commande2
    ...
`
```

Exemple

- considérons le script suivant:

```
#! /bin/sh
echo "1ere commande"
maPremiereFonction(){
    echo "coucou depuis maPremiereFonction"
}
echo "2e commande"
maPremiereFonction
echo "après l'appel de maPremièreFonction"
```

- son exécution provoque l'affichage suivant:

```
1ere commande
2e commande
coucou depuis maPremièreFonction
après l'appel de maPremièreFonction
```

Paramètres des fonctions

- comme avec toute commande, il est possible de fournir des paramètres à une fonction.
  - utilisation :
    - les paramètres sont ajoutés après le nom de la fonction, séparés par des espaces ou tout autre caractère séparateur
    - exemple: test premier second
  - définition de la fonction
    - les paramètres sont des variables locales à la fonction nommées: $1, $2, ..., $ç, ${10}, ...
    - les variables spéciales $#, $* et $@ sont aussi

paramètres: exemple

- script: test.sh

```
#! /bin/bash
f1 () {
    echo "1er argument de f1: $1"
    echo "deuxieme argument de f1: $2"
    echo "nombre d'arguments de f1: $#"
    echo "tous les arguments de f1: $*"
    echo "nom du script: $0"
}
echo début du script
echo "1er argument du script: $1"
echo "deuxieme argument du script: $2"
echo "nombre d'arguments du script: $#"
echo "tous les arguments du script: $*"
echo "nom du script: $0"
f1 quand est-ce "qu'on" mange
```

paramètres: exemple

- execution du script: « ./test.sh salut les tepos »

```
début du script
1er argument du script: salut
deuxieme argument du script: les
nombre d'arguments du script: 3
tous les arguments du script: salut les tepos
nom du script: test
f1 quand est-ce "qu'on" mange#! /bin/bash
1er argument de f1: quand
deuxieme argument de f1: est-ce
nombre d'arguments de f1: 4
tous les arguments de f1: quand est-ce qu'on mange
nom du script: ./test.sh
```

code de retour d'une fonction

- comme toute commande, une fonction a un code de retour
- commande « return n»: termine la fonction et retourne le code d'erreur « n ».
- Erreur à ne pas faire: *return* ne retourne pas le résultat de la fonction mais son code d'erreur

Exemple : recherche dans un annuaire

```
petit 01694780?
acces2400 0136642424
jean 0170?????
sophie 0164570101
Malik 0237463201
```

- script test.sh

```
#! /bin/bash
estDansAnnnuaire (){
   if grep $1 annuaire
   then return 0
   else  return 1
   }
if estDansAnnuaire petit
   then petit est présent dans annuaire
   else pas de petit en vue
fi
```

- exécution:

```
$ ./test.sh
```

Exemple revu (1)

- script test.sh

```
#! /bin/bash
estDansAnnuaire (){
   return grep $1 annuaire
   }
if estDansAnnuaire petit
   then petit est présent dans annuaire
   else pas de petit en vue
fi
```

- on utilise directement le code de retour de la commande grep.

Exemple revu (2)

- script test.sh

```
#! /bin/bash
estDansAnnuaire (){
   if [ $# -ne 1 ]
   then
     echo "estDansAnnuaire: utilisation sans
     argument"
      return 1
   fi
   return grep $1 annuaire
   }
if estDansAnnuaire petit
   then petit est présent dans annuaire
   else pas de petit en vue
fi
```

- programmation défensive: vérification du nombre de paramètres

Variables

- les variables utilisées dans un script sont globales.
- une exception: les variables locales définies par *typeset* (ne sont pas au programme de l'UEL).
- leur portée est dynamique: la variables est utilisable dès que le flot d'exécution l'a rencontrée.

variables locales

- ràf: dans une version ultérieure de ce document et de cet enseignement.

fonction: retourner une valeur

- une fonction à 3 modes de communication avec le monde extérieur:
  - via le code de retour : ne pas faire sauf pour une valeur binaire vrai/faux « à la grep »
  - via la sortie standard: méthode conseillée car utilisable via redirection. On se contente d'afficher le résultat sur la sortie standard
  - via une variable : la variable porte en général le nom de la fonction.

Exemples

```
#! /bin/bash
double(){
   echo $1$1
}
triple (){
   triple=$1$1$1
}
d=$(double pa)
triple pa
echo le double de pa est $d
echo le triple de pa est $triple
```

- exécution:

```
le double de pa est papa
le triple de pa est papapa
```

Substitution de commandes

- remplacer une commande par l'affichage de son résultat d'exécution
- ancienne syntaxe à éviter : `commande`
- syntaxe moderne: $(commande)
- Exemples:

```
$ echo je suis $(whoami)
je suis petit
$ nom=$(whoami)
$ echo je suis $nom
```

caractères de protection

- apostrophe : protège tous les caractères spéciaux sauf elle
  - echo '$HOME' affiche '$HOME'
  - echo $HOME affiche la valeur de la variable HOME
- \: protège le caractère qui suit
- guillemets : protège tout sauf lui-même, $, $() et \

substitutions de noms de fichiers

- les noms de fichiers paramètres de commandes du shell peuvent être cités exhaustivement ou décrits à l'aide d'expression générique (« caractères joker »)
- caractères:
  - *: une suite de caractères
  - ?: un unique caractère
  - [abcd]: l'un des caractère entre crochets
  - [a-d]: un caractère situé entre a et d
  - [^*liste*]: cité en première position, ^ est un caractère de négation: tous les caractères sauf ceux de *liste*

Exemples

- *.sh: les fichiers dont le nom finit par .sh
- p*: les fichiers dont le nom commence par la lettre p
- [A-Z]*.sh : les fichiers dont le nom commence par une majuscule et finissant par .sh
- ???: les fichiers dont le nom fait 3 lettres
- [^a-z]* : les fichiers dont le nom ne commence pas par une miniscule non accentuée

structures de contrôle: for

- syntaxe:

  for *var* in *listeValeur*

  do

  commande1

  ...

  done

- sémantique:
  - la variable *var* prend chaque valeur de la liste de valeur
  - les commandes situées entre do et done sont exécutées pour chaque valeur de la variable *var*

Exemple 1: liste de valeurs citée

```
$for i in 1 2 3 4
 do
    echo $i
done
1
2
3
4
```

Exemple 2: liste des valeurs contenues dans une variable:

```
$ varTest="a b c d"
$for i in $varTest
do
   echo $i
done
a
b
c
d
```

Exemple 3: liste des valeurs via substitution de commande

```
$for i in $(ls)
do
  echo $i
done
fichier1
fichier2
```

... (résultat de la commande ls, un fichier par ligne)

Exemple 4: liste des valeurs via génération de noms de fichiers

```
$for i in f*
do
  echo $i
done
fichier1
fichier2
```

... (les noms des fichiers commençant par f dans le dossier courant, un fichier par ligne)

structures de contrôle: case/in/esac

- syntaxe:

```
case mot in
   modele1) liste de commandes;;
   modele2|modele3|modele4) liste de
     commandes;;
   ...
esac
```

- les modèles sont des chaînes incluant éventuellement des caractères spéciaux * ? []^ (utilisés dans la substitution des noms de fichiers)

structures de contrôle: case/in/esac

- principe
  - la valeur du mot est comparée à chaque modèle
  - la liste de commandes du premier modèle correspondant est exécutée jusqu'au « ;; »
  - l'exécution se poursuit ensuite après le « esac ».

case/in/esac: exemples

```
$ cat codePostal.sh
#! /bin/bash

case "$1" in
  75[0-9][0-9][0-9])
    echo "code postal parisien"
    ;;
  7[78][0-9][0-9][0-9]|9[1-5][0-9][0-9][0-9])
    echo "code postal ile de France"
    ;;
  [0-9][0-9][0-9][0-9][0-9])
    echo "code postal de la france metropolitaine"
    ;;
  *)
   echo "code postal non reconnu"
   ;;
esac
```

structures de contrôles: while/do/done

- syntaxe

```
while commande1
do
   liste de commandes
done
```

- « commande1 » est exécutée,
  - si le code de retour vaut 0, la liste de commande est exécutée et on retourne exécuter « commande1 »
  - si le code de retour de « commande 1 » est

while/do/done: exemples

```
$ cat facto.sh
#! /bin/bash
n=5
x=1
while (( n > 0))
do
   ((x*=n))
   ((n-=1))
done
echo "12!=$x"

$ ./facto.sh
12!=120
```

while/do/done: exemples

```
$ cat test.sh
#! /bin/bash
n=1
while (( $# > 0 ))
do
  echo "parametre No $n: $1"
  ((n=n+1))
  shift
done

$ ./test.sh a g df "ze rt"
parametre No 1: a
parametre No 2: g
parametre No 3: df
parametre No 4: ze rt
```