Unix Utilisateur

ENSIAS – 2013/2014

Université Mohammed V – Souissi

Pr. Mostapha Zbakh

Plan Général

1. Introduction
2. Le système de fichiers
3. Les droits d 'accès
4. La gestion des processus
5. Les fichiers standard et la redirection d'E/S
6. La programmation du shell
7. Les filtres programmables

Plan Général (Répartition)

Semaine	Chapitre
1	Introduction
2 et 3	Le système de fichiers
3 et 4	Les droits d 'accès
5	La gestion des processus
6	Les fichiers standard et la redirection d'E/S
7	Installation et Révision
8	Contrôle
9, 10 et 11	La programmation du shell
12 et 13	Les filtres programmables
14	Révision
15	Contrôle

Plan

- Généralités
- Historique
- Caractéristiques
- Structure d'Unix
- Le cas Linux
- 1ère connexion

Généralités

- **Quelques échos :**
  - le phénomène Unix existe dans le catalogue de tous les constructeurs;
  - il existe sur toutes les gammes de machines du micro-ordinateur au super calculateur;
  - un système universel;
  - un système adopté par la communauté scientifique;
  - effort de standardisation (POSIX, X11);

Historique

- **1966:** Projet MULTICS de AT&T

  Participation de Denis Ritchie et Ken Thompson (ingénieurs au groupe de recherche Bell-Labs) au projet de mise au point du système d'exploitation à temps partagé MULTICS

  (Multiplexed Information & Computing Service)

Historique

- **1969:** Bell Labs se retire du projet
  - Mise en œuvre de MULTICS : K. Thompson et D. Ritchie développent leur version du système
  - noyau de système d'exploitation pour la famille d'ordinateurs PDP-11 (usage interne au labo)

  ⇒ 1ère version d'UNIX , nommée UNICS :

  (Uniplexed Information & Computing Service)

  ⇒ exploitation en monoprogrammation
  - Thompson a cherché à lui associé un langage plus évolué que l'assembleur

Historique

- **1970:**

  D. Ritchie a conçu et a réalisé un compilateur du langage C permettant l'écriture d'Unix dans ce langage de haut niveau.

  ⇒ Le système est à priori portable sur tout type de machine disposant d'un compilateur de ce langage.

- **1973:**

  K. Thompson et D. Ritchie ont réécrit le noyau d'Unix en C.

Historique

- **1978:** Sortie de la version 7 d'Unix

  Unix devient un produit commercialisé et ne pas seulement un thème de recherche.

  ⇒ Cette version représente l'ancêtre de la plupart des systèmes d'exploitation Unix.

  ⇒ Unix version 7, System III et System V sont des versions officielles successives des laboratoires Bell.

Historique

- **Remarques:**
  - Adaptations particulières des constructeurs :
    - HPUX, Ultrix, Scenix, Solaris, etc.
  - 3 grandes familles d' Unix :
    - Versions issues de la souche Berkeley (BSD: Berkeley Systems Development)
    - Versions respectant le standard système V : Unix Systems Laboratories, Branche commerciale de AT&T
    - Versions de recherche au sein du laboratoire Bell

Caractéristiques

- **Noyau :**
  - Temps partagé
  - Multi-utilisateurs, Multi-tâche
  - Multi-traitement (SMP)
  - Mémoire virtuelle paginée (pour chaque utilisateur)
    mémoire allouée >= mémoire physique

Caractéristiques

- **Interfaces:**
  - Interface aisée réalisée par un programme: le shell (interpréteur de commandes relativement simple).

    Définition: véritable langage de programmation possédant des instructions et des structures de contrôle de très grande puissance

  - Système interfacé avec le noyau : applications en C réalisant des appels au noyau (appels système).

    Ex : création de processus

Caractéristiques

- **Outils:**
  - Système de développement riche, mais à découvrir : plusieurs centaines d'outils (manipulation de texte, développement de logiciels, communication, etc.)
  - Système de messagerie assez complet
  - Système ouvert (pas de code propriétaire ; seules certaines implémentations propriétaires)

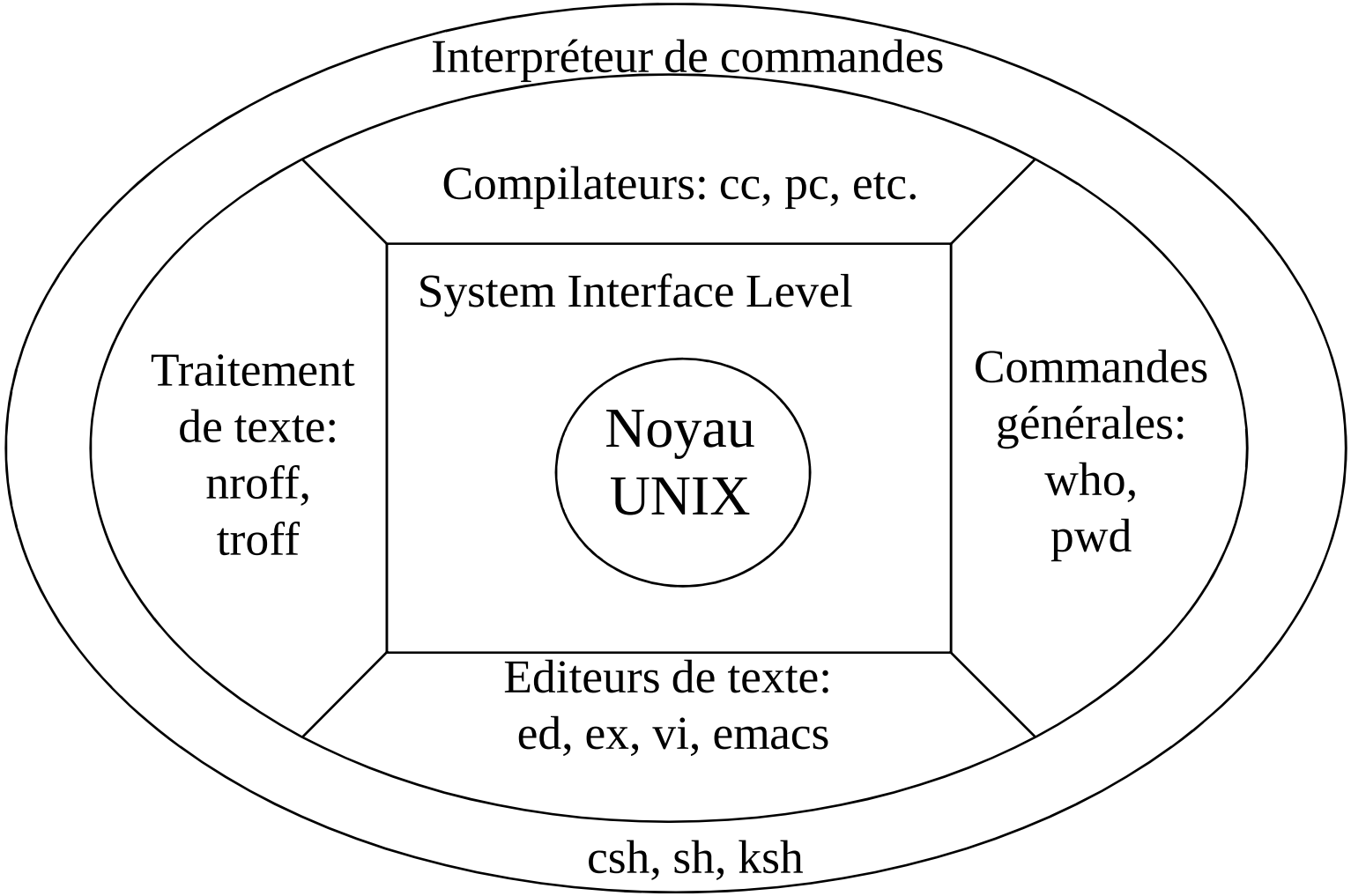
Caractéristiques

- **Parmi les défauts d'Unix :**
  - L'existence de plusieurs versions d'UNIX qui nuit à la portabilité des programmes , toutefois ce défaut tend à disparaître avec la définition de la norme POSIX en 1990
  - Il n'existe pas sous UNIX de traitement de texte puissant et simple à utiliser

Caractéristiques

- **Parmi les défauts d'Unix :**
  - Interface mode caractère peu conviviale:
    - Côté ligne de commande un peu démodé demandant un minimum d'investissement avant de pouvoir faire la moindre tâche (liste de commandes avec options).
    - Interface pourtant inégalée depuis 30 ans !
    - Depuis plus de dix ans, apparition d' interfaces graphiques se rapprochant du système Windows.

Structure d'Unix



Structure d'Unix

- **Le shell:**
  - Définition : ensemble de fonctions permettant d'exécuter les commandes de l'utilisateur
  - Possibilité d'écrire nos propres commandes contrairement aux appels système.
  - Intermédiaire entre l'homme et la machine.
  - Plusieurs versions du shell:
    - Bourne Shell : sh,
    - C Shell : csh,
    - Korn Shell : ksh,
    - etc.

Structure d'Unix

- **Les outils:**
  - Définition : utilitaires servant généralement à développer et mettre au point les programmes
  - Constitué de:
    - Commandes générales
    - Editeurs de texte (vi) ;
    - Compilateurs de programmes (langage C ; fortran ; cobol …), éditeur de liens …
    - Traitement de texte (nroff, troff, …)

Le cas Linux

- **Les forces :**
  - Linux ne s'achète pas
  - Linux n'évolue pas dans un monde isolé
  - Linux n'a pas besoin de clients
  - ….

Le cas Linux

- **Quelques paradoxes :**
  - Programme de bonne qualité: Comment le donner?
  - Produit sans garantie: Comment l'adopter en entreprise?
  - Propriétaire ne s'engage pas à faire évoluer son œuvre:
    Comment garantir le futur du programme?
  - Développement et test via communauté d'internautes:
    Comment en résulter un programme de meilleure qualité?
  - Code source visible de tous:
    Comment prétendre que la sécurité est meilleure?

Le cas Linux

- **Quelques chiffres :**
  - 25% du marché des serveurs.
  - Croissance 2.5 fois + rapide que tout autre OS
  - Fiabilité (source Microsoft) :
    - Taux de panne 2 à 5 fois plus faible que les Unix commerciaux.
    - MTBF de W2K=4 mois,
    - MTBF de NT=1 mois
    - MTBF de W98=10 jours.

Le cas Linux

- **Quelques dates importantes:**
  - 1989: FSF définit la GPL
  - 1991: version 0.02 du noyau Linux rendue publique
  - 1992: Linus Torvalds écrit Linux pour i386,
  - 1993: définition du sdf ext2fs
  - 1994: portage progressif pour Alpha, RISC, Sparc, etc
  - Mars 1994: Linux 1.0 (175.000 lignes de code),
  - Mars 1995: Linux 1.2 (310.000 lignes),
  - Juin 1996: Linux 2.0 (780.000 lignes),
  - Janvier 1999: Linux 2.2 (1.800.000 lignes),
  - 2001: Linux 2.4 (2.800.000 lignes)

Le cas Linux

- **Numérotation des versions originales:**
  - Pour le 2ème chiffre des n° de version:
    - Pair : pour utilisateurs avec des versions stables, avec ajout de nouvelles fonctionnalités éprouvées, et corrections de bugs,
    - Impair : pour les versions de développement rapide de nouvelles technologies avec un grand nombre de béta testeurs,
  - Remarques:
    - la version de développement commence dès que la version stable est suffisamment robuste
    - pour le noyau 2.4 cela aura demandé une année !

Le cas Linux: Fiche produit

- **Sous-système de communication réseau:**
  - TCP/IP v4 et v6, IPX/SPX, AppleTalk, X25, Ethernet, …
  - Serveur de messagerie SMTP, POP et IMAP
  - Serveur Web Apache avec sécurisation SSL
  - Serveur de fichiers FTP, noms DNS, news
  - Fonctions DHCP, NIS, Bootp, PAM, annuaire LDAP, Telnet, NAT, Masq, Tunneling, IP Mobile, VPN, multicast
  - Outils d'administration SNMP
  - Utilisation dédiée: routeur, pare-feu, proxy
- **Sous-système d'interface utilisateur:**
  - Architecture C/S X Window
  - Interface mode texte disponible

1ère connexion

- **Termes à connaître :**
  - ***Super Utilisateur*** : root , il s'occupe de l'administration du système UNIX (installation des logiciels, création des profiles utilisateurs, sauvegarde et restauration des données etc…)
  - ***Hôte*** (serveur) : système centrale sur lequel peuvent se connecter les utilisateurs
  - ***Terminal*** (console) : machine composée généralement d'un écran et d'un clavier, branchée directement à la machine Hôte

1ère connexion

- **Les configurations possibles**
  - Terminal
  - Architecture Client / Serveur native
  - Architecture Client / Serveur allégée (émulation de terminal)

1ère connexion

- **Les étapes d'une connexion**
  - **Pré-requis :**
    - une machine Hôte tournant sous Unix
    - des postes clients connectés soit directement (liaison série) soit par l'intermédiaire d'une connexion réseau (carte réseau et protocole TCP/IP)
    - Connexion au serveur Unix par l'instruction :
      - *telnet* <*serveur_Unix*>
    - Création préalable des utilisateurs sur l'Hôte:
      - Utilitaire < smit sous AIX >, <sysadm> sous SCO ou la commande <adduser> sous Linux etc .

1ère connexion

- **Étape 1/3 :** connexion machine (login)
  - côté serveur:
    - chargement du noyau
    - lecture du fichier /etc/rc
    - attente de connexion des utilisateurs
  - côté client :
    - séquence login/password

      information stockée dans le fichier /etc/passwd

1ère connexion

- **Étape 1/3 :** connexion machine (login)
  - une entrée dans le fichier /etc/passwd contient:
    - nom de l 'usager: son login
    - mot de passe encrypté associé
    - UID
    - GID
    - commentaire
    - home directory (répertoire de travail)
    - programme à lancer après le login (shell)

1ère connexion

- **Étape 2/3 :** la session de travail
  - initialisation du terminal
  - lecture du fichier /etc/motd
  - mise en place de l'environnement système
  - chargement du profil de l 'utilisateur
    (lecture des fichiers .login et .cshrc)
  - attente des instructions en affichant un prompt

1ère connexion

- **Étapes 3/3 :** déconnexion (logout)
  - Fermeture de session
    - 'logout' ou 'exit' ou 'ctrl+d'
- Remarques :
  - Configuration de l'environnement utilisateur
  - Introduction des commandes nécessaires dans un fichier du répertoire de connexion:
    - fichier .profile : utilisateurs du Bourne ou Korn Shell,
    - fichiers .login et .cshrc : utilisateurs du C Shell

1ère connexion

- Les premières commandes Unix:
  - syntaxe : <commande> <liste d'arguments>
  - distinction entre majuscules et minuscules
  - importance du séparateur ' '
  - le manuel en ligne : man <commande Unix>
- Application:
  - les premières commandes Unix
  - changement du mot de passe
  - changement de répertoire

Série d'exercices n°1

1. Afficher le calendrier de l'année 2009.
2. Afficher le calendrier du mois de septembre 1752, et utiliser la commande « man » pour avoir des explications sur la sortie.
3. Afficher la date au format jj-mm-aa (Exemple: 28-09-2009).
4. La commande « touch » existe, que fait-elle?
5. Afficher le nom d'hôte (hostname), le numéro de « Release », et le numéro de version de votre machine Unix.
6. Afficher les noms de « login » des utilisateurs connectés, ainsi que leur nombre.
7. Retrouver le format du fichier « /etc/passwd » (fichier de définition des utilisateurs) en partant du mot-clé password.
8. Exécuter une déconnexion, suivie d'une connexion.

Plan

- Structure de i-node
- Types de fichiers
- Désignation d'un fichier
- Arborescence du SdF
- Les liens

Structure de i-node

- **Définition :**
  - Structure d'enregistrement contenant un certain nombre d'informations d'ordre général concernant le fichier.
  - A tout fichier est associé un bloc d'informations appelé ***i-node*** (nœud d'index ou nœud d'informations)
  - Les fichiers sont groupés dans des répertoires
  - Accès à ces informations par la commande 'ls –l'

Structure de i-node

- **Composition :** champs de la structure
  - n° de i-node
  - Taille
  - Adresse des blocs utilisés sur le disque
  - Identification de son propriétaire ainsi que les droits d'accès
  - Son type
  - Compteur de références dans le système (nombre de répertoires contenu dans le répertoire courant)
  - Dates relatives aux principales opérations réalisables sur le fichier.

Types de fichiers

- **Types de fichiers :**
  - On distingue plusieurs types de fichiers :
    - Les fichiers normaux
    - Les catalogues ou répertoires
    - Les fichiers spéciaux ou fichiers périphériques

Types de fichiers

- Les fichiers normaux/ordinaires :
  - Texte (sources de programmes, courriers, scripts, etc.)
  - Programme exécutable (fichiers binaires)
- La commande file:
  - A priori, on ne peut pas savoir si un fichier contient du texte ou un programme
  - Fournit plus d'informations sur le contenu d'un fichier.

Types de fichiers

- Les répertoires :
  - Permettent d'organiser l'espace du disque dur.
  - Un répertoire peut aussi contenir d'autres répertoires (structure hiérarchique).
  - Les répertoires sont des fichiers qui contiennent des listes de fichiers.
  - *Exemple : le Répertoire de connexion ~/ :*
    - Chaque utilisateur dispose d'un répertoire particulier appelé, répertoire de travail (ou ' home directory ') où il sera placé juste après la connexion.

Types de fichiers

- Les fichiers spéciaux :
  - Représentent l'interface entre le système d'exploitation et les périphériques (/dev).
  - Toute lecture/écriture sur ces fichiers entraîne une lecture/écriture sur le périphérique associé
  - Classification:
    - F. S. mode caractère : unité d'échange = caractère

  Exemple: imprimante
    - F. S. en mode bloc : unité d'échange = bloc

  Exemple: lecteur de bande

Désignation d'un fichier

- **Structure arborescente du SdF**
  - SdF:

arbre dont la racine est le répertoire (/) et les feuilles sont des fichiers non catalogues ou des catalogues vides.

  - Répertoire:

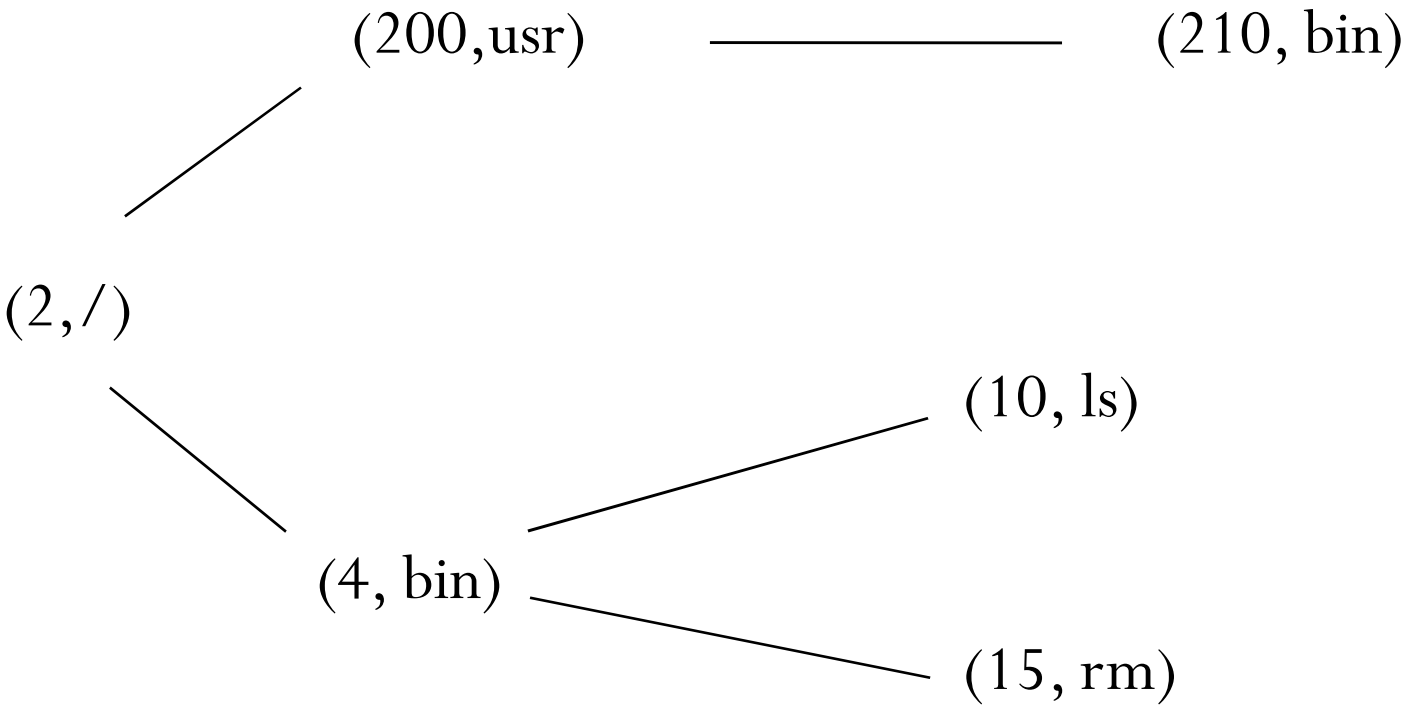
fichier dont le contenu est une suite de couples formés d'un numéro (i-node number) et d'un nom identifiant les fichiers dans le répertoire.

  - Le répertoire (/):

racine absolue (n° de i-node correspondant = 2)

Désignation d'un fichier

- Exemple :



Désignation d'un fichier

- **Noms de fichiers :**
  - Mêmes règles, quelque soit leur type :
    - UNIX distingue caractères minuscules et majuscules.
    - Longueur maximale d'un nom de fichier :

      dépend du système de fichiers 14 ou 255 caractères.
    - UNIX accepte tous les jeux de caractères pour les noms de fichiers
    - « éviter les caractères spéciaux »: risquent d'introduire des erreurs d'interprétation lors de la manipulation de ces fichiers ! @ # $ % ^ & * ( ) [ ] { } ' » \ / ; < >

Désignation d'un fichier

- **Noms de fichiers :**
  - Les caractères spéciaux :
    - Permettent d'élaborer des modèles sur les noms de fichiers :
    - * : remplace chaîne de caractères de taille non nulle
    - ? : remplace un caractère unique quelconque.
    - [ ] : remplace une liste dans une plage donnée.
    - [!] : inverse la définition de la plage de caractères.

Désignation d'un fichier

- **Noms de fichiers :**
  - Exemples:
    - [0-9] : Caractères numériques.
    - [a-z] : Caractères alphabétiques minuscules.
    - [A-Z] : Caractères alphabétiques majuscules.
    - **$ rm fich*** **$ rm fich?**
    - **$ rm fich[0-9]** **$ rm fich[!0-9]**
  - Pour désactiver un caractère spécial, il faut le faire précéder par le caractère "\" :
    - Exemple : **$ rm fich***

Désignation d'un fichier

- **Désignation par défaut:**
  - Un fichier est défini par rapport au répertoire courant de travail
  - Exemple:
    - ls : affiche la liste des fichiers et répertoires contenus dans le répertoire courant
    - vi .login: le fichier .login se trouve dans le répertoire courant de travail

Désignation d'un fichier

- **Généralisation:**
  - Le fichier peut se trouver dans un catalogue différent du catalogue courant:
    - Le même nom de fichier peut apparaître à différents niveaux de l'arborescence. Le système d'exploitation doit pouvoir différencier entre ces différents fichiers.
    - Un nom de fichier ne suffit pas: il doit être complété par l'indication de son chemin d'accès.

Désignation d'un fichier

- **Référence absolue:**
  - *Composée à partir du chemin d'accès absolu :*

Chemin établi à partir de la racine

  - *Exemple :*

/users/Ali/demos/demo1 :

référence absolue du fichier demo1

Désignation d'un fichier

- **Référence relative:**
  - *Composée à partir du chemin d'accès relatif :*

Chemin établi par rapport au répertoire courant

- Le même fichier aura des désignations différentes à partir de répertoires différents
- Deux références particulières : '.' et '..'

Désignation d'un fichier

- **Répertoire courant:**
  - Répertoire où l'utilisateur est positionné dans l'arborescence à un instant donné, sa valeur peut être connue grâce à la commande 'pwd'
- **Catalogue privé:** ' home directory '
- **Remarques:**
  - Raccourci du répertoire de travail associé à l'utilisateur pour repérages relatifs : ~/
  - Le compteur de référence d'un catalogue >=2

Arborescence du SdF

- **Les principaux répertoires:**
  - / : Répertoire racine, là où tous les autres répertoires sont montés (accrochés)
  - /bin : commandes UNIX, une partie des binaires du système et quelques commandes
    - *exemple* : ls, date, who
  - /sbin : programmes exécutables indispensables à la gestion du système.
  - /etc : quelques fichiers de configuration et des fichiers systèmes pour le démarrage
    - *exemple* : /etc/shutdown, /etc/init, /etc/passwd, /etc/group

Arborescence du SdF

- **Les principaux répertoires:**
  - /dev : fichiers unité (périphériques, spéciaux)
    - *exemple :* /dev/lp0 imprimante 0
  - /home : partie où sont stockés les fichiers propres aux utilisateurs
  - /var : fichiers temporaires de taille variable de quelques démons, de spools d'email et d'imprimantes, de logs, de locks …
  - /opt : lieu d'installation préféré des logiciels "modernes"
  - /boot : image du noyau pour Linux

Arborescence du SdF

- **Les principaux répertoires**:
  - /tmp : (temporary) fichiers temporaires, utilisés par l'éditeur de texte vi, les compilateurs…
  - /usr : espace "standard"
  - /usr/bin : pour les binaires
  - /usr/lib : (library) fichiers d'information, pour les bibliothèques du langage C
  - /usr/include : fichiers d 'entête pour programmes C (.h)

Arborescence du SdF

- **Les principaux répertoires**:
  - /usr/local : espace "non std", personnalisation locale du système
  - /usr/local/bin : rajout de binaires en local
  - /usr/local/lib : idem pour les bibliothèques
  - /usr/local/include : idem pour les "includes"
  - /usr/local/src : code source des différents programmes du système
  - /usr/man : aide en ligne

Arborescence du SdF

- **Les principaux répertoires**:
  - /mnt : (mount) montage de disquettes, donc la possibilité d'accéder aux données présentes dans la disquette à partir du répertoire /mnt, en utilisant les commandes d'UNIX
  - /lost+found : (perdu et trouvé) contient les fichiers retrouvés par la commande fsck
    - fsck :vérifie l'intégrité des données dans un SdF

Arborescence du SdF

- **Remarques**:
  - aux yeux des utilisateurs, l 'arborescence des répertoires est une entité transparente
  - beaucoup de répertoires appartenant à l 'arborescence forment différentes partitions sur un ou plusieurs disques, voire ordinateurs
  - toute ressource est rattachée à l 'arborescence du système de fichiers par un répertoire appelé point de montage

Les liens

- But
- Définition :
  - permettent de donner à un même fichier plusieurs noms (raccourci).
  - Les fichiers sont identifiés par leur i-node.
- 2 types de liens:
  - Liens ordinaires
  - Liens symboliques

Les liens

- **Les liens ordinaires :**
  - Ce ne sont pas des fichiers
    - Ils correspondent à des entrées de répertoires qui pointent vers le même i-node du fichier correspondant
    - La table des i-nodes tient à jour la liste des liens établis avec chaque fichier.
    - Le fichier d'origine n'est supprimé qu'après suppression de tous les liens.
    - La création d 'un lien : ajout d 'une étiquette dans la structure de i-node associée au fichier

Les liens

- **Les liens ordinaires:**
  - Exemple :

```
% ls –l
-rw------- 1 user g …  f1
% ln f1 f2
% ls -i
412   f1          412    f2
% ls –l
-rw------- 2 user g …  f1
-rw------- 2 user g …  f2
```

Les liens

- **Les liens symboliques :**
  - Entrée dans le SdF contenant le i-node d'un fichier lui même référence vers un autre fichier
  - Un lien symbolique peut pointer vers un fichier ou répertoire situé sur :
    - La même partition
    - Des partitions différentes sur le même disque
    - Des disques différents
    - Des ordinateurs différents
  - La destruction du fichier origine entraîne automatiquement la destruction de tous les liens

Les liens

- **Les liens symboliques:**
  - Exemple :

```
% ls –l
-rw-------        2 user g …  f1
-rw-------        2 user g …  f2
% ln –s f1 f12
% ls -i
412   f1         412   f2         414   f12
% ls –l
lrwxrwxrwx       1 user g …  f12 -> f1
-rw-------        2 user g …  f1
-rw-------        2 user g …  f2
```

Série d'exercices n°2:

1. Dans votre répertoire de connexion, créez le répertoire de nom « exercice », et dans ce dernier, les sous-répertoires « serie1 » et « serie2 ».

2. Afficher l'arborescence créée précédemment en utilisant deux commandes différentes.

- Indication : on utilisera les commandes « ls » et « du ».

3. Afficher, dans votre répertoire de connexion, la liste des fichiers en utilisant deux commandes différentes pour reconnaître les répertoires.

- Indication : on utilisera la commande « ls » avec 2 options différentes.

4. Copier le fichier « /etc/passwd » dans votre répertoire en le nommant « fic_pass ».

5. Renommez le fichier « fic_pass » en « passwd ».

6. Déplacer le fichier « password » dans le répertoire « serie_1 », sous-répertoire de « exercice ».

Série d'exercices n°2(suite):

7. Copier les fichiers « /etc/passwd » et « /etc/group » dans le répertoire « serie2 », sous-répertoire de « exercice », en étant :
   - a) dans le répertoire /etc
   - b) dans le répertoire serie2
   - c) dans un répertoire quelconque de votre choix

8. Sélectionner le sous-répertoire « serie1 » comme répertoire de travail, et listez depuis ce dernier les fichiers du sous-répertoire « serie2 ».

9. Créer le fichier « document » à l'aide de la commande « touch ». Afficher ses caractéristiques à l'aide des commande « ls » et « file ».

10. Positionnez-vous dans votre répertoire de connexion. Affichez les attributs, y compris la taille en blocs, de tous les fichurs, y compris ceux dont le nom commence par « . ».

Série d'exercices n°2(suite):

11. Afficher les attributs de votre répertoire de connexion.
12. Quelles sont les commandes qui permettent de comparer des fichiers ? Utiliser l'une d'elles pour comparer votre fichier « .profile » avec celui de l'autre utilisateur.
13. Créez dans votre répertoire de connexion, un répertoire de nom « exemples », et copier dans ce répertoire l'arborescence située sous « exercice ».
14. Supprimez l'arborescence « exercice » avec une seule commande et sans demande de confirmation pour les fichiers en lecture seule.

Plan

- Objectif
- Principe
- Exemple
- Commandes usuelles
- Droits d'accès étendus
- ACL
- La commande find

Objectif

- Gérer les données avec méthode et sécurité
- Les fichiers ne doivent être accessibles que par les personnes qualifiées
- Nécessité de positionner des droits d'accès pour chaque fichier, afin de limiter le nombre d'opérations de manipulation permises

Principe

- **Catégories d'utilisateurs:**
  - Unix distingue 3 catégories d'utilisateurs par rapport à chaque fichier :
    - *Le propriétaire* : User; Symbole : u
    - *Le groupe* : Group; Symbole : g
    - *Les autres* : Others; Symbole : o
  - Tout utilisateur est identifié par le système sur la base de son UID et GID.
  - La gestion des fichiers se base sur:
    - Le nom de l'utilisateur propriétaire du fichier
    - Le nom du groupe auquel appartient le fichier

Principe

- **Modes d'accès :**
  - Unix distingue 3 modes d'accès

  (Code d'autorisation)
    - r (read)
    - w (write)
    - x (execute)
  - Signification particulière selon type du fichier:
    - Fichier ordinaire
    - Répertoire

Principe

- **Modes d'accès :**
  - Pour les fichiers ordinaires:
    - r: le contenu du fichier peut être lu
      - exemple : la commande cp peut le lire pour le stocker sous un autre nom
    - w: le contenu du fichier peut être modifié
    - x: le fichier contient un programme qui peut être exécuté

Principe

- **Modes d'accès:**
  - **Pour les répertoires :**
    - r: Consulter le contenu du répertoire.
      - exemple : la commande ls peut afficher son contenu
    - w: Modifier le contenu du répertoire.
      - exemple : supprimer un fichier, créer des fichiers ou des répertoires
    - x: On peut entrer dans ce répertoire, qui devient notre répertoire courant

(Retirer ce droit est une manière de verrouiller un répertoire)

Exemple

- Affichage des autorisations d'accès :

$ ls -l programme

-rwxr-xr-- 1 ali invite 182 Mai 13 23:15 programme

- Pour l'accès à un fichier, plusieurs cas de figure:
  - UID(utilisateur)=UID(fichier):
    l'utilisateur = propriétaire du fichier
  - GID(utilisateur)=GID(fichier):
    l'utilisateur appartient au même groupe auquel appartient le fichier
  - UID(utilisateur)≠UID(fichier) et GID(utilisateur)≠GID(fichier):
    pas de lien entre utilisateur&fichier

Commandes usuelles

- **Modification des droits d'accès :**

$ chmod droits fichier

  - Méthode symbolique
  - Méthode octale

- **Changement de propriétaire et groupe :**

$ chown [-R] utilisateur fichier 1 [fichier2]

$ chgrp [-R] groupe fichier 1 [fichier2]

Commandes usuelles

- **Changement de groupe d'utilisateur:**
  - $ newgrp [nouveau_groupe]
  - La commande id
- **Droits d'accès par défaut:**
  - La commande umask: convention
  - Fichier/Répertoire

Droits d'accès étendus

- **SUID : Set-User-ID-Bit**
  - Définition:
    - Il n'a de sens que pour les fichiers exécutables.
    - Il donne à l'utilisateur qui lance un programme (utilisateur réel), durant le temps d'exécution, l'identité du propriétaire du fichier (utilisateur effectif).
    - Par défaut = 0
    - Valeur octale: 4000
    - Matérialisé par le "s" à l'endroit de x du propriétaire

Droits d'accès étendus

- **SUID : Set-User-ID-Bit**
  - Exemple:
    - fichier contenant les mots de passe utilisateurs **/etc/shadow** et de commande de changement de mot de passe **/bin/passwd**.
    - Si on vérifie les droits d'accès à ces deux fichiers:

```
$ ls  -l     /bin/passwd
   -r-s--x--x    1  root  system    ----------     /bin/ passwd
$ ls -l  /etc/shadow
   -rw-------    1  root  system    ----------     /etc/shadow
```

Droits d'accès étendus

- **SUID : Set-User-ID-Bit**
  - Exemple:
    - Un utilisateur peut changer son mot de passe en utilisant la commande /bin/passwd, qui entraîne une modification dans le fichier /etc/shadow.
    - Si on suit les règles de restriction des autorisations d'accès, la modification du mot de passe est impossible pour tout utilisateur autre que le root.
    - Le temps du traitement, affecte à l'utilisateur qui a lancé la commande /bin/passwd les mêmes droits que celui du propriétaire (root).

Droits d'accès étendus

- **SGID : Set-Group-ID-bit**
  - De la même façon on peut placer le SGID-bit sur un programme, et dans ce cas, un utilisateur qui lance ce programme se verra attribuer les mêmes privilèges que les membres du groupe propriétaire du fichier.
  - Valeur octale 2000
  - Matérialisé par le "s" à l'endroit du groupe.
  - Remarque:
    - Pour un répertoire, le SGID permet aux fichiers créés ds rép. d'appartenir au groupe du rép.

Droits d'accès étendus

- **Positionnement des droits d'accès étendus:**
  - *Méthode symbolique:*
    - $ chmod u+s programme1
    - $ chmod g+s programme2
  - *Méthode Octale:*
    - Les autorisations d'accès étendues sont mentionnés à la position des milliers du chiffre en base 8 :
      - UID - BIT 4000
      - GID - BIT 2000

Droits d'accès étendus

- Remarque:
  - Comme pour les droits d'accès, les droits d'accès étendus ne peuvent être affectés que par le propriétaire du fichier ou par l'administrateur du système.

Droits d'accès étendus

- **Sticky-Bit:**
  1. Pour les fichiers :
     - Permet d'optimiser le temps de réponse

```
$ ls  -l    /bin/ sh     /bin/ ls      /bin/ vi
-rwx--x--t      6 bin    bin    17884   Nov 21    05 : 53   /bin/ls
-rwx--x--t      2 bin    bin    38001   Nov 18    15 : 03   / bin/sh
-rwx--x--t      5 bin    bin    126624  Nov 21     03 : 45  / bin/vi
```

     - Il faut limiter le nombre de programmes auxquels le mode sticky est affecté: seul le gestionnaire l'assigne à un fichier (l'espace mémoire étant précieux).

Droits d'accès étendus

- **Sticky-Bit:**
  1. Pour les répertoires :
     - Ce bit positionné sur un répertoire permet de le faire partager entre plusieurs utilisateurs, sans qu'un utilisateur ne puisse supprimer les fichiers créés par d'autres utilisateurs même si les autorisations d'accès permettent l'action de suppression.
     - C'est le cas du répertoire /tmp.
     - Valeur octale: 1000

3. La commande find

- Structure et technique de travail:

  3 indications

  - A partir de quel répertoire commencer la recherche
  - Quels sont les critères de recherche à mettre en œuvre
  - Que doit-il se passer si un fichier répond à ce critère
- Syntaxe:
  - find répertoire [-critère [argument_critère]] …

3. La commande find

- Critères de sélection:

-name *nom du fichier*

-type f, d, c, b, p, s, l

-size +-*valeur*cbk : taille >,< à valeur (car, blocs, ko)
sans signe, par défaut = taille

-user *propriétaire*

-group *groupe*

-perm +-*droits* : au plus/moins les droits (rwx)

-ctime *nbjours* : status fichier modifié depuis nbjrs

-mtime *nbjours* : dernière modif. remonte à nbjrs

-atime *nbjours* : dernier accès remonte à nbjrs

-links

3. La commande find

- Critères d'exécution:
  - –print Affiche le chemin d'accès
  - –exec cmde {} \; Exécute cmde avec comme argument le fichier
  - –ok cmde {} \; Demande une confirmation pour exécuter la cmde avec comme argument le fichier

3. La commande find

- Opérateurs logiques:

Opérateur logique	Expression dans la commande
ET	-a
OU	-o
NON	!
(expression)	\(expression \)

Plan

- Généralités
- Classification
- Propriétés
- Caractéristiques
- Les signaux
- Foreground/Background

Généralités

- Définition: Appelé aussi ‘tâche’
  - Toute exécution d ’un programme à un instant donné.
  - Le programme lui-même=objet inerte du SdF (fichier exécutable)
  - Sélection par le processeur à chaque nouvelle tranche de temps d’un nouveau processus pour qu’il soit exécuté
- Conséquence:
  - L ’exécution de l ’interpréteur de commandes est elle-même un processus
  - Le shell: données (*setenv*) + commandes (*instructions*)

Classification

- Processus système:
  - créés au lancement du système

    ou
  - à des dates fixées par l'administrateur du système

Classification

- Processus système:
  - Les premiers ne sont interrompus qu ’à l ’arrêt du système.
    - Exemple: inetd, routed, named, ftpd, httpd
    - Localisation: /etc/rc.d

Classification

- Processus système:
  - La commande *at* ne lance qu'une seule fois une commande à une heure particulière
  - Syntaxe: at date < commande>
  - **Exemple**:

```
$ at now +4 hours
 at> echo « Bonjour » > /tmp/message
 at>^D

$ at  4am
at>touch  test.txt
at>^D
```

$at 3:30am Apr 10
at> echo « C'est l'heure » > /dev/message
at>^D

$ at now +3 minutes
find /home/jpp -name "core" -exec rm {} \;
at> ^D
$
$ *at –l* : visualise la liste des processus programmés avec at

Classification

- Processus système:
  - La commande *crontab* permet l'exécution périodique et automatique.
  - Chaque ligne doit comporter 6 colonnes, si une colonne contient une astérisque, le champs devient non discriminant.
    - 1ère : minutes (0-59) (*/5=toutes les 5 minutes)
    - 2e : heures (0-23)
    - 3e : jour du mois (1-31)
    - 4e : mois (1-12)
    - 5e : jour de la semaine (0-6,0 pour dimanche)
    - 6e : nom de la commande ou du script
  - Syntaxe:

```
$ crontab min heure_jour jour_mois mois_année jour_semaine action
```

Classification

- Processus système:
  - crontab accepte deux options:
    - -l : lister les commandes déjà enregistrées
    - -e : éditer la liste des commandes enregistrées.

      édition sous vi par défaut,
  - Syntaxe:

    min heure_jour jour_mois mois_année jour_semaine action
  - Exemple:

    $ crontab -l

    20 3 8 * * /usr/local/bin/mon_script_de_sauvegarde

    0 8 * * 0 /home/jpp/bin/recup_mail

Classification

- Processus utilisateurs:
  - lancés depuis un terminal donné à une date donnée
  - exemple: login=lancement d 'un processus qui exécute l 'interpréteur de commandes (csh)
- Exemple:
  - la commande *sleep X* : passer un certain délai entre deux commandes, permettra de différer de X secondes l'exécution de la commande suivante.

```
$ echo debut;sleep 10;echo fin
debut


fin
```

Propriétés

- Lancement d 'un processus = Chargement en mémoire centrale (à partir du disque) du programme correspondant en vue de son exécution
- Image mémoire d'un processus:
  - 4 composantes
  - 2 parties

Propriétés

- Les programmes Unix sont réentrants:
  - Si plusieurs utilisateurs Unix demandent l 'exécution du même programme, alors une seule copie est placée en mémoire centrale.
  - Le système assure à chacun des processus la gestion d 'une zone de données propre à chacun

Caractéristiques

- identificateur (PID)
- propriétaire
- groupe propriétaire
- terminal d'attachement
- nom du programme
- PID du processus ayant créé ce processus (PPID)
- autres attributs

Caractéristiques

- La commande ps:
  - Syntaxe :
    - La commande ps (-u, -l, -e, -f)
  - Exemple:

```
$ ps
  PID      TTY       TIME        COMMAND
  882       0        00 : 01       -csh
  893       0        00 : 00        ps
```

Sachant que:

- TIME: Durée de traitement en secondes
- PID: processus
- TTY: terminal d'attachement
- COMMAND. la commande exécutée.

Caractéristiques

- Table des processus:
  - chacun des processus possède une entrée dans cette table contenant toutes les informations nécessaires au système lorsque le processus n 'est pas actif
- Enchaînement de processus:
  - exemple: ls; echo salut; date

Caractéristiques

- Hiérarchie de processus
  - PID
  - PPID
  - ps ‘init’: (PID=1, PPID=0)
- Exemple:
  - Fonctionnement de l’interpréteur de commandes

Les signaux

- **Arrêter un processus:**
  - La commande **kill:** elle envoie un signal au processus pour mettre fin à son activité.
  - Cette commande a besoin de deux informations pour s'exécuter :
    - Le numéro de signal
    - Le numéro du processus qui sera désactivé.
- Syntaxe:

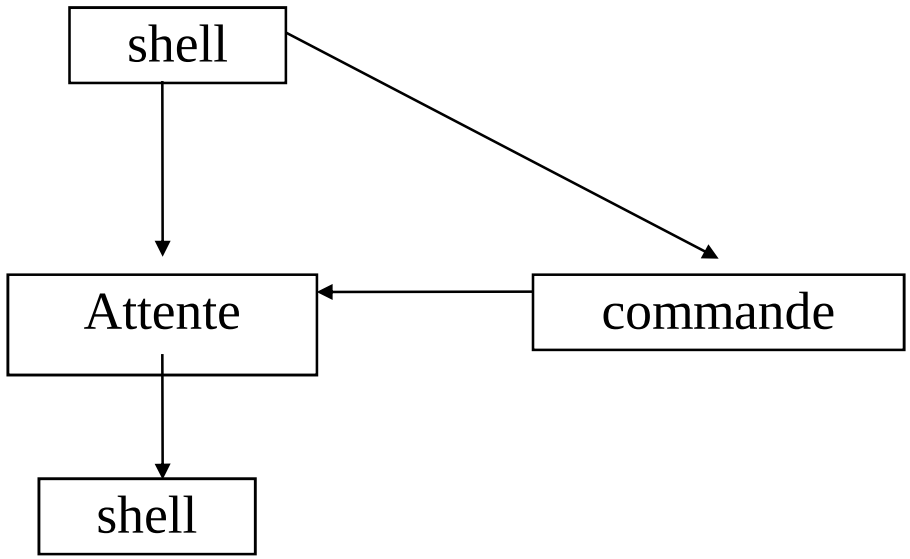
```
$ kill  [Numéro de signal] PID …
$ kill -l
```

Les signaux

- **Les signaux les plus utilisés:**
  - 1 (SIGHUB): il est envoyé par le processus père à tous les processus fils quand il termine son exécution
  - 2 (SIGINT): il est envoyé par la combinaison de plusieurs commandes envoyées par l'utilisateur (^C).
  - 9 (SIGKILL): il ne peut être ignoré par aucun processus. C'est le frein à tous les processus parasites.
  - 15 (SIGTERM): il est utilisé par défaut par la commande kill pour mettre fin à un processus.

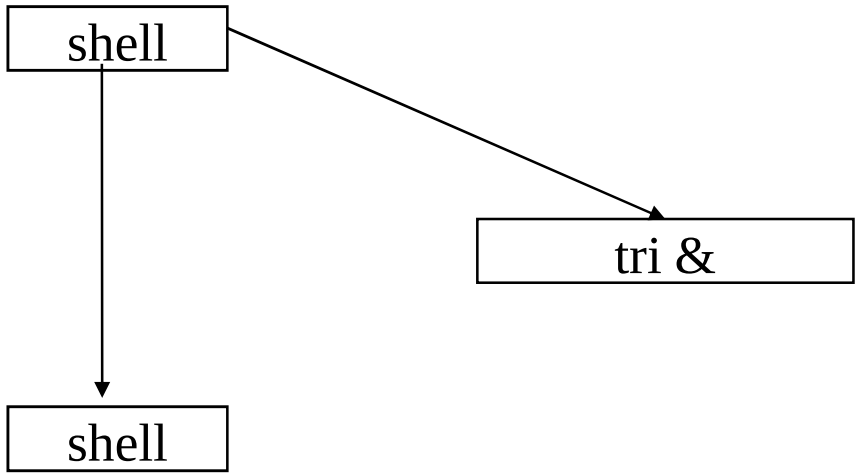
Foreground/Background

- Fonctionnement du shell:



Foreground/Background

- Lancement de processus en mode détaché:
  - Dit aussi arrière plan ou batch
  - Syntaxe: '&'
  - Contrainte: entrée standard inaccessible
  - jobs
  - kill %n
  - fg %n
  - bg %n



Foreground/Background

- Lancement de processus en mode détaché:
  - La commande nohup:
    - elle empêche que les tâches en arrière plan ne soient tuées au moment de la déconnexion du shell.
    - L'output de la commande ira dans le fichier *nohup.out*

Syntaxe: nohup commande … &

Foreground/Background

- Lancement de processus en mode détaché:
  - La commande nice:
    - elle permet de faire exécuter une tâche d'arrière plan à plus faible priorité que la normale
    - destinée aux tâches d'arrière plan effectuant beaucoup de calcul

Syntaxe: nice commande … &

Foreground/Background

- Lancement de processus en mode détaché:
  - La commande wait [PID]:
    - La commande « wait » permet de suspendre l'exécution d'un shell script jusqu'à ce que le processus, dont le PID est spécifié en argument, se termine.
    - Si aucun PID n'est spécifié, on attendra, dans ce cas, que tous les processus lancés en arrière plan soient terminés.
  - Syntaxe: wait

Foreground/Background

- Exemple:

```
$cat  /usr/man1   /etc/passwd  > file1 &
$find  /  -print > file2 &
$wait
$echo  "cat  et  find  sont termines ..."
```

- Stopper un processus: ^Z

Foreground/Background

- Lancement de processus en mode détaché:
  - La commande top:
    - souvent installée par défaut
    - permet d'identifier les processus les plus consommateurs en temps CPU et en mémoire.

```
8:50pm     up 30 min,        6 users,       load average: 1,29,   2,05,   1,68
61 processes: 60 sleeping, 1 running, 0 zombie, 0 stopped
CPU states: 0,1% user, 0,3% system, 0,0% nice, 99,4% idle
Mem: 256932K av, 150520K used, 106412K free, 0K shrd, 15312K buff
Swap:          0K av,          0K used,          0K free                67288K cached

PID USER PRI NI SIZE RSS SHARE STAT %CPU %MEM TIME COMMAND
1956 root   15    0   1032 1032      820  R           7,3        0,4   0:00   top
1812 root   12    0 13336 13M    11488  S           0,1        5,1   0:05   kdeinit
    1 root    8    0    484   484       420  S           0,0        0,1   0:04   init
```

SERIE N 4

1)- Exécutez la commande « ps » et donner la signification de chacune des colonnes affichées.

2)- Créer un programme qui affiche le message « Salut » toutes les 10 secondes. Lancer ce programme en arrière plan, et afficher son PID, puis son numéro de job.

3)- Tuer ce programme en utilisant son PID.

4)- Lancer à nouveau ce programme en détaché du terminal (non sensible à la déconnexion) , où écrit-il ses sorties?

5)- Déconnectez vous, et connectez-vous. Affichez vos processus en tapant: ps
Le processus Salut n'apparait pas, pourquoi?

6)- Quelle commande devez-vous exécuter pour afficher le processus qui exécute le processus Salut (ps –u user).

7)- Tuez le processus Salut et détruisez son fichier de sortie .

8) – Supprimez de votre répertoire les fichiers temporaires dans 1 minute .

9)- Supprimez de votre répertoire les fichiers temporaires chaque dimanche à 10 h

.

10) Supprimez de votre répertoire les fichiers temporaires à 4 h du matin.

Plan

- Entrée/Sortie standard
- Redirection d'E/S standard
- Redirection de la sortie standard
- Redirection de l'entrée standard
- Redirection de la sortie erreur standard
- Les tubes de communication

E/S standard

- **Les fichiers d'E/S Standards :**
  - Toutes les commandes utilisent les canaux d'E/S pour lire ou transmettre leur résultat.
  - Ces canaux portent des noms spéciaux :
    - Canal d'entrée standard (0) : Clavier par défaut
    - Canal de sortie standard (1) : Écran par défaut
    - Canal de sortie erreur standard (2) : Écran par défaut

Redirection d'E/S standard

- **Les fichiers d'E/S Standards :**
  - En interne, toute utilisation d'un canal d'E/S se traduit par une utilisation de son fichier spécial associé.
  - Une des principales caractéristiques du shell

Redirection de la sortie standard

- **Définition**
- **Syntaxes:** $ commande > fichier_dest

  $ commande 1> fichier_dest

  « fichier_dest » contient le résultat d'exécution de commande
- **Exemple :**

$ who > connect

  - Le fichier 'connect' contient la liste des pers connectées
  - Si le fichier 'connect' n'existait pas, alors il sera créé

    Sinon il sera écrasé et remplacé par le résultat de 'who'

Redirection de la sortie standard

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

Redirection de l'entrée std

- **Définition**
- **Syntaxes :** $ Commande < fichier_source

  $ Commande 0< fichier_source

« fichier_source » contient les arguments de la cmd

- **Exemple :**

$ wc -l < connect

Permet de compter le nombre de lignes dans 'connect'.

Redirection de l'entrée std

- **Définition**
- **Syntaxe :** $ Commande << chaine

  Permet d'envoyer à « commande » toutes les lignes tapées en entrée, jusqu'à la chaîne saisie, méthode du "here document"
- **Exemple :**

```
$ wc -l << fin
"collez quelque chose avec la souris"
fin
1
$
```

Permet de compter le nombre de lignes dans la chaîne saisie avant « fin ».

Redirection de la sortie erreur std

- **Définition**
- **Syntaxe :** $ Commande 2> fichier_erreurs

  « fichier_erreurs » contient les messages d'erreur associés à l'exécution de la commande.
- **Exemple :**

$ cc programme.c 2> erreurs

Les erreurs de compilation du fichier programme.c seront redirigées vers le fichier erreurs.

Les tubes de communication

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

Les tubes de communication

- Exemple:

$ who | wc -l (**)

$ who : liste de personnes connectés au système.

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

La commande (**) permet de compter le nombre de personnes connectés, elle est équivalente à la ligne de commande suivante:

$ who>tmp ; wc -l tmp ; rm tmp

Les tubes de communication

- **La commande tee :**
  - En utilisant des filtres le résultat de la commande1 n'est pas visualisé à l'écran, pour pouvoir le visualiser on utilisera des tuyaux : commande **tee**.
  - *Syntaxe:*

    $ commande1 | tee fichier1 | commande2
  - Redirige le résultat intermédiaire de commande1 vers fichier1. Ce même résultat sera traité par la commande commande2.

Les tubes de communication

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

```
$ ls | grep poème | tee fichier1 | wc -l
  1        2          3             4
```

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

Les tubes de communication

- **xargs et sh:**
  - La commande *xargs* peut aider à exécuter un flux:
    - -t pour avoir l'écho des commandes et
    - -n pour découper le flux d'entrée en paquets :
  - *Exemple:*

```
$ xargs -t -n 2 diff <<fin
fic1 fic2 fic3
fic4 fic5 fic6
fin
diff fic1 fic2
diff fic3 fic4
diff fic5 fic6
```

Les tubes de communication

- **xargs et sh:**
  - La commande *xargs* peut aider à exécuter un flux :
    - -i pour insérer le flux
  - Exemple:

```
$ ls | xargs -t -i{} mv {} {}.old
mv chap1 chap1.old
mv chap2 chap2.old
mv chap3 chap3.old
```

Plan

- Généralités
- Le profil d'utilisateur
- Le shell en interactif et les scripts shell
- Les variables et les chaînes de caractères
- Les instructions de contrôle
- Les alias, les fonctions, et l'arithmétique
- La gestion de fichiers
- La programmation multi-tâches en shell
- Quelques commandes utiles

Généralités

- Shell = Interpréteur de commandes
- **Définition**
  - Programme gérant l'interface entre l'utilisateur et le système
  - Tout dialogue avec la machine se fait par l'intermédiaire du Shell (pas de compilation)
- **Caractéristiques**
  - 1 exemplaire particulier du Shell par utilisateur
  - Il peut lancer +ieurs Shells sur sa session

CHAPV La programmation du shell

- Généralités
- Le shell en interactif et les scripts shell
- Les variables et les chaînes de caractères
- Les instructions de contrôle
- Les alias, les fonctions et l'arithmétique
- La gestion de fichiers

Généralités

- Shell = Interpréteur de commandes
- **Définition**
  - ✓ Programme gérant l'interface entre l'utilisateur et le système
  - ✓ Tout dialogue avec la machine se fait par l'intermédiaire du Shell (pas de compilation)
- **Caractéristiques**
  - ✓ 1 exemplaire particulier du Shell par utilisateur
  - ✓ Il peut lancer plusieurs Shells sur sa session

Généralités

- sh ("Bourne shell", Steve Bourne, AT&T)
  - shell historique (/usr/old/bin/sh, ou bsh)
- csh ("C-shell", Bill Joy, UCB)
  - shell BSD (disponible sur tous les Unix)
- ksh ("Korn shell", David Korn, AT&T)
  - Complète le shell Bourne et intègre avantages du C-shell
- sh (shell normalisé POSIX P1003.2 et ISO 9945.2)
  - Dérivé du shell Bourne et intègre fonctionnalités ksh
- bash ("Bourne again shell" de Brian Fox & C. Ramey)
  - shell libre de droit, compatible shell POSIX (par défaut Linux)
- zsh ("Zero shell")
  - ksh du domaine public (installé sur toutes les machines)
- rsh ("Bourne shell réduit", cd, <, > inopérants)
- tcsh ("Toronto C-shell")

Généralités

Algorithme du shell

- Ouverture initiale de 3 fichiers standard
  - entrée standard (stdin, fd=0), par défaut le clavier
  - sortie standard (stdout, fd=1), par défaut l'écran
  - sortie erreur standard (stderr, fd=2), par défaut l'écran
- Le shell lit sur son entrée standard
  - lecture de la commande jusqu'à la fin de ligne (\n) et interprétation des métacaractères et opérateurs .
  - lecture des lignes de commandes jusqu'à la fin de fichier (^D)

Généralités

- **Format d'une commande:**
  - Invite (le prompt) principale
    - $ en mode utilisateur
    - # en mode super-utilisateur
    - Variable PS1
  - Invite secondaire
    - > en mode interactif

Généralités

- **Format d'une commande:**
  - Commande simple
    - une commande simple est une séquence de mots séparés par un séparateur blanc et de redirections.
    - Le premier mot désigne le nom de la commande à exécuter, les mots suivants sont passés en arguments à la commande.
    - la valeur retournée d'une commande est celle de son exit.
  - Chemin de recherche du nom d'une commande : PATH ou path (csh)
    - PATH=/usr/ucb:/bin:/usr/bin:/usr/local/bin:~/bin:.

Généralités

- **Enchaînement de commandes:**
  - Commande simple (en premier plan)

    $ commande [arg …]
  - Commande simple (en arrière plan)

    $ commande [arg …] &
  - Pipeline (tube de communication)

    $ commande1 [arg …] | commande2 [arg …] | …
  - Séquencement de commandes (en premier plan)

    $ cmd1 [arg …]; cmd2 [arg …]; cmd3 …

Généralités

- **Enchaînement de commandes:**
  - Exécution par un nouveau shell ()

    $ sh commande [arg …]
  - Regroupement de commandes

    $ (com1 [arg …]; com2 [arg …]; com3 …)

    $ (com1 [arg …]; com2 [arg …]; com3 …)&

Le shell en interactif et les scripts shell

- Le shell en intéractif:
  - Les jokers: ?, *, []
  - Les caractères d'échappement: \ ‘ "
    - Simples quotes '…' : les caractères inclus entre 2 simples quotes ne sont pas évalués, ils conservent leur valeur littérale.
    - Doubles quotes "…" : les caractères inclus entre 2 doubles quotes conservent leur valeur littérale à l'exception de $ ` et \.
  - Les redirections: > >> < << 2> 2>>
  - Les tubes |
  - Le remplacement de commandes:

    `cmd` ou $(cmd)

Le shell en interactif et les scripts shell

- Les shell-scripts:
  - Les commentaires:
    - #!/bin/bash
    - # @(#) résultat cmd what
  - Donner le droit d'exécution:
  - chmod +x shell-script
  - Exécution d'un script
    - bash shell-script
    - bash < shell-script
    - ./shell-script

Le shell en interactif et les scripts shell

- Exemple d'un script bash:

```
#!/bin/bash
# premier script bash
echo  '' Hello World ''
```

Les variables

- **Les types de variables :**
  - Variables prédéfinies:
    - lors de la connexion
  - Variables positionnelles:
    - paramètres d'un script
  - Variables spéciales du shell:
    - maintenues par le shell
  - Variables créées par l'utilisateur

Les variables

- **Syntaxe:**
  - Déclaration:
    - variable=valeur
  - Utilisation:
    - variable2=$variable
    - variable2=${variable}
  - Liste des variables:
    - set
  - Suppression des variables:
    - unset nom

Les variables

- **Exemple 2:**

```
#!/bin/bash
x=10
chaine=’’ Bonjour tout le monde’’
echo  $x
echo $chaine
```

Les variables

- **L'environnement:**
  - Ensemble des variables dont un shell fournit une copie à la descendance
  - La commande export :
    - Syntaxe:

      export [variable[=valeur]]

    - Exemple:
      - export PATH=/home/user01/mpich2-install/bin:$PATH

Les variables

- **Les variables d'environnement (créées lors de la connexion):**
  - HOME chemin d'accès au répertoire initial de l'utilisateur
  - PATH suite de chemins d'accès aux répertoires des exéc.
  - PS1 invite principale du shell en mode interpréteur
  - PS2 invite secondaire du shell en mode programmation
  - IFS séparateurs de champ des arguments
  - MAIL chemin d'accès à la boîte aux lettres utilisateur
  - MAILCHECK intervalle en sec au bout duquel le mail est contrôlé
  - CDPATH liste de chemins d'accès pour la commande cd
  - ENV nom du fichier des variables d'environnement
  - TERM nom du type de terminal

Les variables

- **Les variables positionnelles:**
  - $0 nom du script (pathname)
  - $1,…, $9, $10,… arguments (du 1er au 9ème, 10ème,…)
  - $# nombre d'arguments
  - $* liste de tous les arguments
  - $@ liste de tous les arguments
  - $? code retourné par la dernière commande
  - $$ numéro de processus de ce shell
  - $! numéro du dernier processus en arrière plan

Les variables

- **Application:** Qu'affiche le script test.bash suivant?

```
#!/bin/bash
echo "nom du script=$0"
echo "nombre de parametres : $#"
echo '$*'=$*
echo '$@'=$@
```

Les variables

- **Les variables positionnelles:**
- La commande shift: décale les paramètres de n positions vers la gauche (par défaut 1)

  Syntaxe: shift [n]
- La commande set: remplace les paramètres du script par les arguments de la commande, ainsi que les variables #, * et @

  Syntaxe: set argument1 …
  - Exemple:

Les variables

- **Exemple2:** Qu'affiche le script suivant?

```
#!/bin/bash
echo "nombre de parametres : $#"
echo "param1=$1"
echo "param2=$2"
echo "param3=$3"
echo "param4=$4"
echo "param5=$5"
shift 3
echo "apres 'shift 3' il nous reste $# parametres"
echo "param1=$1"
echo "param2=$2"
echo "param3=$3"
echo "param4=$4"
echo "param5=$5"
```

Après l'exécution: $./test.bash a b c 100

Les variables

- **Exemple3:** Qu'affiche le script suivant?

```
#!/bin/bash
nb=25
mon_texte="bonjour tout le monde"
mon_texte_2="coucou"
echo $nb
echo $mon_texte
echo $mon_texte_2
let $[ nb=$nb+1 ]
let $[ mon_texte=$mon_texte+1 ]          # erreur faite
exprès !!
```

Les variables

- **L'instruction read:**

  Syntaxe: read variable …

- **L'instruction readonly:**

  Crée une variable qu'on ne peut plus supprimer ni modifier

  Syntaxe: readonly variable=valeur

Les variables

- **Exemple4:** Qu'affiche le script suivant?

```
#!/bin/bash
echo -n "entrez ce que vous voulez : "
read val1 val2
echo val1=$val1
echo val2=$val2
readonly val=10
echo $val
 unset val
echo $val
```

Les variables

- **Le remplacement de variables:**

variable=$var : si var ∃ ou != 0 alors var sinon rien

variable=${var} : si var ∃ ou != 0 alors var sinon pas d'affectation

variable=${var:-val} : si var ∃ ou != 0 alors var sinon val

variable=${var:=val} : si var ∃ ou != 0 alors var sinon val et var ⇐ val

variable=${var:?mess}: si var ∃ ou != 0 alors var sinon affiche 'mess'

variable=${var:+val} : si var ∃ ou != 0 alors val sinon aucune substitut°

Les chaînes de caractères

- **La manipulation de chaînes:**
  - Calcul de la longueur d'une chaîne
    - expr chaîne:".*"
    - Exemple:

```
$ expr "bonjour":".*"
7
```

  - Extraction d'une sous-chaîne
    - \(..\)
    - Exemple:

```
$ Temps="Il fait beau"
$ expr "$Temps":"\(..\)"
Il
$ expr "$Temps":".*\(....\)"
beau
```

```
$ chemin=/usr/bin/ls
$ expr "$chemin":".*/\(.*\)"
ls
$ expr "$chemin":"\(.*\)/"
/usr/bin
```

  - Les commandes basename $chemin et dirname $chemin

Les chaînes de caractères

- **Le remplacement de variables:**

variable=${#var} : longueur en octet de la chaîne var
variable=${var#str} : supprime en début de var la + petite des 2 chaînes
variable=${var##str} : supprime en début de var la + grande des 2 chaînes
variable=${var%str} : supprime en fin de var la + petite des 2 chaînes
variable=${var%%str} : supprime en fin de var la + grande des 2 chaînes

Les instructions de contrôle

- **Familles de commandes avancées:**
  - ✓ Sélection
  - ✓ Aiguillage
  - ✓ Itération
  - ✓ Rupture de boucle
  - ✓ Commande de test

Les instructions de contrôle

- **Sélection**
  - Syntaxe :
    - Condition simple: **if then ... fi**

if *commande* ; **then**
liste de commandes
fi

    - Condition avec alternative: **if then ... else ... fi**

if *commande* ; **then**
liste de commandes
else
liste de commandes
fi

Les instructions de contrôle

- **Sélection**
  - Syntaxe :
    - Condition multiple: **if then ... elif then ... fi**

if *commande* ; **then**
liste de commandes
elif *commande* ; **then**
liste de commandes
else
liste de commandes
fi

Les instructions de contrôle

Aiguillage

- Aiguillage case: **case in ... esac**

case *$variable* **in**
*motif1***)** liste de commandes **;;**
*motif2***)** liste de commandes **;;**
***)** liste de commandes **;;**
esac

- **Exemple :**

```
case $# in
        1) arg1=$1;;
        2) arg1=$1; arg2=$2 ;;
        0) echo "usage: $0 arg1 [ arg2 ] exit 1;;
esac
```

Les instructions de contrôle

Tests sur les fichiers/répertoires:

Voici une liste des tests possibles sur les fichiers et/ou répertoires :

"-e fichier" : vrai si le fichier/répertoire existe.

"-s fichier" : vrai si le fichier à une taille supérieure à 0.

"-r fichier" : vrai si le fichier/répertoire est accessible en lecture.

"-w fichier" : vrai si le fichier/répertoire est accessible en écriture.

"-x fichier" : vrai si le fichier est exécutable ou si le répertoire est accessible.

"-O fichier" : vrai si le fichier/répertoire appartient à l'utilisateur.

159

Les instructions de contrôle

"-G fichier" : vrai si le fichier/répertoire appartient au groupe de l'utilisateur.

"-b nom" : vrai si nom représente un périphérique (pseudo-fichier) de type bloc (disques et partitions de disques généralement).

"-c nom" : vrai si nom représente un périphérique (pseudo-fichier) de type caractère (terminaux, modems et port parallèles par exemple).

"-d nom" : vrai si nom représente un répertoire.

"-f nom" : vrai si nom représente un fichier.

160

Les instructions de contrôle

"-L nom" : vrai si nom représente un lien symbolique.

"-p nom" : vrai si nom représente un tube nommé.

"fichier1 -nt fichier2" : vrai si les deux fichiers existent et si fichier1 est plus récent que fichier2.

"fichier1 -ot fichier2" : vrai si les deux fichiers existent et si fichier1 est plus ancien que fichier2.

"fichier1 -ef fichier2" : vrai si les deux fichiers représentent un seul et même fichier.

161

Les instructions de contrôle

Tests sur les entiers

"entier1 -eq entier2" : vrai si entier1 est égal à entier2.

"entier1 -ge entier2" : vrai si entier1 est supérieur ou égal à entier2.

"entier1 -gt entier2" : vrai si entier1 est strictement supérieur à entier2.

"entier1 -le entier2" : vrai si entier1 est inférieur ou égal à entier2.

"entier1 -lt entier2" : vrai si entier1 est strictement inférieur à entier2.

"entier1 -ne entier2" : vrai si entier1 est différent de entier2.

162

Les instructions de contrôle

Tests sur les chaines de caractères

Les chaines doivent être entourées par des guillemets.

"-n "chaîne"" : vrai si la chaîne n'est pas vide.

"-z "chaîne"" : vrai si la chaîne est vide.

""chaine1" = "chaine2"" : vrai si les deux chaînes sont identiques.

""chaine1" != "chaine2"" : vrai si les deux chaînes sont différentes.

Comparaisons alphanumériques

"$str1" > "chaine1"

"$str1" < "chaine1"

"$str1" >= "chaine1"

"$str1" <= "chaine1"

163

Les instructions de contrôle

Les combinaisons de tests

Les combinaisons de tests sont utilisées quand on doit faire plusieurs tests simultanément, c'est à dire, quand on doit répondre à plusieurs conditions.

On utilise les opérateurs && et || comme dans les commandes composées.

L'opérateur ! sert à inverser la condition

164

Les instructions de contrôle

Exemple:

```
#!/bin/bash
echo -n "Entrez un nombre entier : " ; read nombre
if [ $nombre -lt 0 ]; then
  echo "$nombre est negatif";
elif [ $nombre -gt 0 ]; then
  echo "$nombre est positif";
else
  echo "$nombre est egal a zero";
fi
```


165

Les instructions de contrôle

Exemple: traduire ce script csh en script bash

```
#!/bin/csh
#Liste d'un fichier s'il existe

if  ($#argv == 1) then

  if  (-e  $1) then

    if  (-r  $1)   then

     if  (-d  $1)  then

        echo   Il s\'agit d\'un répertoire

          ls $1

     else  cat $1

     endif

   else echo Lecture impossible

  endif

  else echo $1 n\'existe pas

endif
```


166

Les instructions de contrôle

- **Itération**
  - Boucle for: for in do … done

Syntaxe:

for *variable* [in *liste d'arguments*]
do
liste de commandes
done

- **Exemple1 :**

```
#!/bin/bash
for mois in janvier fevrier mars avril; do
echo $mois;
done
```

Les instructions de contrôle

- **Itération**
  - Boucle for: for in do … done
  - **Exemple2 :**
  - #!/bin/bash
    if ! [$# -eq 2]; then
      echo "Le nombre de parametres doit etre de 2 (ancienne-extension nouvelle-extention) !"
    else
      ext1=$1
      ext2=$2
      for file in `ls *.$ext1`; do
        mv $file ${file%.$ext1}.$ext2;
      done
    fi

Les instructions de contrôle

- **Itération**
  - Boucle while: while do … done

Syntaxe:

while *commande*
do
liste de commandes
done

- **Exemple2 :**

```
#!/bin/bash
i=1 # on initialise le compteur
while [ $i -le 10 ]; do
  echo $i
  let $[ i+=1 ] # incremente i de 1 a chaque boucle
done
```

Les instructions de contrôle

- **Itération**
  - Boucle until: until do … done

Syntaxe:

until *commande*
do
liste de commandes
done

Les instructions de contrôle

- **Instructions de rupture de boucle**
  - continuation de boucle: **continue [n]**
    - revenir au début de la boucle for, while, until
    - n = niveau de boucle à partir de la boucle la plus externe
    - for var1 in ${liste1} do for var2 in ${liste2} do if cmd2 ; then continue 2 fi done done
  - Sortie de boucle : **break [n]**
    - sortir d'une ou n boucles for, while, until
    - n = niveau de boucle à partir de la boucle la plus externe
  - Sortie de programme: **exit n**
    - sortir d'un script-shell avec un code de retour (n)

Applications

- Écrire un Shell script UNIX ‘datef’, qui donne la date en français
- Écrire un Shell script UNIX ‘ou_suis_je’, qui vous indique si vous vous trouvez dans votre répertoire de travail par défaut ou pas
- Écrire un Shell script UNIX ‘indice ’, qui affiche l’indice d’une variable dans un tableau saisi en argument
- ftp

Les instructions de contrôle

- **Commandes de test**

  test sur les fichiers

  test -f fichier, test -d fichier, test -s fichier
  test -r fichier, test -w fichier, test -x fichier

  Exemple : if [! -d repert]; then mkdir repert ; fi

Applications

- Écrire un Shell script UNIX ‘modfi’, qui prend un nom de fichier en entrée (au clavier), et répond s’il peut être lu, écrit, ou exécuté (envisager les messages d’erreur)

- Écrire un Shell script UNIX ‘rep’, qui affiche le nombre et la liste des répertoires qui se trouve dans le répertoire courant

Applications (suite)

- Écrire un Shell script UNIX ‘taille’, qui affiche la liste des répertoires contenus dans un répertoire saisi en paramètre, ainsi que leur taille en ko :
  - en changeant de répertoire
  - sans changer de répertoire
- La commande ‘arbre’

Alias, fonction et arithmétique

- **Alias**
  - Création d'un alias:
    - alias dir='ls –lba'
  - Utilisation d'un alias
    - dir
  - Liste des alias
    - alias
  - Suppression d'un alias
    - unalias dir

Alias, fonction et arithmétique

- **Fonction**
  - Syntaxe: [function] nom() {……}
    - appel de fonction: nom [arg …]
    - pas de déclaration d'arguments entre parenthèses ()
    - les fonctions doivent être définies avant leur utilisation
  - Exemple:

```
#!/bin/sh
usage() { echo "usage: $1 $2" }
main() {
if [ $# = 0 ]; then usage `basename $0` "fichier"; exit 1 fi
    }
```

Alias, fonction et arithmétique

- **Arithmétique**
  - Commande Unix **expr**
    - **expr** *val1 oper val2*
    - opérations : **+**, **-**, *****, **/**, **%**
    - utilisation : substitution de commande en sh
    - peu performant : nécessite un fork()-exec()
  - Commande Shell (ksh, bash, zsh)
    - **$[expression]**
    - interne au shell -> performant

Alias, fonction et arithmétique

- **Arithmétique**
  - Exemple:

```
#!/bin/sh
while true;
do compteur=`expr $compteur + 1`
done

#!/usr/local/bin/zsh
while true;
do compteur=$[$compteur + 1]
done
```

Alias, fonction et arithmétique

- **Expression**

()	parenthèse pour forcer une évaluation			
+	addition		-	soustraction
/	division		*	multiplication
>	supérieur	<	inférieur	
>=	supérieur	<=	inférieur	
==	égalité		!=	inégalité
~	complément à 1		!	complément logique
%	modulo		^	disjonction exclusive bit à bit
>>	décalage à droite		<<	décalage à gauche
&	conjonction bit à bit	|	disjonction bit à bit	
&&	conjonction		||	disjonction

La gestion de fichiers

- La redirection >>
- La redirection <<
- Définition du séparateur de champs: IFS
  - Syntaxe:

IFS=valeur

  - Exemple:

IFS=,

La gestion de fichiers

- Rediriger les E/S de tout un script:
  - Syntaxe:
    - exec > fichier
    - exec < fichier
  - Exemple:

```
$ more programme
#!/bin/sh
exec < f1
exec > f2
read A; echo "1:$A"
read A; echo "2:$A"
```

La gestion de fichiers

- Lecture d'un fichier dans une boucle:
  - Syntaxe:
    - while read var [...] ; do ; ... ; done < fichier
  - Exemple:

```
$ more programme
#!/bin/sh
while read ligne
do
echo ">$ligne"
done < fichier
```

Série d'exercices n°6

1- Ecrire un script bash qui permet d'afficher la somme de quelques éléments donnés comme paramètres de la ligne commande.

2- Ecrire un script qui permet de comparer deux entiers (utilisez la fonction read et la ligne commande pour lire les deux entiers).

3- Ecrire un script bash qui permet de lire un nom et de tester :
s'il s'agit bien d'un fichier du répertoire courant
d'afficher son contenu s'il représente un fichier ordinaire
d'afficher son contenu s'il s'agit d'un répertoire

4- Ecrire un script bash 'modif' qui prend un nom de fichier entrée (au clavier), et répond s'il peut être lu, écrit, ou exécuté (envisager les messages d'erreur).

Série d'exercices n°6 (suite)

5- Ecrire un script 'rep', qui affiche le nombre et la liste des répertoires qui se trouve dans le répertoire courant.

6- Ecrire un shell script UNIX 'taille', qui affiche la liste des répertoires contenus dans un répertoire saisi en paramètre, ainsi que leur taille en ko :
en changeant de répertoire
sans changer de répertoire

7- Ecrire un script 'renomme.bash', qui permet de renommer les fichiers d'extension .txt en .tmp de votre répertoire de connexion en testant si le nombre de paramètres est égale à 2.

Généralités

- sh ("Bourne shell", Steve Bourne, AT&T)
  - shell historique (/usr/old/bin/sh, ou bsh)
- csh ("C-shell", Bill Joy, UCB)
  - shell BSD (disponible sur tous les Unix)
- ksh ("Korn shell", David Korn, AT&T)
  - Complète le shell Bourne et intègre avantages du C-shell
- sh (shell normalisé POSIX P1003.2 et ISO 9945.2)
  - Dérivé du shell Bourne et intègre fonctionnalités ksh
- bash ("Bourne again shell" de Brian Fox & C. Ramey)
  - shell libre de droit, compatible shell POSIX (par défaut Linux)
- zsh ("Zero shell")
  - ksh du domaine public (installé sur toutes les machines)
- rsh ("Bourne shell réduit", cd, <, > inopérants)
- tcsh ("Toronto C-shell")

Généralités

- Comparaison des Shells

Shell	program.	histo	aliases	gestion tableaux	chaînes	var. num	arith. C
bsh	std		non	non	-	non	non
sh-POSIX	norme	oui	oui	oui	-	oui	
csh	non std		oui	oui	non	oui	oui
tcsh	non std		oui	oui	non	oui	oui
ksh	std		oui	oui	oui	oui	oui
bash	norme		oui	oui	oui	-	oui
zsh	norme		oui	oui	oui	-	oui

Généralités

- Comparaison des Shells

Shell	Programmation
sh	sûre (standard)
csh	à déconseiller
tcsh	à déconseiller
ksh	normalisée
bash	normalisée
zsh	normalisée + extensions

Généralités

- **Algorithme du shell**
  - Ouverture initiale de 3 fichiers standard
    - entrée standard (stdin, fd=0), par défaut le clavier
    - sortie standard (stdout, fd=1), par défaut l'écran
    - sortie erreur standard (stderr, fd=2), par défaut l'écran
  - Le shell lit sur son entrée standard
    - lecture de la commande jusqu'à la fin de ligne (\n) et interprétation des métacaractères et opérateurs
    - lecture des lignes de commandes jusqu'à la fin de fichier (^D)

Généralités

- **Algorithme du shell**
  - Lancement d'une commande
    - recherche du nom de la commande à l'aide du PATH (path)
    - en premier plan :

      fork() exec(commande) wait() prompt
    - en arrière plan :

      fork() exec(commande) prompt

Généralités

- **Caractères de contrôle au clavier**
  - Caractères de contrôle:
    - <cr> fin de ligne (retour chariot) mappé en <cr>
    - <lf> fin de ligne (nouvelle ligne) mappé en <cr><lf>
    - <tab> tabulation
    - <bs> ^H, backspace, effacement du caractère précédent
    - <del> ^?, souvent identique à <bs>
    - <^C> interruption d'un processus attaché au terminal
    - <^\> arrêt d'un processus avec vidage mémoire (core)

Généralités

- **Caractères de contrôle au clavier**
  - Caractères de contrôle (suite):
    - <^Z> suspension d'un processus en premier plan
    - <^U> effacement de la ligne complète
    - <^W> effacement du mot qui précède
    - <^D> fin de fichier => si le shell lit, fin d'un shell
    - <^S> suspension de l'affichage écran (Xoff)
    - <^Q> reprise de l'affichage écran (Xon)
  - Gestion des caractères de contrôle par stty
    - stty erase ^H kill ^U intr ^C susp ^Z quit ^\ eof ^D

Généralités

- **Format d'une commande:**
  - Invite (le prompt) principale
    - $ en mode utilisateur
    - # en mode super-utilisateur
    - variable PS1
  - Invite secondaire
    - > en mode interactif
    - variable PS2

Généralités

- **Format d'une commande:**
  - Commande simple
    - une commande simple est une séquence de mots séparés par un séparateur blanc et de redirections. Le premier mot désigne le nom de la commande à exécuter, les mots suivants sont passés en arguments à la commande.
    - la valeur retournée d'une commande est celle de son exit.
  - Chemin de recherche du nom d'une commande : PATH ou path (csh)
    - PATH=/usr/ucb:/bin:/usr/bin:/usr/local/bin:~/bin:.

Généralités

- **Enchaînement de commandes:**
  - Commande simple (en premier plan)

$ commande [arg …]

  - Commande simple (en arrière plan)

$ commande [arg …] &

  - Pipeline (tube de communication)

$ commande1 [arg …] | commande2 [arg …] | …

  - Séquencement de commandes (en premier plan)

$ commande1 [arg …]; commande2 [arg …]; cmd3 …

Généralités

- **Enchaînement de commandes:**
  - Exécution par un nouveau shell ()

$ sh commande [arg …]

$ (commande [arg …])

$ (commande [arg …]) &

  - Regroupement de commandes

$ (com1 [arg …]; com2 [arg …]; com3 …)

$ (com1 [arg …]; com2 [arg …]; com3 …)&

Le profil d'utilisateur

- **Fichiers d'initialisation:**

  /etc/profile .profile

  .login .cshrc .logout

  .bash_profile .bashrc .bash_alias

  .inputrc

  .zshrc

  .tcshrc

Le profil d'utilisateur

- **/etc/profile (sous System V):**
  - Initialisation commune à tous les login-shells
  - point de passage obligé pour tous les login
  - fourni par constructeur/adapté par administrateur
  - non modifiable par l'utilisateur
  - Exemple :

```
umask 027
USER=`logname`                          HOSTNAME=`uname -n`
PATH=/bin:/usr/bin:/usr/X11/bin                 OS=`uname -s`
MANPATH=/usr/local/man:/usr/man         REL=`uname -r`
MAIL=/usr/spool/mail/$USER              OSREL=${OS}${REL}
export USER PATH MANPATH MAIL HOSTNAME OS REL OSREL
```

Le profil d'utilisateur

- **.profile:**
  - Initialisation propre à l'utilisateur
  - situé dans le HOME de l'utilisateur : ~/.profile
  - redéfinition des variables proposées par /etc/profile
  - définition de nouvelles variables (var=valeur)
  - détection si le shell est interactif ou non
  - exécution ou non de scripts d'initialisation
  - configuration des paramètres du terminal (stty)
  - peut-être "ressourcé" (. ou source):
    - . profile (sh, ksh, bash, zsh)
    - source .profile (bash)

Le profil d'utilisateur

- **.zshrc, .bashrc (en zsh ou bash):**
  - Spécificité des shells zsh et bash
  - Initialisation propre à l'utilisateur
  - situé dans le HOME de l'utilisateur : ~/.zshrc ~/.bashrc
  - redéfinition du prompt PS1
  - définition des aliases
  - définition de fonctions
  - peut-être "ressourcé": ~/.bashrc ou source ~/.bashrc . ~/.zshrc ou source ~/.zshrc

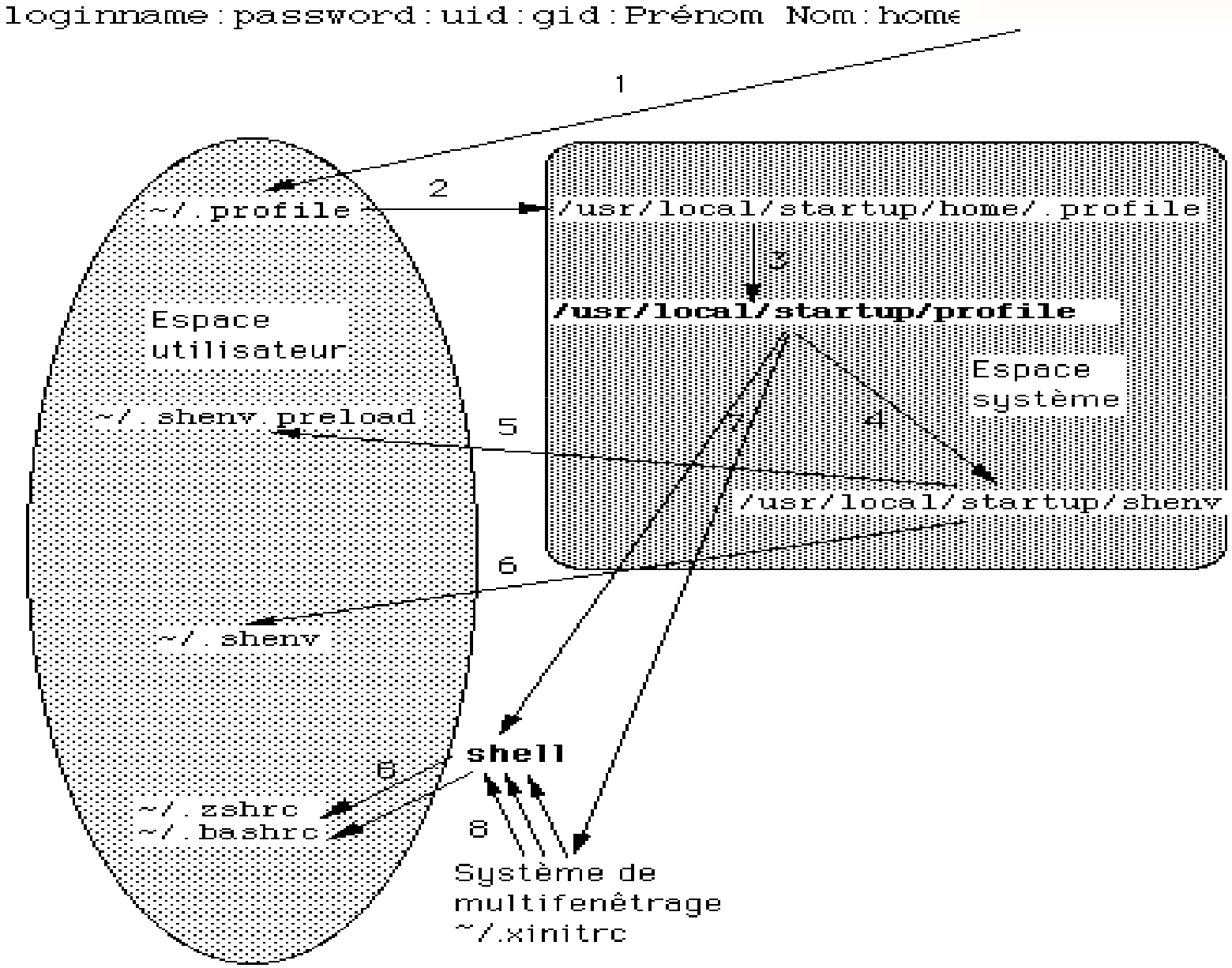
Exemple:

alias ls='ls -F' alias ll='ls -laF'

Le profil d'utilisateur

- **.login, .cshrc, .tcshrc, .logout (en csh ou tcsh):**
  - .login : Fichier d'initialisation d'un login C-shell
    - interprété à chaque login
    - définition de nouvelles variables
    - peut-être "ressourcé" par source .login
  - .cshrc/.tcshrc : Fichier d'initialisation d'un Csh
    - réinterprété à chaque invocation du csh/tcsh
    - définition des aliases (shells interactifs)
    - peut-être "ressourcé" par source .cshrc/.tcshrc
  - .logout : Fichier de clôture d'une session C-shell
    - effacement des fichiers temporaires, clear écran
  - Exemple: setenv EDITOR emacs

Le profil d'utilisateur



- **Application:**
  - L 'éditeur 'vi'
  - créer un fichier '.logout' qui efface l 'écran et bloque l 'interpréteur de commandes 5 secondes avant de fermer la session
  - exécution: la commande 'source'
  - Alias: ll, la

Le shell en interactif et les scripts shell

- Le shell en intéractif:
  - Les jokers: ? * []
  - Les caractères d'échappement: \ ' "
    - Simples quotes '…' : les caractères inclus entre 2 simples quotes ne sont pas évalués, ils conservent leur valeur littérale.
    - Doubles quotes "…" : les caractères inclus entre 2 doubles quotes conservent leur valeur littérale à l'exception de $ ` et \.
  - Les redirections: > >> < << 2> 2>>
  - Les tubes |
  - Le remplacement de commandes:

`cmd` ou $(cmd)

Le shell en interactif et les scripts shell

- Les shell-scripts:
  - Les commentaires:
    - #!/bin/sh
    - # @(#) résultat cmd what
  - Exécution d'un script
    - Shell fils:
      - sh shell-script
      - sh < shell-script
      - ./shell-script
    - Shell courant:
      - . shell-script

Le shell en interactif et les scripts shell

- Les shell-scripts:
  - La mise au point:
    - Activation/Désactivation du mode trace ds le script:

```
#!/bin/sh
set –x
# l'option x (eXtra output: affiche les étapes de
#                    substitution signalées par le signe +)
set –
# revient au mode normal
```

    - Exécution d'un script, affiche la trace d'une cmde:

```
sh –xv shell-script
```

Le shell en interactif et les scripts shell

- Les shell-scripts:
  - La mise au point:
    - L'option x:
      - affiche les étapes de substitution, signalées par la ligne +
    - L'option v (verbose):
      - affiche la ligne telle qu'elle est écrite dans le script, avant que le shell ne l'interprète
    - L'option n:
      - permet de seulement vérifier la syntaxe
    - L'option e:
      - provoque la fin d'exécution d'un script si une commande se termine avec un code de retour différent de zero

Le shell en interactif et les scripts shell

- Les shell-scripts:
  - Les options du shell:
    - Gestion des options:
      - set –o drapeau [shell-script]# lève un drapeau
      - set +o drapeau [shell-script] # abaisse un drapeau
      - set –o # affiche la liste des drapeaux
    - Principales options:
      - ignoreeof :oblige à utiliser exit et interdit l'usage de ^D
      - noclobber :interdit à une redirection d'effacer un fichier
      - vi : active l'édition de commandes en mode vi
      - noglob : n'interprète pas les jockers * et ?

Les variables

- **Les types de variables :**
  - Variables prédéfinies:
    - lors de la connexion
  - Variables positionnelles:
    - paramètres d'un script
  - Variables spéciales du shell:
    - maintenues par le shell
  - Variables créées par l'utilisateur

Les variables

- **Syntaxe:**
  - Déclaration:
    - variable=valeur
  - Utilisation:
    - variable2=$variable
    - variable2=${variable}
  - Liste des variables:
    - set
  - Suppression des variables:
    - unset nom

Les variables

- **L'environnement:**
  - Ensemble des variables dont un shell fournit une copie à la descendance
  - La commande export :
    - Syntaxe:

      export [variable[=valeur]]
    - Exemple:

Les variables

- **Les variables d'environnement (créées lors de la connexion):**
  - HOME chemin d'accès au répertoire initial de l'utilisateur
  - PATH suite de chemins d'accès aux répertoires des exéc.
  - PS1 invite principale du shell en mode interpréteur
  - PS2 invite secondaire du shell en mode programmation
  - IFS séparateurs de champ des arguments
  - MAIL chemin d'accès à la boîte aux lettres utilisateur
  - MAILCHECK intervalle en sec au bout duquel le mail est contrôlé
  - CDPATH liste de chemins d'accès pour la commande cd
  - ENV nom du fichier des variables d'environnement
  - TERM nom du type de terminal

Les variables

- **Les variables positionnelles:**
  - $0 nom du script (pathname)
  - $1,…, $9, $10,… arguments (du 1er au 9ème, 10ème,…)
  - $# nombre d'arguments
  - $* liste de tous les arguments
  - $@ liste de tous les arguments
  - $? code retourné par la dernière commande
  - $$ numéro de processus de ce shell
  - $! numéro du dermier processus en arrière plan
  - $- drapeaux fournis au shell par set (options du shell)

Les variables

- **Les variables positionnelles:**
  - La commande shift: décale les paramètres de n positions vers la gauche (par défaut 1)

  Syntaxe: shift [n]
  - La commande set: remplace les paramètres du script par les arguments de la commande, ainsi que les variables #, * et @

  Syntaxe: set argument1 ...
  - Exemple:

Les variables

- **L'instruction read:**

Syntaxe: read variable …

- **L'instruction readonly:**

Crée une variable qu'on ne peut plus supprimer ni modifier

Syntaxe: readonly variable=valeur

Les variables

- **Le remplacement de variables:**

variable=$var : si var ∃ ou != 0 alors var sinon rien

variable=${var} : si var ∃ ou != 0 alors var sinon pas d'affectation

variable=${var:-val} : si var ∃ ou != 0 alors var sinon val

variable=${var:=val} : si var ∃ ou != 0 alors var sinon val et var ⇐ val

variable=${var:?mess} : si var ∃ ou != 0 alors var sinon affiche 'mess'

variable=${var:+val} : si var ∃ ou != 0 alors val sinon aucune substitut°

Les variables

- **Application:**
  - Écrire un Shell-script qui affiche la liste de ses arguments
  - Écrire un Shell-script qui affiche la valeur de son 1er argument

Les chaînes de caractères

- **La manipulation de chaînes:**
  - Calcul de la longueur d'une chaîne
    - expr chaîne:".*"
    - Exemple:

```
$ expr "bonjour":".*"
7
```

  - Extraction d'une sous-chaîne
    - \(..\)
    - Exemple:

```
$ Temps="Il fait beau«
$ expr "$Temps":"\(..\)"
Il
$ expr "$Temps":".*\(....\)"
beau
```

```
$ chemin=/usr/bin/ls
$ expr "$chemin":".*/\(.*\)"
ls
$ expr "$chemin":"\(.*\)/"
/usr/bin
```

  - Les commandes basename $chemin et dirname $chemin

Les chaînes de caractères

- **Le remplacement de variables:**

variable=${#var} : longueur en octet de la chaîne var

variable=${var#str} : supprime en début de var la + petite des 2 chaînes

variable=${var##str} : supprime en début de var la + grande des 2 chaînes

variable=${var%str} : supprime en fin de var la + petite des 2 chaînes

variable=${var%%str} : supprime en fin de var la + grande des 2 chaînes

Application

- La commande ‘pers_env’
  - Écrire un shell-script ‘pers_env’ qui personnalise la commande ‘setenv’.
  - Lancer l’exécution de ‘pers_env’ comme une commande (chmod u+x pers_env)et(!#/bin/csh)
  - Exemple:

% pers_env

Je suis l ’utilisateur ali

Je suis dans le répertoire /etc

Mon répertoire de travail par défaut est: /home/ali

Les instructions de controle

- **Familles de commandes avancées:**
  - Sélection
  - Aiguillage
  - Itération
  - Rupture de boucle
  - Commande de test

Les instructions de contrôle

- **Sélection**
  - Syntaxe :
    - Condition simple: **if then ... fi**

if *commande* ; **then**
liste de commandes
fi

    - Condition avec alternative: **if then ... else ... Fi**

if *commande* ; **then**
liste de commandes
else
liste de commandes
fi

Les instructions de contrôle

- **Sélection**
  - Syntaxe :
    - Condition multiple: **if then ... elif then ... fi**

if *commande* ; **then**
liste de commandes
elif *commande* ; **then**
liste de commandes
else
liste de commandes
fi

Les instructions de contrôle

- **Aiguillage**
  - Aiguillage case: **case in ... Esac**

case *$variable* **in**
*motif1***)** liste de commandes **;;**
*motif2***)** liste de commandes **;;**
***)** liste de commandes **;;**
esac

  - Exemple :

```
case $# in
        1) arg1=$1;;
        2) arg1=$1; arg2=$2 ;;
        0) echo "usage: $0 arg1 [ arg2 ] exit 1;;
esac
```

Les instructions de contrôle

- **Itération**
  - Boucle for: for in do … done

Syntaxe:

for *variable* [in *liste d'arguments*]
do
liste de commandes
done

  - Boucle while: while do … done

Syntaxe:

while *commande*
do
liste de commandes
done

Les instructions de contrôle

- **Itération**
  - Boucle until: until do … done

Syntaxe:

until *commande*
do
liste de commandes
done

Les instructions de contrôle

- **Instructions de rupture de boucle**
  - continuation de boucle: **continue [n]**
    - revenir au début de la boucle for, while, until
    - n = niveau de boucle à partir de la boucle la plus externe
    - for var1 in ${liste1} do for var2 in ${liste2} do if cmd2 ; then continue 2 fi done done
  - Sortie de boucle : **break [n]**
    - sortir d'une ou n boucles for, while, until
    - n = niveau de boucle à partir de la boucle la plus externe
  - Sortie de programme: **exit n**
    - sortir d'un script-shell avec un code de retour (n)

Applications

- Écrire un Shell script UNIX ‘datef’, qui donne la date en français
- Écrire un Shell script UNIX ‘ou_suis_je’, qui vous indique si vous vous trouvez dans votre répertoire de travail par défaut ou pas
- Écrire un Shell script UNIX ‘indice ’, qui affiche l’indice d’une variable dans un tableau saisi en argument
- ftp

Les instructions de contrôle

- **Commandes de test**

  test sur les fichiers

  test -f fichier, test -d fichier, test -s fichier

  test -r fichier, test -w fichier, test -x fichier

  Exemple : if [! -d repert]; then mkdir repert ; fi

  Comparaisons alphanumériques

  - test "$str1" = "chaine1"
  - test "$str1" != "chaine1"
  - test "$str1" > "chaine1"
  - test "$str1" < "chaine1"
  - test "$str1" >= "chaine1"
  - test "$str1" <= "chaine1"

Les instructions de contrôle

- **Commandes de test**
  - Comparaisons algébriques
    - -eq : Exemple: $? eq 0
    - -ne
    - -gt
    - -lt
    - -ge
    - -le
  - Comparaisons booléennes
    - -z : Exemple: -z $chaine chaine nulle
    - -n : Exemple: -n $chaine chaine non nulle

    Autres opérateurs
    - !
    - -a : Syntaxe: Expr.Logique1 –a Expr.Logique2 (and)
    - -o : Syntaxe: Expr.Logique1 –o Expr.Logique2 (or)

Applications

- Écrire un Shell script UNIX ‘modfi’, qui prend un nom de fichier en entrée (au clavier), et répond s’il peut être lu, écrit, ou exécuté (envisager les messages d’erreur)
- Écrire un Shell script UNIX ‘rep’, qui affiche le nombre et la liste des répertoires qui se trouve dans le répertoire courant

Applications (suite)

- Écrire un Shell script UNIX ‘taille’, qui affiche la liste des répertoires contenus dans un répertoire saisi en paramètre, ainsi que leur taille en ko :
  - en changeant de répertoire
  - sans changer de répertoire
- La commande ‘arbre’

Alias, fonction et arithmetique

- **Alias**
  - Création d'un alias:
    - alias dir='ls –lba'
  - Utilisation d'un alias
    - dir
  - Liste des alias
    - alias
  - Suppression d'un alias
    - unalias dir

Alias, fonction et arithmétique

- **Fonction**
  - Syntaxe: [function] nom() {……}
    - appel de fonction: nom [arg …]
    - pas de déclaration d'arguments entre parenthèses ()
    - les fonctions doivent être définies avant leur utilisation
  - Exemple:

```
#!/bin/sh
usage() { echo "usage: $1 $2" }
main() {
if [ $# = 0 ]; then usage `basename $0` "fichier"; exit 1 fi }
main $*
```

Alias, fonction et arithmétique

- **Arithmétique**
  - Commande Unix **expr**
    - **expr** *val1 oper val2*
    - opérations : **+**, **-**, *****, **/**, **%**
    - utilisation : substitution de commande en sh
    - peu performant : nécessite un fork()-exec()
  - Commande Shell (ksh, bash, zsh)
    - **$[expression]**
    - interne au shell -> performant

Alias, fonction et arithmétique

- **Arithmétique**
  - Exemple:

```
#!/bin/sh
while true;
do compteur=`expr $compteur + 1`
done

#!/usr/local/bin/zsh
while true;
do compteur=$[$compteur + 1]
done
```

Alias, fonction et arithmétique

- **Expression**

()	parenthèse pour forcer une évaluation			
+	addition		-	soustraction
/	division		*	multiplication
>	supérieur	<	inférieur	
>=	supérieur	<=	inférieur	
==	égalité		!=	inégalité
~	complément à 1		!	complément logique
%	modulo		^	disjonction exclusive bit à bit
>>	décalage à droite		<<	décalage à gauche
&	conjonction bit à bit	|	disjonction bit à bit	
&&	conjonction		||	disjonction

La gestion de fichiers

- La redirection >>
- La redirection <<
- Définition du séparateur de champs: IFS
  - Syntaxe:

IFS=valeur

  - Exemple:

IFS=,

La gestion de fichiers

- Rediriger les E/S de tout un script:
  - Syntaxe:
    - exec > fichier
    - exec < fichier
  - Exemple:

```
$ more programme
#!/bin/sh
exec < f1
exec > f2
read A; echo "1:$A"
read A; echo "2:$A"
```

La gestion de fichiers

- Lecture d'un fichier dans une boucle:
  - Syntaxe:
    - while read var [...] ; do ; ... ; done < fichier
  - Exemple:

```
$ more programme
#!/bin/sh
while read ligne
do
echo ">$ligne"
done < fichier
```

La programmation mult shell

- Panorama des commandes:
  - cmd&
  - ps
  - kill
  - wait
  - nice
  - nohup
  - at
  - crontab

La programmation multitâche en shell

- Regroupement de commandes:
  - Exécutées par le shell en cours:
    - {cmd; …; cmd;}
  - Exécutées par un shell fils:
    - (cmd; …; cmd)
  - Les variables $ et !:
    - Exemple:

La programmation multitâche en shell

- La gestion des signaux: trap et kill
- Les groupes de processus
- L'échange de données par tube nommé
- Les verrous

Quelques cmdes utiles

- eval
- select (ksh)
- getopts
- what
- args
- tput

La programmation multitâche en shell: Déroutements

- Séquence d'exception:
  - Syntaxe: trap '*liste de commandes*' *liste de signaux*
  - Exemple : trap 'rm /tmp/appl.$$; exit 1' 2
- Ignorance des signaux :
  - Syntaxe: trap '*/ liste de signaux*
  - Exemple : trap '' 1 2 3 15
- Réinitialisation
  - Syntaxe: trap *liste de signaux*
  - Exemple : trap 2 3
- Liste des signaux
  - Syntaxe: trap
  - Exemple : trap

Plan

- Définition
- Filtre comm
- Filtre diff
- Filtre grep
- Filtre programmable awk

Définition

- Commandes qui :
  - acceptent des données de l'entrée standard
  - effectuent transformations sur ces données
  - dirigent ces données vers la sortie standard
  - affichent leurs messages d'erreur vers la sortie erreur standard

Filtre « comm »

- Définition:
  - Compare le contenu de 2 ensembles de données
  - L'un des 2 ensembles peut provenir de l'entrée std
- Syntaxe:

comm [options] fichier1 fichier2
-1 N'affiche pas la 1ère colonne (lignes présentes uniquement dans le fichier1)
-2 N'affiche pas la 2ème colonne (lignes présentes uniquement dans fichier2)
-3 N'affiche pas la 3ème colonne (lignes présentes dans les deux fichiers)

- Statut de sortie:

0 pour indiquer qu'il n'y a pas eu d'erreur;

>0 pour indiquer qu'une erreur est rencontrée.

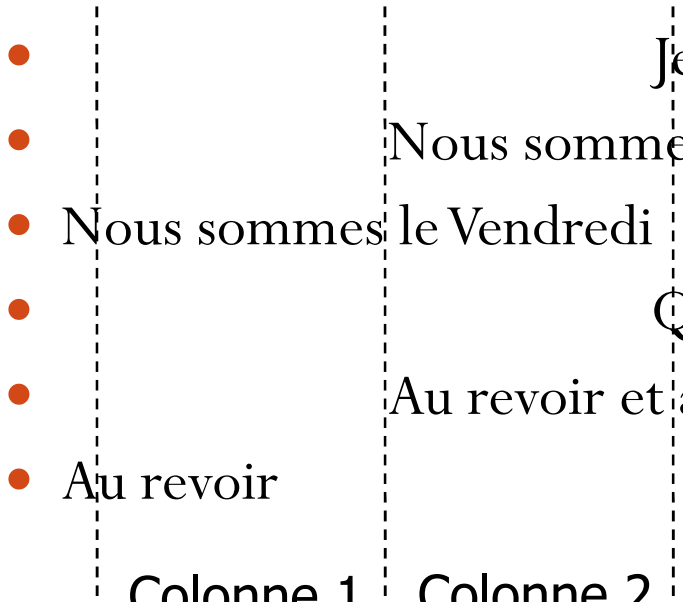
Filtre « comm »

- Remarque:
  - Si le nom de l'un des fichiers est le caractère - alors son contenu est lu de l 'entrée standard.
  - Exemple:

Fichier 1	Fichier 2
Je suis venu vous voir	Je suis venu vous voir
Nous sommes le Vendredi	Nous sommes le Vendredi.
Quelle heure est-il?	Quelle heure est-il?
Au revoir	Au revoir et à bientôt

Filtre « comm »

- $ comm fichier1 fichier2



Filtre « diff »

- Définition:
  - donne les différences entre 2 ensembles de données
  - accepte des noms de répertoire
- Syntaxe:

diff [options] fichier1 fichier2
diff [options] fichier1 répertoire
diff [options] répertoire fichier1
diff [options] répertoire1 répertoire2
-b Ignorer les blancs (Espace et Tab) à la fin d'une ligne
-c[n] Afficher le contexte où les lignes différentes ont été trouvées

- Statut de sortie:

0 Pas de différence rencontrée
1 Des différences ont été relevées
>1 Une erreur est rencontrée

Filtre « grep »

- Définition:
  - Trouve des caractères dans un ensemble de données
  - Chaîne de caractères recherchée décrite par regex
- Synopsis:

egrep [options] [-e expr] [expression] [-f fich] [fichiers …]

-b Afficher au début de chaque ligne trouvée, n° de bloc sur disque

-c Afficher uniquement le nombre de lignes contenant la chaîne

-h N'afficher pas les noms de fichier

-i Ne pas faire de distinction entre les majuscules et les minuscules

-s Aucune sortie sauf les messages d'erreur

-e Nécessaire pour les expressions régulières commençant par un tiret

-f fich Expression régulière est contenue dans le fichier fich

Filtre « grep »

- Statut de sortie:

0 la chaîne est trouvée

1 la chaîne n'est pas trouvée

2 erreur(s) rencontrée(s)

- Remarque:
  - Par défaut, les lignes trouvées sont envoyées à la sortie standard.

Filtre « grep »

- Exemple:

```
centi 4>   cat passwd |  grep ':Pierre.*'
parp2808:cRuT6m7jPRJWM:30502:11:Pierre Parent:/export/home/exa/email/parp2808:/
bin/csh
doup2609:S3iOoM68.AuVQ:30251:11:Pierre Doucet:/export/home/exa/email/doup2609:/
bin/csh
robcad22:Z.VGpH9oanp3M:11423:114:Pierre St-Denis:/usr/people/exa/robcad/robcad22
:/bin/csh
gaup2806:8fZaJLAA4yb0Y:30848:11:Pierre-Luc Gauthier:/export/home/exa/email/gaup
2806:/bin/csh
dupp0911:LBXJgdV.Z181g:30267:112:Pierre Duplessis:/export/home/exa/email/dupp091
1:/bin/csh
gerp1209:JLiqJTddvRdJY:30307:112:Pierre Hugues Gervaud:/export/home/exa/email/ge
rp1209:/bin/csh
gefp0904:zHizZC2Vw0j1Q:30304:112:Pierre Olivier Geffroy:/export/home/exa/email/g
efp0904:/bin/csh
boup0806:aX.tOb2BGmZcM:30804:11:Pierre-Etienne Bouchard:/export/home/exa/email/
boup0806:/bin/csh
grop2001:jzWi0Wxd.2iPQ:30966:112:Pierre Grondin:/export/home/exa/email/grop2001:
/bin/csh
duvp0101:4jttBQNX/rHcg:30271:11:Pierre-Yves Duval:/export/home/exa/email/duvp01
01:/bin/csh
habp0410:ytzP1NEqscex2:30857:112:Pierre-Luc Habel:/export/home/exa/email/habp041
0:/bin/csh
: : :
```

Filtre « grep »

- Exemple:

```
centi 5>   cat passwd |  grep ':Pierre .*'
parp2808:cRuT6m7jPRJWM:30502:112:Pierre Parent:/export/home/exa/email/parp2808:/
bin/csh
doup2609:S3iOoM68.AuVQ:30251:112:Pierre Doucet:/export/home/exa/email/doup2609:/
bin/csh
robcad22:Z.VGpH9oanp3M:11423:114:Pierre St-Denis:/usr/people/exa/robcad/robcad22
:/bin/csh
dupp0911:LBXJgdV.Z181g:30267:112:Pierre Duplessis:/export/home/exa/email/dupp091
1:/bin/csh

gerp1209:JLiqJTddvRdJY:30307:112:Pierre Hugues Gervaud:/export/home/exa/email/ge
rp1209:/bin/csh
gefp0904:zHizZC2Vw0j1Q:30304:112:Pierre Olivier Geffroy:/export/home/exa/email/g
efp0904:/bin/csh
grop2001:jzWi0Wxd.2iPQ:30966:112:Pierre Grondin:/export/home/exa/email/grop2001:
/bin/csh
stdp2705:srllvkqzhiJho:30582:112:Pierre St-Denis:/export/home/exa/email/stdp2705
:/bin/csh
berp0604:m0T.Bp4f9rQ3I:30138:112:Pierre Berube:/export/home/exa/email/berp0604:/
bin/csh
pinp2309:1kVuOqUAYsMmg:30519:112:Pierre Pinard:/export/home/exa/email/pinp2309:/
bin/csh
daup2612:6qqRrxM0kIbkA:30220:112:Pierre Noel Dautrey:/export/home/exa/email/daup
2612:/bin/csh
: : :
```

Filtre « grep »

- Exemple:

```
$ grep '[.,!]' *
comm1.dat :cher Monsieur Jean.
comm1.dat :Non, Pas le Dr. Watson de Windows!
comm2.dat :Non, Pas le Dr. Watson de Holms!
```

Trouver le caractère point ou le caractère virgule ou le caractère point exclamation dans tous les fichiers du répertoire courant.

- Remarque:
  - Le filtre grep est très utile dans la recherche des motifs dans un fichier ou un ensemble de fichiers.
  - La vitesse d'exécution de grep est très très grande!

Le filtre awk

- Définition
- Syntaxe
- Principe de fonctionnement
- Forme générale des programmes
- Application

Définition

- Filtre programmable utilisant les instructions légales selon le format 'awk'
- Ressources:
  - Le filtre/commande 'awk'
  - Le programme/script
  - Le(s) fichier(s) objet(s) du filtrage
- Concept de programmation « piloté par les données » (Data Driven)
- Utilisé pour:
  - Valider les concepts de programmation
  - Automatiser les tâches de gestion
  - Évaluer rapidement les algorithmes, etc…

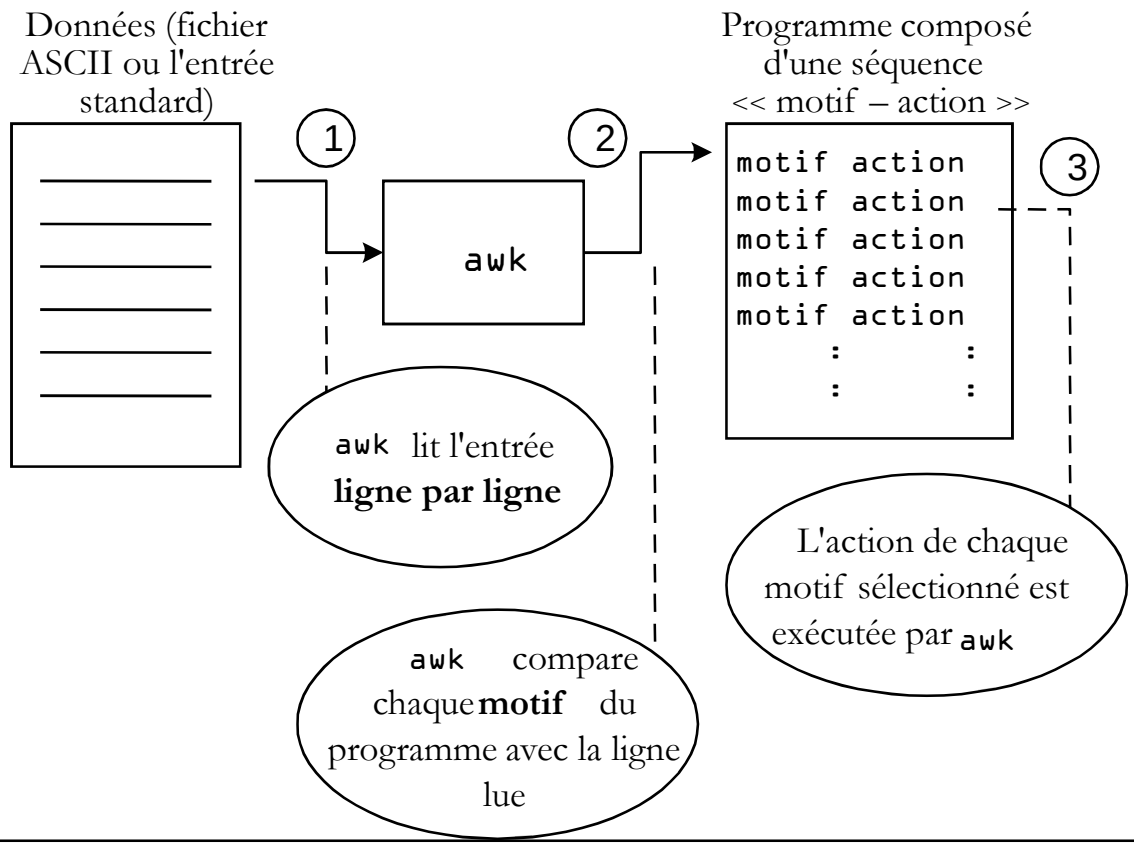
Définition

- La commande ‘awk’ permet de rechercher dans chacun des fichiers référencés les lignes satisfaisant 1 ou plusieurs motifs parmi ceux énumérés dans le programme awk

Syntaxe

- awk [-F<car>] [-f <refprog>] [‘<programme>’] [<fichier> …]
  - Option –F : définit un séparateur de champs dans les lignes différent du séparateur usuel (espace ou tabulation).
  - Fichiers traités: entrée standard ou paramètres selon l’ordre où ils déclarés.

Principe de fonctionnement



Forme des programmes

- Le programme awk est divisé en 3 sections :
  - Section initiale

syntaxe : BEGIN {actions avant}

  - Corps

syntaxe : motif {actions sur chaque ligne}

  - Section terminale

syntaxe : END {actions après}

- Mécanisme d'exécution de awk :
  - Exécuter les actions de la section initiale
  - Tant que (fin du dernier fichier non atteinte) faire

Lire l'enregistrement suivant

Pour (chaque motif vérifié par l'enregistrement)

exécuter actions correspondant à ce motif

Fin pour

  - Fin tant que
  - Exécuter les actions de la section terminale

Forme des programmes

- Actions et motifs par défaut :
  - Action par défaut :
    - Impression de l'enregistrement
  - Motif par défaut :
    - Tout enregistrement contient le motif par défaut

Forme des programmes

- Forme des motifs :
  - motif =

expression rationnelle

||

expression relationnelle

Forme des programmes

- Forme des motifs (suite):
  - Expression rationnelle (pour la ligne): /motif/
    - / : délimiteur d'expression
    - ^ : début de ligne
    - $: fin de ligne
    - - : dans un crochet, définit un intervalle
    - * : itérer un caractère
    - [] : définit un ensemble

Forme des programmes

- Forme des motifs (suite):
  - Expression relationnelle :
    - <expression> ~ <expression rationnelle> :

expression de gauche contient motif satisfaisant expression de droite

- exemple:

$1 ~ /^toto/

le premier champ est une chaîne de caractères

qui commence par 'toto'

Forme des programmes

- Forme des motifs (suite):
  - Expression relationnelle (suite):
  - <expression> !~ <expression rationnelle> :

expression de gauche ne contient pas de motif satisfaisant expression de droite

- exemple:

$1 !~ /^toto/

le premier champ est une chaîne de caractères

qui ne commence pas par 'toto'

Forme des programmes

- Forme des motifs (suite):
  - Expression relationnelle (suite):
  - <expression> <opérateur> <expression> :

avec <opérateur> : <, >, ≤, ≥, ==, !=

exemple:

$1 == ”toto ”

le premier champ est une chaîne de caractères

qui coïncide avec ‘toto’

Forme des programmes

- Les actions :
  - Variables utilisateurs :
    - i.e. langage C
  - Variables de champ :
    - $0, $1, …, $i

Forme des programmes

- Les actions (suite):
  - Variables prédéfinies :
    - FS : séparateur de champ traité
    - RS : séparateur d'enregistrement
    - OFS :séparateur de champ en sortie
    - ORS :séparateur d'enregistrement en sortie
    - NF : nombre de chps dans l'enregistrement courant
    - NR : n° de l'enregistrement en crs de traitement
    - FILENAME :nom du fichier en cours de traitement

Forme des programmes

- Les actions (suite):
  - Opérateurs :
    - arithmétiques : ++, --, +, -, *, /
    - relationnels : <, >, ≤, ≥, ==, !=, ~, !~
    - booléens : &&, ||
    - affectation : =, +=, -=, *=, /=
  - Tableau : ligne[NR]=$0

Forme des programmes

- Les actions (suite):
  - Instructions :
    - Impression :

      print [<expression>, <expression>, …]
    - Condition :

      if (<condition>)

      <instruction>

      else

      <instruction>

Forme des programmes

- Les actions (suite):
  - Instructions (suite):
    - Itération :

for (<expression>, <expression>,<expression>)
<instruction>
ou
for (<var> in <tableau>)
<instruction>
ou
while (<condition><)
<instruction>

Forme des programmes

- Les actions (suite):
  - Les fonctions prédéfinies
    - length(s)
    - exp(x)
    - log(x)
    - sqrt(x)
    - int(x)
    - substr(s,position,longueur)
    - index(s1,s2)