UNIVERSITÉ DE VERSAILLES ST-QUENTIN-EN-YVELINES

INSTITUT DES SCIENCES ET TECHNIQUES DES YVELINES

Introduction au Système UNIX

L.Yeh

September 3, 2018

Basé sur le cours de Van Dat Cung

Plan du Cours

SECTION: Le Système UNIX

Un Système d'Exploitation

- Programme qui permet aux utilisateurs d'interagir avec un ordinateur:
  - pour accéder à ses ressources (disques, ...);
  - pour lancer d'autres programmes.
    $\Rightarrow$ programme fondamental qui existe sur tout ordinateur:
  - MS-DOS, VMS, Unix, Multics, MS-Windows, Android, ...

Objectifs Dénéraux du Système UNIX

- UNIX = Noyau + Interpréteurs de commandes + Utilitaires
  - Noyau réalise les fonctions primitives
    - Exemple: Allouer de la mémoire à un processus (programme)
  - Interpréteur de commandes
    - Interactions avec l'utilisateur
  - Utilitaires
    - Ex: Outil d'impression (lp)
- Caractéristiques
  - d'usage général et interactif,
  - multi-tâches, multi-utilisateurs, en temps partagé

Le Noyau Unix

- Noyau Unix est basé sur deux concepts :
  - Un système de fichier hiérarchisé:
    - Différents types de fichiers pour réaliser les différentes taches: uniformité des E/S entre : périphériques, fichiers ou processus,
    - Organisation de la mémoire secondaire (disque)
  - Gestion de processus
    - Un processus = exécution d'un programme
    - la possibilité d'exécution de processus (exécution multi-tâches) avec contrôle synchrone ou asynchrone
- Fournit des services aux utilitaires conçus au-dessus
  - Exemple `man` (Manuel d'une commande).

Interpréteur de Commandes vs. Utilitaires

- Interpréteur de commandes
  - Interface utilisateur qui lit, puis interpréte et d'exécute les commandes demandées par l'utilisateur.

Exemples:

```
$ date
dimanche 4 septembre 2016, 12:55:39 (UTC+0200)
```

  - Appelé le SHELL
  - N'appartient pas au noyau
- Des milliers d'utilitaires
  - D'outils de gestions de fichiers: ex: tar, ...
  - D'éditeurs de textes: ex vi, gedit
  - D'outils de programmation: ex. gcc
  - Fabriqué par vous ! Appelé des scripts (voir cours correspondant)

Historique

- 1969 début d'Unix avec le projet Multics (Echec). Ken THOMSON, Denis RITCHIE (BELL) développe un système mono-utilisateur, interactif en assembleur sur PDP-7 (DEC), puis PDP-9.
- 1971 Version multi-programme/temps partagé sur PDP-11.
- 1973 1ère version d'UNIX en C.
- 1975 UNIX version 6, sortie des laboratoires BELL (PDP-11).

Versions d'UNIX

- UNIX V7 (1978) 1ère version commerciale.
  - Effort pour la portabilité : compilateur C portable, noyau UNIX découpé en isolant les parties dépendantes de la machine.
  - Nouveaux utilitaires.
- UNIX System III (1982)
  - Fichiers FIFO pour la communication entre processus.
  - Outils d'atelier logiciel.
- UNIX system V (1983)
  - Éditeur de texte plain écran vi.

Depots des versions: `https://www.kernel.org/`

UNIX Berkeley

- Seconde grande branche
  - UNIX BSD 4.1 Version VAX avec gestion de mémoire virtuelle.
  - UNIX BSD 4.2
    - Gestion (les périphériques VAX).
    - Outils de communication (réseaux).
  - Version free: FreeBSD

Autres versions d'UNIX

- UNIX-like, réécriture d'UNIX (Solaris, IDRIS, Xenix, Linux, ...
- Portages d'UNIX (XENIX) par Microsoft.
- Versions free:
  - Linux (RedHat, Debian, . . .)
  - Sur Debian: Ubuntu, Mint, ...
  - cygwin (www.cygwin.org) sous windows
- Version Retravaillé en profondeur:
  - MacOs
  - Android,

Objectifs d'un Shell"

- Un "shell" est un langage :
  - Interprété (vs. compiler)
  - de commandes qui permettent à un utilisateur de lancer depuis le terminal l'exécution de programmes.
  - de script qui permet d'écrire des enchaînements de commandes
    - paramétrables,
    - avec l'utilisation de variables et des structures de contrôle riches (affectations, séquences, conditions, itérations).

Notez:

```
Le but est d'assembler des commandes pour réaliser
des taches plus complexes.
Il se distingue ainsi d'un langage de programmation
comme le C, Java,...
```

- Différents dialectes: sh, bash, csh, ksh, ...

Prise de contact avec UNIX

- Ouverture d'une session
  - Session = Une séquence de dialogues (commandes) échangée avec l'ordinateur pour le compte d'un utilisateur.
- `login` et `password`
  - Mettre le terminal sous tension. Le prompt `login` : doit apparaître.
  - Au message login : répondre par nom d'utilisateur <CR>.
  - Si vous avez défini un mot de passe, au message passwd : répondre le mot de passe <CR>.
  - Tapez une suite de commandes (décrites par la suite)
  - Fin de la session: exit ou ctrl-D

Forme générale des commandes d'UNIX sous SHELL

- Au prompt $, tapez une commande du shell.
  - Forme générale: $ < *commande* > [< *param* > ...] < *CR* >

Exemple

```
$ man zcat
$ cp toto titi
```

- Remarque :
  - En Shell, distingue les majuscules et les minuscules.

En résumé

- Pour maitriser le monde Unix, il faut apprendre:
  - les mécanismes du système
  - des commandes !!!, connaître des commandes de base !
  - Savoir écrire des scripts
- Comment trouver une commande dont on a besoin ?
  - Si on devine approximativement le nom: Exemple: `$ man -k date`
  - Google
  - Ce cours qui vous montre les commandes de base

SECTION: Système de Fichiers Unix

Un Fichier sous Unix

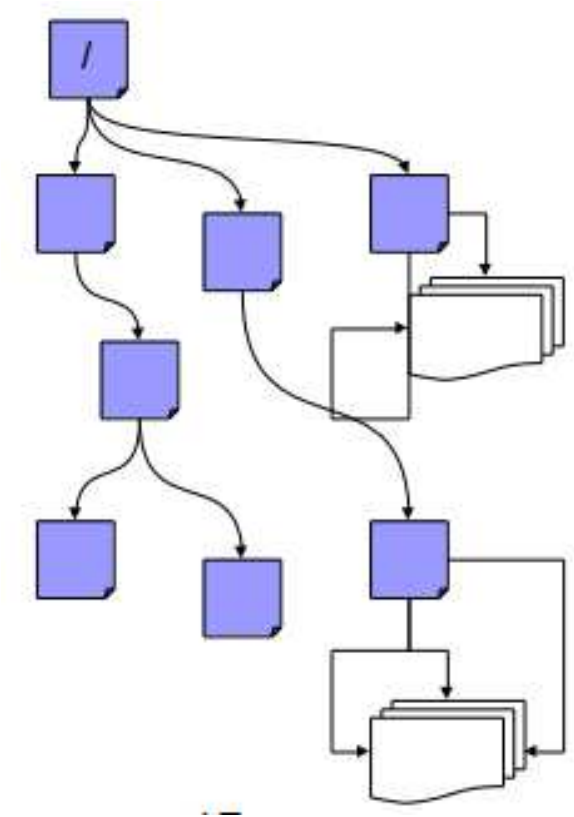
- Un fichier = suite de caractères ou d'octets.
  - Son contenu peut traduire différents objets: ex. Image, scripte, roman.
  - La création des fichiers est assurée par des utilitaires (ed, vi, cp, cat, ...), cf. plus loin.
  - La destruction est assurée par la commande `rm <fichier>`.

Quatre Fypes de Fichiers

- Le noyau Unix connaît quatre types de fichier. Les fichiers:
  - ordinaires (texte);
    - Seul ce fichier est modifiable directement par l'utilisateur.
  - répertoires;
  - spéciaux (souvent connectés à un périphérique). Divisés en des fichiers:
    - en mode caractère (pour les E/S). Ex. Modem
    - en mode bloc (pour les E/S). Ex. Disque
  - tubes (pipe en anglais).

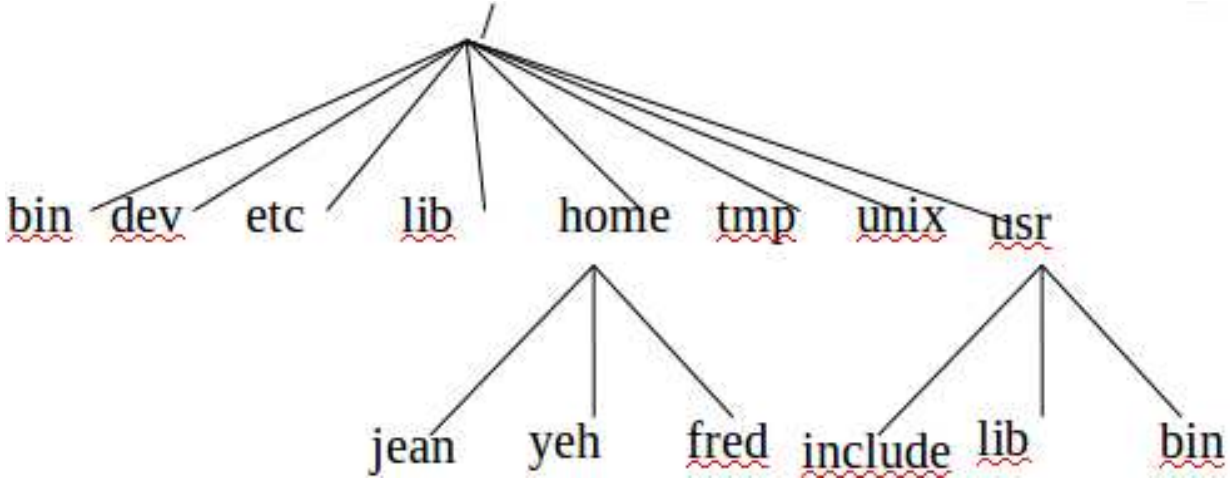
Notions de Répertoires

- Pour l'utilisateur un répertoire ("directory") est un fichier contenant d'autres fichiers ou répertoires.
- L'ensemble des répertoires est organisé en une hiérarchie ressemblant à un arbre inversé.
- Existe un répertoire racine / unique.
  - Les feuilles sont des fichiers ordinaires ou spéciaux (documents).
  - Les nœuds intermédiaires sont des répertoires (dossiers/chemises).



Principaux Répertoires

- Organisation typique



Rôle des Principaux Répertoires

- /bin, /usr/bin et /usr/ucb : les utilitaires du système (rechercher ls).
- /etc : commandes et fichiers de maintenance du système.
- /lib et /usr/lib : les bibliothèques de fonctions (.so .a, ex. libm.a).
- /usr/adm : fichiers de comptabilité.
- /usr/src : codes sources du système.
- /tmp et /usr/tmp : fichiers temporaires.
- /usr/include : fichiers en-têtes pour les programmes.
- /dev : fichiers qui représentent les périphériques d'entrée/sortie (a voir).
- /users ou autres

Le répertoire /dev

- Contient les fichiers spéciaux
  - Contient l'ensemble des périphériques attachés à l'ordinateur
    - Souris, le disque dur, etc. (voir la visite guidée)
  - Un fichier particulier: `null`

Désignation des Fichiers: **Chemin Absolu**

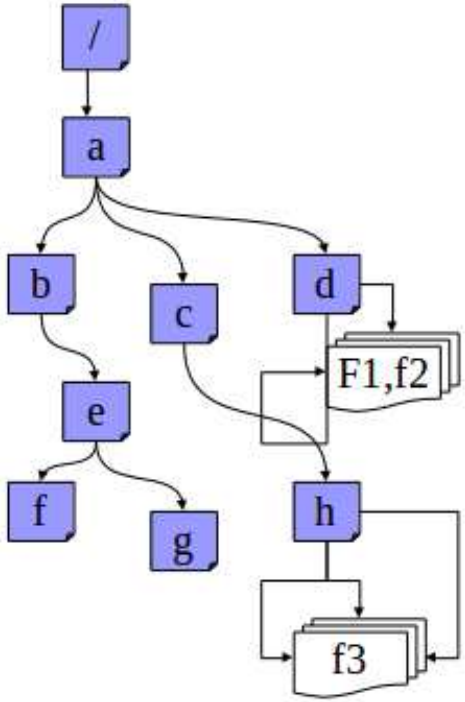
- Objectif: Pouvoir désigner un fichier
  - Chemin absolu/relatif
- Désigner un fichier par son "nom de base"
  - Associé à tout fichier : chaîne de caractères (Conseil: se limiter aux caractères alphanumériques et aux '_' et ".")
  - Ex. : programme.c , mon_programme.c
- Chemin d'accès absolu
  - Position précise dans l'arborescence des répertoires depuis la racine.
  - Reconnaissable car commence par '/'
  - La liste successive des répertoires à traverser en partant de la racine est séparée par des "/".
  - `ex/pwd.txt`

Désignation des Fichiers: **Chemin Relatif**

- A tout instant pour un utilisateur est défini un répertoire courant.
  - ex/pwd.txt
  - Où suis-je ?: pwd
- Toute désignation de fichier peut être définie relativement à ce répertoire en donnant un nom relatif
  - ne commence pas par " /"
  - le `<nom absolu> = <répertoire courant> + <chemin relatif>`
- Exemple:
  - Si le répertoire courant est /home/user
  - Si l'utilisateur désigne un fichier dans : projet/toto.c (chemin relatif)
  - Le chemin absolue est : /home/user/projet/toto.c

Exemples Chemins absolus/relatifs

- Chemins absolus:
  - Pour atteindre un répertoire: / , /a/d, /a/c/h, ...
  - Pour atteindre un fichier contenu dans un répertoire: /a/d/f2
- Chemins relatifs:
  - Si répertoire courant est /a/c
  - Pour atteindre le fichier f3 se trouvant dans h, $\Rightarrow$ ...
  - Car: ??



Nommage de Fichiers Spéciaux (. et ..)

- Par définition:
  - . (point) désigne le répertoire courant.
  - .. (point point) désigne le père du répertoire courant.
- Ex. :ex/ls-la-cd.txt

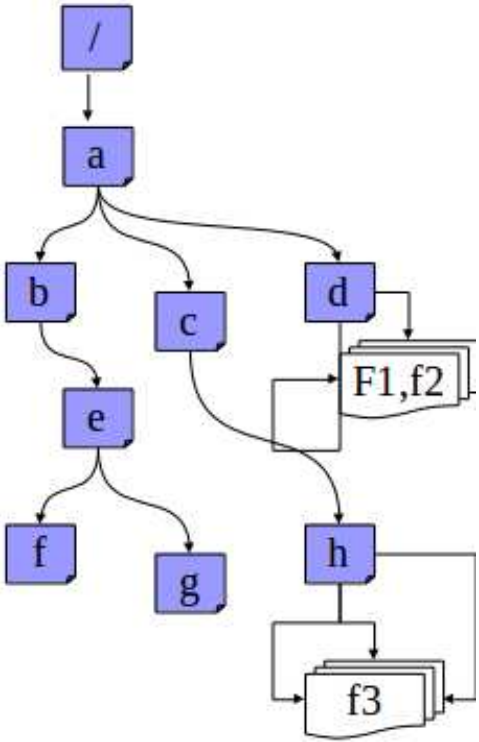
Génération de Noms de Fichier (joker/wildcard)

- * remplace n'importe quelle chaîne.
  - regarde dans le répertoire courant, les fichiers qui peuvent correspondre
  - Ex. ls test*.txt
- ? remplace un caractère quelconque.

  remplace l'un quelconque des caractères entre crochets.
- Ex. : *ce? [0-9] désigne tous les fichiers comportant dans leur terminaison la chaîne ce suivi d'un caractère quelconque et d'un chiffre décimal (0 à 9). ex/cd-etoile.txt

Commandes de Parcours du Système de Fichiers: **cd**

- Changement de répertoire : `cd`
  - Si le répertoire courant est `/a/c`
  - `cd h` permet d'aller dans le répertoire h fils du répertoire courant.
    - Le répertoire courant est: ???
  - `cd /a/b/e` permet d'aller à répertoire par un chemin absolu.
  - en tapant simplement cd, la commande renvoie l'utilisateur directement à son répertoire par "défaut/initial/home"



Manipulations de Répertoires: **pwd/ls/mkdir/rmdir**

- Répertoire courant : `pwd`
- Contenu d'un répertoire : `ls`
  - `ls` donne la liste des fichiers contenus dans le répertoire courant.
  - `ls <chemin>` donne la liste de fichiers pour le répertoire ¡chemin¿.
    - Remarque 1 : `-l` donne les attributs des fichiers.
    - Remarque 2 : `-a` donne tous les fichiers, y compris les fichiers de configuration commençant par un "." .ex.: .kshrc
- `mkdir <chemin>`
  - Exemple mkdir h/i
    $\Rightarrow$??
- `rmdir <chemin>`

Gestion des Fichiers

- Un fichier est constitué:
  - D'un descriptif
    - Un nom
    - Un i-node (descriptif) qui comporte : taille, propriétaire, groupe, droits d'accès, type de fichier, compteur de références (nombre de liens établis), adresse des blocs occupés,
  - Éventuellement d'un stockage
    - Ex. Fichier tty

Exemple

```
bash-2.05b$ ls -l
total 6
drwxr-xr-x    4 Administ Aucun          0 Dec 11  2002 cache
drwxr-xr-x    3 Administ Aucun          0 Nov 20  2002 log
drwxr-xr-x    2 Administ Aucun          0 Nov 20  2002 run
drwxr-xr-x    2 Administ Aucun          0 Nov 20  2002 tmp
drwxr-xr-x    5 Administ Aucun          0 Dec 11  2002 www
-rw-r--r--    1 Administ Aucun       5336 Aug 27 12:07 xmodmap.txt
bash-2.05b$
```

Protection des Fichiers

Exemple

```
bash-2.05b$ ls -l
total 6
drwxr-xr-x    4 Administ Aucun          0 Dec 11  2002 cache
-rw-r--r--    1 Administ Aucun       5336 Aug 27 12:07 xmodmap.txt
bash-2.05b$
```

- Classes d'utilisateurs. On distingue:
  - le propriétaire "user",
  - les utilisateurs du même groupe que le propriétaire "group",
  - tous les utilisateurs des autres groupes "other".
- Droits d'accès
  - Pour chaque classe d'utilisateur, on attribue le droit : r=lecture, w=écriture, x=exéc.

Protection des Fichiers: **chmod/chgrp/chown**

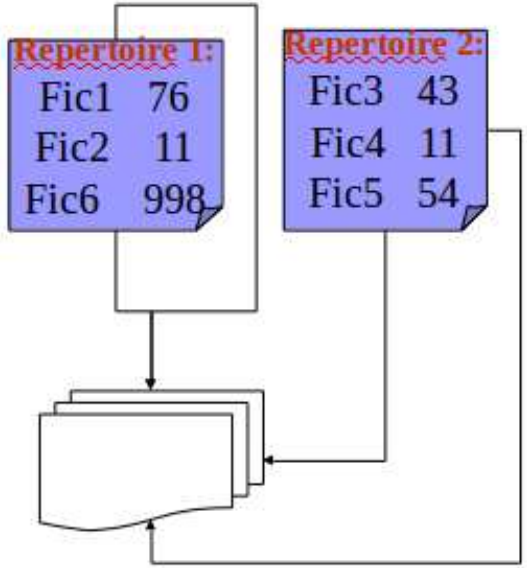
Exemple

```
bash-2.05b$ ls -l
total 6
drwxr-xr-x    4 Administ Aucun            0 Dec 11  2002 cache
-rw-r--r--    1 Administ Aucun         5336 Aug 27 12:07 xmodmap.txt
bash-2.05b$
```

- Commandes de modification des protections (utilisez man)
  - `chmod {u,g,o,a}{+/-}{r,w,x} <fichier>`.
  - `chgrp <group> <fichier>`.
  - `chown <nom> <fichier>`.
  - Exemple: `chmod g-r xmodmap.txt`

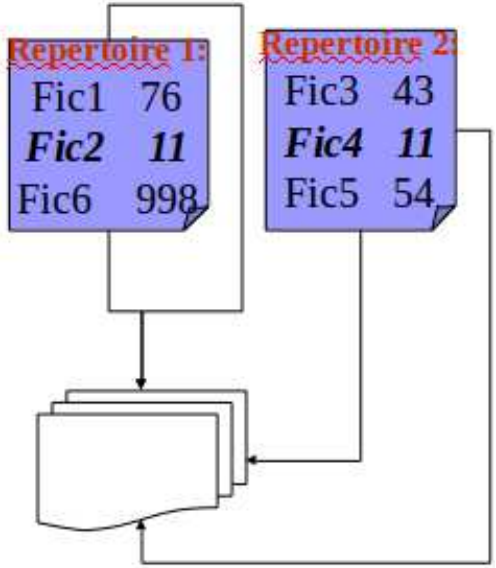
Répertoire & Inode

- Un répertoire
  - Est un fichier
  - Associe un nom à un Inode
  - Chaque Inode est associé les descriptifs d'un fichier:
    - Physique
    - Virtuel périph.
    - Virtuel Tube (pipe)



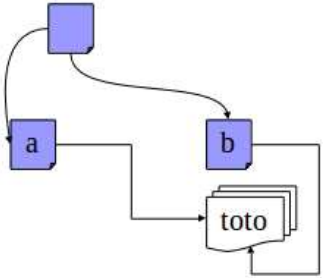
Liens

- Objectif: Accéder à un fichier depuis plusieurs répertoires.
- Pas de correspondance bi-univoque entre le nom absolu d'un fichier et son implantation physique.
- Un lien = relation entre un nom (descriptif) et le stockage physique d'un fichier.



Commande `ln`

- Lien symbolique et physique
- Création de liens (symboliques) `ln [-s] <source> <lien>`
  - Il y a création d'une nouvelle entrée dans le catalogue de fichiers qui correspond au nom lien. Le fichier physique est le même (pie celui du fichier d'origine.
  - Exemple Illustration EAL
- Exemple ex/ln2
- Restriction :
  - Il ne peut y avoir plusieurs liens "durs" sur un répertoire.
  - Pas de liens "durs" à travers les frontières de volumes.
    $\Rightarrow$ peuvent être levées par l'utilisation des liens "symboliques".



Les commandes `find/grep`

- `find <nom de rép.> <expression>`
  - Rechercher dans différents répertoires des fichiers
    - Utilisation de nombreux paramétrage (voir man find).
    - Exemple: `find / -name f1.txt {print`
- `grep [options] <expression> <fichier(s)>`
  - Rechercher dans un fichier d'une chaîne donnée,
    - Ex. : `grep -c f1.txt`
    - Donner le nombre de lignes contenant la chaîne toto dans le fichier a.
  - Une variante egrep ou `grep -e`

Commandes cp/mv/cat

- Copier un fichier
  - `cp <chemin source> <chemin destination>`
  - Ex. : `cp /a/b/f1 a` Illustration et `ex/cp-mouton.txt`
- Renommer/déplacer un fichier
  - `mv <source> <nouveau/destination>`
    - mv déplace et/ou change le nom d'un fichier.
      Exemple: `mv /a/d/f1 /a.` `mv /a/d/f1 f4`
- Recopie le fichier sur l'écran (sortie standard) `cat <fichier>`
  - `ex/cat-mouton.txt`
- Concaténation
  - `cat f1.txt f6.txt`
    voir Exemple CAT

Conclusion sur le Système de Fichiers

- Tout n'est que fichier! Uniformisation de la vision
  $\Rightarrow$ simplification
- Une arborescence pouvant devenir un graphe avec les liens symboliques.
- Connaitre les commandes de bases: `cd`, `pwd`, `mkdir`, `rmdir`, `rm` ...

SECTION: Gallery de Commandes

Commandes de Base: echo, passwd et man

- `echo <chaîne de caractères>` ; imprime son argument sur l'écran.
  - Exemple: `$ echo bon`
    `bon`
- `passwd` créer ou change le mot de passe.
- `man <commande>` explication d'une commande.
  - Très utile: `$ man -k date`

Manipulation de Contenu de Fichiers: cat, more, head, tail, od, tr

- `cat <Fichier>` ; imprime le contenu du fichier sur l'écran.
- `more <fic>` affichage du contenu d'un fichier page par page.
- `head <fic>` affichage les n premières lignes d'un fichier.
- `tail <fic>` affichage les n dernières lignes d'un fichier.
- `od [-o/c/d/x] <fichier>` Editer un fichier en (o)ctal, as(c)ii, (d)écimal ou hé(x)adécimal.
- `tr <fic>` translate ou efface des caractere dans un fichiers.
- `wc` compte le nombre caractères/mots/lignes.
- Utilitaires de comparaison de contenu de fichiers
  - `cmp [options] <fichierl> <fichier2>`
    - Comparer et arrêt, à la 1ère différence constatée sinon rien.
  - `diff [options] <fichierl> <fichier2>`
    - Donner toutes les différences sous la forme de commandes d'éditeur permettant de passer de l'un à l'autre.

Les commandes date, who, file, filetype, finger et id

- `date` donne la date et l'heure. voir `exemple/date.txt`
- `who` donne la liste des utilisateurs en session sur votre serveur.
- `file` détermine le type de fichier
- `filetype` détermine le type de fichier
- `finger <user>` user information
- `id <user>` user information

Utilitaires d'édition: lp

- Imprimantes:
  - `lpr [-h] [-P<nom imprimante>] <fichier>`
    - Impression du fichier sur imprimante `<nom imprimante>` (en utilisant le système d'impression).
  - `lp <fichier>` impression d'un fichier.
  - `pr <fichier>`
    - Recopier le fichier sur la sortie standard après avoir effectué une mise en page,

Utilitaires d'archivage/compression

- `tar [options] <fichier archive> <fichier(s)>`
  - Archivage de fichiers sur fichier, bande ou autres.
  - Exemple: `tar cvf mes_textes.tar *.txt`
- `(un)compress/g(un)zip <fichier(s)>`
  - (Dé)compresser la taille des fichiers.
  - Exemple: `gzip programmes.C.tar` donne `programmes-C.tar.gz`
  - `tar cvfz mes_textes.tgz *.txt`
  - `tar xvfz mes_textes.tgz`

SECTION: Editeur vi

Pourquoi vi ?

- Editeur de texte de dernier recours
  - Une installation sans X-Windows
  - Panne Système
  - Éditeur de l'administrateur système
  - aussi pour les developpeurs de logiciel, car très puissant !
- Un éditeur pour les nulles `nano`.

Notez:

Important de connaître dans le monde Unix si on n'utilise pas Unix que comme un outil bureautique !

L'éditeur de texte vi

- vi (prononcez Vihaille)
  - Plus populaire avec emacs dans le monde Unix.
  - Entièrement mode texte
  - Peu pratique, mais très puissant, et très utile en cas de grève X-Window
- Exécution de vi en introduisant la ligne suivante

```
$ vi <nom-de-fichier>
```

- Caractéristiques de vi
  - Utilisela notion de fenêtre
  - Notion de mot, phrase et paragraphe (exemple, effacer un mot ...)
- vi possède ses propres commandes
  - Ne pas confondre avec les commandes Shell

Bascule Entre Plusieurs Modes

- On peut basculer entre le mode *Direct* ou le mode *Commande*
- Dans le mode *Direct*, on a deux sous modes: *Normal* ou *Insertion*
- En mode *Normal*, on peut déplacer le curseur à l'aide d'une lettre, indiquer si l'on veut insérer des caractères, effacer caractères, lignes, etc:
  - Mode à l'ouverture du fichier. vi se met en attente de commandes.
  - Exemples, tapez:
    - h, j, k, et l pour déplacer le curseur
    - a ajouter du texte après le curseur,
    - x efface le caractère actuellement sous le curseur
    - dd efface la ligne actuellement sous le curseur...
  - Voir l'annexe pour plus d'info.
  - On bascule vers le mode
    - *Insertion* en tapant 'a','A','i',...
    - *Commande* (voir ci-après) en tapant ':'
- En mode *Insertion*, vous permet d'insérer des caractères
  - On peut revenir en mode *Normal* en tapant ESC

Bascule Entre Plusieurs Modes (2)

- Dans le mode *Commande*, on peut exprimer des commandes complexes.
- En mode commande direct, taper ':' pour basculer vers ce mode
- On peut revenir au mode *Direct* Normal en tapant `<cr>`
- Exemples d'expressions en mode commande:
  - Remplacer toutes les mots toto en titi.
    Exemple: `1,$s/toto/titi/g`
  - Effacer une sous partie du document texte.
    Exemple: `10,20d`
  - Recopier un bloc de textes puis coller ailleurs (copier coller).
    Exemple: `10,20y` ; recopie
    Puis aller à l'endroit et tapez 'p' pour coller le morceau de textes.

Commandes en Lignes

- Autres exemples:
  - :wq Enregistrer et puis quitter vi.
  - :w [file] Enregistrer le tampon d'édition dans un fichier appelé file . Sans argument, enregistrer sur le fichier courant.
  - :sh Invoquer shell temporairement pour exécuter qq commandes. Ctrl-d Finir le shell temporaire et revenir à vi.
  - :u pour défaire une action
  - :q Quitter vi.
  - :q! Quitter vi sans enregistrer.

Désigner une ou plusieurs lignes d'un document

- Dans le mode Commande, désigner des lignes permet de se déplacer ou d'appliquer des actions sur un ensemble de lignes.
- Désigner une ligne permet
  - Pour se déplacer:
    - :n Placer le curseur à la ligne numéro n.
    - :$ Déplacer à la dernière ligne du tampon. De manière générale, $ désigne la dernière ligne.
  - Pour une action
    - :3d – efface la ligne 3 en mode commande
- Désigner plusieurs lignes pour des actions
  - :3,5d– efface les lignes 3 à 5
  - Autres exemples:
    - :3,9w file Enregistrer les lignes de 3 à 9 dans le fichier file .
    - :.,$d Effacer jusqu'à la dernière ligne à partir de la ligne courante.
    - : 10,$s/old/new/ Remplacer la première occurrence de `old` par `new` depuis la ligne 10 jusqu'à la fin
      Ici `old` peut être une expression régulière (à voir cours suivant)

Commandes Utiles à Savoir

- Copier coller de lignes
  - Sur la ligne, tapez ¡n¿ yy pour n lignes, ou bien yy pour une ligne. Les lignes sont mises dans un tas.
  - Pour coller, aller à la ligne cible, puis tapez pp.
- Variables d'environnement
  - Pour permettre de paramétrer l'éditeur
    - :set
    - :set all
      $\Rightarrow$ Pour activer des variables
  - Exemples:
    - Set number (ou set nu)
    - Set nonumber
- La commande :help

et **vim** ?

- sur-ensemble de vi
- coloriage syntaxique
- éditer via des protocoles réseaux (SSH and HTTP)
- découpage fenêtre en sous fenêtres
- peut éditer un fichier compressé en gzip, zip, tar, etc.
- admet des plugins.
- vimscript via des langages comme python, perl, shell
- ...

vi en résumé

- Un editeur puissant
- Demande de la pratique pour sa maitrise
- Peut être un premier pas vers emacs !
- Un éditeur plus simple: nano

SECTION: Expressions Régulières

C'est quoi "Expressions Régulières" ?

Objectif:

Permettre de spécifier un motif pour rechercher des occurrences dans un flot de texte.

- Exemple: ecouter un flux de texte et trouver les mots *password*
- Utilisée par des outils comme: `vi, sed, grep,find, ...`
- Très puissant pour manipuler du texte
  - Exemple: `egrep 'Chapitre [1-6]' roman.txt`
- Existe des méta-caractères: `[ . ? ^` etc.
  - Qui ont chacune une signification précise
- Ne pas confondre avec les expressions de jokers !
  - Qui sont utilisées au niveau du Shell
  - Exemple: * en Shell signifie l'ensemble des fichiers du répertoire courant, et * en expression régulière signifie répétition d'occurrence.

Un Caractère Quelconque. Le (.)

Le point (.)

Dans une expression régulière, le point (.) est un méta caractère qui représente n'importe quel caractère sauf "aller à la ligne".

- Exemple :
  - Chercher toutes les lignes concernant la seconde génération des processeurs d'Intel.
  - `grep -e '80.86' histoire-micro`
    Donne "80286" et au "80386". grep-point

Excercice:

Exprimer sous vi le remplacement dans le texte entier de la phrase "processeur 80X86" par le mot "CPU processor".

Répétitions d'un Caractère

- Le méta caractère astérisque (*)
  - indique que l'expression régulière précédente peut apparaître zéro ou plusieurs fois.
  - Exemple:
    - [15]0∗
    - [15]00∗
    - Valident 1 5 10 50 100 500 1000
  - grep '< /∗ >' liste-entete
- Le méta caractère (+)
  - indique que l'expression régulière précédente peut apparaître au moins une fois.
  - Résultat pour les même exemples

Accolades

- Indique un nombre de répétitions.
- Possible que dans les expressions régulières étendues.
- Exemples:
  - a{1,3}
    ⇒ a, aa, aaa et rien d'autre
  - abc{2,4}d
    ⇒ abccd, abcccd, abccccd et rien d'autre
  - a.{2,2}z ou a.{2}z
    ⇒ abcz, aXXz, ak0z, ...

L'omniprésente contre-oblique & Ce qu'on recherche

- Le méta caractère contre-oblique (\)
  - transforme les méta caractères en caractères normaux.
    - Ex: `grep -e '\.' texte.txt`
    - Exemple: comment trouver `\f` dans un texte ?
      `grep -e '\f' texte.txt`
- Ce qu'on recherche
  - Difficulté
    ⇒ Décrire le bon motif de recherche.
  - Exemple de problème:
    - Rechercher Qui et qui
    - Utiliser "ui" ne produit pas un résultat satisfaisant
  - Plus on utilise de méta-caractères
    ⇒ plus c'est précis.

Classes de Caractères

- Définit par les méta-caractères [ ]
  - Choix d'un caractère dans l'ensemble
  - Une classe de caractères est un raffinement du méta caractère (.) .
- Exemple
  - grep 'H[12]' liste-entete.txt
  - Voir le résultat: `ex\grep-classe.txt`
- Comment faire pour trouver Qui/qui ?

Excercice:

Exprimer sous vi l'effacement de toutes les voyelles dans les 10 premières lignes du texte.

Question:

Comment faire dans un éditeur sans expression régulière ?

Intervalle de Caractères & exclusion

- Exemple:
  - Chercher toutes les entêtes comprises entre 0 et 9:
- On peut utiliser des intervalles de caractères
  - Syntaxe exemple: `[0-9]` `[a-z]`
  - Exemple: `grep '[cC]hapitre [0-6]' mon_roman.txt‘`
  - Exemple
    - Occurence `[cmt]an`
      $\Rightarrow$ can, man, tan mais ne trouve pas pan, ...
    - `a[a-d]z`
      $\Rightarrow$ aaz, abz, acz, adz et rien d'autre
- Exclusion d'une classe de caractères
  - L'accent circonflexe `'^'` en première position de la classe exclut de la concordance tous les caractères repris dans la classe.
  - `[^aeiou]`

Méta-caractères positionnels

- Permet de désigner le début ou la fin d'une ligne.
- ^ indique le début de ligne
  - Exemple: `grep "^chapitre" doc.txt`
- $ indique la fin d'une ligne
  - Exemple: `\grep "\.\$" doc.txt`
- Compte le nombre de lignes vides
  - `grep -c '^$' doc.txt`
- Debut et fin d'un mot
  - `grep '\<b' doc.txt`
  - `grep 's\>' doc.txt`
  - `grep '\<i\>' doc.txt ext (i=10`
  - `\<.*\> bonj.our!`

L'alternative

- | barre verticale marque l'alternative
- Alternative $a|b$
  $\Rightarrow$ a, b $(a|b)+$
  $\Rightarrow$ a, b, ab, abb, bbbbbb, ababb, ...
- Exemple: ^ $(De|Sujet|Date)$: @ reconnaît tout ce qui commence par De:@ ou Sujet:@ ou Date:@

Group Pattern

- Utile dans les expressions de substitution. Ce qu'on trouve dans la partie recherche, on peut le rappeler dans la partie remplacement.
- Pour isoler un groupe: `\(pattern\)`
- Pour rappeler un groupe : `\number`
- Exemples:

Exemple: Rajouter à chaque numéro de téléphone, l'indicatif 33 sous vi

```
1,$s/\([0-9]*\)/33 \1/g
```

- Différences entre sed et vi ?
  - vi
    $\Rightarrow$ un fichier
  - sed sur un flot (ligne par ligne)
    L'exemple précédent s'exprime en: `s/\([0-9]*\)/33 \1/`

Group Pattern (2)

- Autres Exemples:
  - `s/^\(.*\):.*/\1/g`
    - `Yeh:234`
      $\Rightarrow$ Yeh
  - `1,$s/key=\([0-9]+\)/cle=0\1 3x/ key=345 rego rfrg`
    - `Frrf key=677 reddef`
  - `s/\(.*\):\(.*\)/\2:\1/g`

Question:

Dans l'exemple du no de téléphone, comment faire pour rajouter 33 entre parenthèses?

Un mot

- $(tipo|adp)+$
  - Exemple:
    - abcabcd
    - deux acb k
    - donc abc
    - adp action
    - abt dic
  - `$ egrep '(abc)+' txt1`
    - abcabcd
    - donc abc

Expressions Régulières Étendues

\w	A-Za-z0-9_
\W	^A-Za-z0-9_
\a	A-Za-z
\a	A-Za-z
\s	\t
\b	Positions de début et fin de mots
\B	Positions ne correspondant pas à un début ou une fin de mot
\d	0-9
\D	^0-9
\l	a-z
\p	\x20-\x7E
_s	\t\r\n\v\f
\S	^ \t\r\n\v\f
\u	A-Z
\x	A-Fa-f0-9

Expressions Régulières Étendues

`expr{n}`	Exactement n occurrences de l'expression précédant les acc
`expr{n,}`	Au moins n occurrences de l'expression précédant les accol
`expr\{m,n\}`	m à n occurrences de ce qui précède,

En Résumé des Expressions Régulières

- Un langage pour identifier des chaines de caractères très puissant
- Beaucoup d'outils l'utilise et même un langage (perl)
- Interfacer dans beaucoup de langage de programmation

SECTION: Langages de Commandes

Langage de Commande Shell

- Shell (coquillage) : enveloppe externe du système UNIX,
  - Deux/trois versions importantes :
    - .shell de base fourni par ATT, Bourne shell /bin/sh
      $\Rightarrow$ prompt : $
    - C-shell /bin/csh de l'université de Berkeley disponible principalement avec UNIX BSD 4.2.X. Compatible avec la version de base, mais présentant des fonctions plus riches (historiques de commande, alias, contrôles de travaux)
      $\Rightarrow$ prompt : %
- Variantes plus récentes: `/bin/bash`, `/bin/ksh`, `/bin/tcsh`, `/bin/zsh`, `...`

Les Modes d'Exécutions

- On peut lancer une série des commandes, mais il faut les synchroniser
- Si une commande dépend du résultat de la commande précédente, alors il faut l'exécuter en synchrone.
- dans le cas contraire, on peut exécuter en asynchrone
  - Bénéfique, surtout si l'une des commandes attends des traitements qui n'occupent pas entièrement le processeur.
  - Meilleur utilisation des ressources.
  - Exemple: d'un script utilisé plus tard qui fait:
    - Je compte les moutons durant X seconde(s)
    - il sommeille X seconde(s)
    - if affiche: J'ai fini de compter et je ne suis toujours pas endormi !

Exemple:

```
$ mouton 6
Je compte les moutons durant 6 seconde(s)
J'ai fini de compter et je ne suis toujours pas endormi !
```

Modes d'exécution Synchrones

- Chaque commande est achevée avant le lancement de la suivante.
- Cas d'une commande qui doit attendre qu'un autre ait terminé l'exécution avant de commande.
  - Terminées par $< CR >$ ou
  - séparées par un ";" pour en mettre plusieurs sur une même ligne.

Exemple:

```
mouton 6
ls
# Equivaut
mouton 6 ; ls
```

  - le ; permet d'exécuter des commandes en mode synchrone

Modes d'exécution Asynchrones

- Commandes simples "asynchrones" ou (arrière plan)
  - si une commande est déterminée par &, n'attend pas sa terminaison pour lancer la suivante.
  - Exemple: `mouton 6 & ls`

Les Scripts

- Scriptes (procédure programmé)
  - Liste de commandes enregistrées dans un fichier
  - Exécution de la liste de commandes en évoquant le nom du script.
  - Cette liste de commandes devient à son tour une commande.
    - Utilisation des scripts pour écrire des programmes/commandes

Exemple:

```
$ cat mouton
echo "Je compte les moutons durant $1 seconde(s)"
sleep $1
echo "J'ai fini de compter et je ne suis toujours pas endormi !"
```

- Sous un shell, l'invocation d'un nom de script lance un autre programme (processus)
- Un script (père) peut lancer un ou plusieurs autres scripts (fils)
  - Un graphe d'exécution en forme d'arbre

Un Script Exécutable

- La ligne magique !
  - `#!/usr/bin/bash`
  - En première ligne
- Ce qu'il faut faire pour le rendre exécutable
  - `chmod u+x monprog`

Exemple:

```
1 #!/bin/bash
2 echo "Je ($2) compte les moutons durant $1 seconde(s),
      pid=$$"
3 sleep $1
4 echo "J'ai ($2) fini de compter et je ne suis toujours
      pas endormi !"
```

Utilisation des Scriptes

- Automatisation de taches
  - Exemple: Synchronisation de disques
- Développement de programme
  - Automatisation de la séquence dans un script

Exemple

```
extraire tm.c
doc tm.c
cc {o tm.o tm.c
ld ....
ar
```

Variables d'Environnement

- Rendre persistant certaines informations d'un environnement (préférence utilisateur)
  - Exemple, quel est mon répertoire de base. `echo $HOME`
  - Exemple: Comment vérifier qu'un script est bien lancé depuis un répertoire précis ?
- Variables
  - Pour contenir une donnée
    - Un seul type : les chaînes de caractères.
    - Nom de variable : une chaîne alphanumérique.
  - Remarque: Toujours en majuscule par convention

Définir le Contenu d'une Variable

- Définir (suivant les shells):
  - `var=valeur`
  - `export var=valeur`
  - `setenv var valeur`
    definit dans (t)csh, en *bash* c'est équivalent à `export`
  - il sera exposé plus tard les différences entre ces affectations
- Définir des variables positionnelles :
  - `set var ma valeur`
  - alors echo $1 $2 $3 contient ?

Détruire/Afficher une Variable

- Détruire:
  - `unset var`
  - `unsetenv var`
- Afficher les variables:
  - `echo $<var>`
    - exemple: `echo $HOME`
    - Affiche le contenu d'une variable
    - Notez le $ avant le nom de la variable
  - env ou printenv
    - affiche l'ensemble des variables
- Par la suite, nous verrons l'usage des variables dans les scriptes.

Variables Connues

Le Shell pour son fonctionnement, pour la personalisation, pour ... utilisent un ensemble de variables qu'il faut connaître

- `HOME` nom du répertoire par défaut de l'utilisateur.
- `PATH` répertoires dans lesquels le shell va chercher les commandes.
- `MAIL` nom absolu de la boîte aux lettres del'utilisateur.
- `TERM` modèle du terminal utilisateur.
- `PS1` prompt principal.
- `PS2` prompt secondaire.
- `IFS` liste des caractères reconnus comme séparateur.

Utilisations des Variables dans les Scripts

- Affectation: $< variables >=< valeur >$
- Exemple: `VAR2=bon`
- Les guillemets/quottes sont nécessaires que si la valeur contient des espaces.
- Accéder au CONTENU d'une variable: `${variables}`
  - Exemple: `echo $var2`
    $\Rightarrow$ bon
- Opération de concaténation : suite de variables
  - Exemple : `echo $var2 "!"`
    $\Rightarrow$ bon !
- Affectation multiple par clavier : read $< variables >$

Utilisations des Variables dans les Scripts (2)

Exemple

```
$ read v1 v2 v3
Voici un texte
$ echo $v3
texte
```

- Remarque :
  - Pour indiquer clairement la substitution à réaliser, il faut éventuellement parenthèser (avec des accolades).
  - Ex.: `echo ${var2}jour = bonjour`
- Variable avec séparateur :
  - Exemple: `echo $var2 jour = bon jour`
- Variable sans séparateur :
  - Pour indiquer clairement la substitution à réaliser, il faut éventuellement parenthèser (avec des accolades).
  - Exemple: `echo ${var2}jour = bonjour`

Visibilité des Variables

- Variables locales (au shell), elles sont perdues lorsque l'on passe d'un script/Shell père au script/Shell fils.
- Variables d'environnement : `export <variable(s)>`
  - Permet de rendre globale une variable pour tous les Shells descendants.
  - Exemple: export V1='toto'
    - Remarquer qu'il n'y a pas d'espace après V1
- Variables pré-définies :
  - fichier `.profile`
    - exécute au login permet d'initialiser des variables d'environnement connues de tous les processus descendants du shell initial.

bash, Extraction de Sous Chaînes

- Longueur
  - `v="yeh laurent"`
  - `echo ${#v}`
- Par défaut
  - `echo ${var:-defaut}`
- Invocation cmd
  - `v=$(ls {l)`
- Calculs et boolean
  - `echo $(( 2 * ( 4 + 1) ))`
- Remplacement
  - `${var/motif/rempl}`
  - `echo ${PWD/$HOME/Maison}`
- Sous-chaine
  - `${var:début:lg}`
  - `echo ${v:5:2}` $\Rightarrow$ la
  - `echo ${v:5}` $\Rightarrow$?
- Supprime à gauche
  - `echo ${v#yeh }` $\Rightarrow$ laurent
  - `echo ${v#*u}` $\Rightarrow$ rent
  - `echo ${v#*[le]}` $\Rightarrow$ h laurent

bash, Extraction de Sous Chaînes (2)

- `v="yeh laurent"`
- Plus long préfixe
  - `echo ${v##*e}`⇒ nt
- Garde le préfixe
  - `echo ${v%l*}`⇒ yeh
  - `echo ${v%[tn]*}`⇒yeh lauren
- Plus grand prefixe
  - `v=yeh@prism.uvsq.fr`
  - `echo ${v%.*}`⇒ yeh@prism.uvsq
  - `echo ${v%%.*}`⇒ yeh@prism

Les tableaux

Exemple:

```
$ arr=(Bonjour "Laurent Y")
$ echo ${arr[0]} et ${arr[1]}
Bonjour et Laurent Y
```

Exemple:

```
$ echo $arr
Bonjour
$ echo ${arr[*]} # Tous les items du tableau
Bonjour Laurent Y
$ echo ${!arr[*]} # Tous les indices du tableau
0 1
$ echo ${#arr[*]} # Nb d'elements du tableau
2
$ echo ${#arr[0]} # Taille de l'item 0
```

Les tableaux

Exemple d'utilisation:

```
$ for index in ${!arr[*]}
> do
>     printf "%4d: %s\n" $index "${arr[$index]}"
> done
   0: Bonjour
   1: Laurent Y
```

Après décotage, que vaut echo ${#arr1[*]} ?

```
$ arr1=(${arr[*]})
$ for ix in ${!arr1[*]} ; do printf "   %s\n" "${arr1[$ix]}"
> done
  Bonjour
  Laurent
  Y
```

Les tableaux

Exemple après quote expansion:

```
$ arr2=("${arr[*]}")
$ echo ${arr2[*]}
Bonjour Laurent Y
$ echo ${#arr2[*]}
1
$ arr3=("${arr[@]}")
$ echo ${arr3[*]}
Bonjour Laurent Y
$ echo ${#arr3[*]}
2
```

En Résumé Langage Commandes

- Savoir:
  - contrôler l'exécution de commandes en synchrone ou asynchrone,
  - les inscrire dans un fichier qui devient exécutable (script)
  - définir des variables

  permets d'exploiter un système pour des taches...

SECTION: Processus

Les processus

- **Définition** : un processus est fondamentalement un programme qui s'exécute (Tanenbaum).
  - Programme exécutable, avec ses données et sa pile d'exécution, son compteur ordinal, son pointeur de pile et autres registres, ainsi que toutes les autres informations nécessaires à l'exécution du programme!!!
- Environnement d'exécution
  - Un processus comporte schématiquement une zone de code exécutable et une zone de données,

Gestion des Processus

- Un serveur Unix = un ensemble de processus
  - Exemple:
    - Un processus d'accueil (cron)
    - Un shell
- Tape le nom d'une commande
  - Création d'un processus
    - Shell qui lance le processus

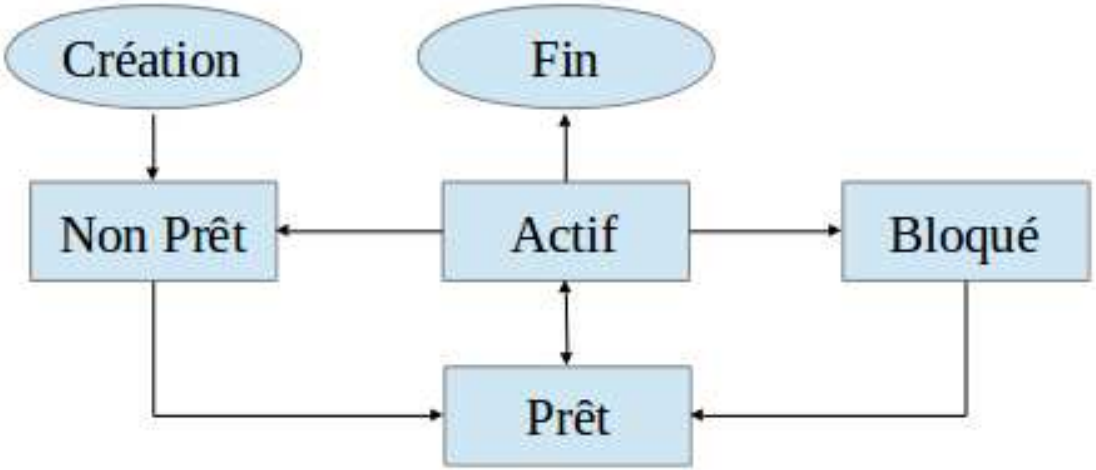
Commande de Gestion de Processus: **ps/kill/sleep/nice**

- `ps [options]`
  - liste de processus actifs.
  - Exemple: `ps -aux`
- `kill <processus>`
  - destruction d'un processus.
  - Exemple: `kill -9 5209`
- `sleep -n`
  - suspend l'exécution pour n secondes d'une commande.
  - Exemple `ex/ps-kill.txt`
- `nice -n <commande>`
  - exécute la commande avec une priorité n donnée.

Commande de Gestion de Processus: **at/wait/exec/exit**

- `at <heure>`
  - lancement d'une commande à une date absolue.
  - Exemple: `at llam ps {aux`
- `wait <processus>`
  - attente de la fin d'exécution d'un processus défini par son numéro.
- `exec <commande>`
  - la commande donnée est exécutée sans créer un nouveau processus à la place du shell. A la fin de la commande, on est donc déloguer.
- `exit n`
  - termine une suite de commande avec un code de retour n.

Transition d'état d'un processus



Vie des processus

- Les processus sont manipulés par 4 (opérations) primitives offertes par le noyau :
  - **fork** permet pour un processus père de créer un processus fils qui est sa copie conforme (même texte, même données) au moment de la création, Le fils peut ensuite selon ses branchements modifier son comportement.
  - **exec** permet à un processus de remplacer le code et les données qu'il exécute par un autre correspondant à un autre programme. Des arguments peuvent être communiqués. Le nouveau processus s'exécute dans le même contexte (fichiers, répertoires courants, ...)
  - **exit** fin normale d'un processus.
  - **wait** attente de la terminaison d'un fils

Job

- Chaque liste de commandes séparées par des ';' est considérée comme un job et est attribuée un numéro par le shell.
  - Un "job" peut être arrêté provisoirement en utilisant le signal TSTP (généré par le caractère susp de la commande stty). (Ctrl-Z)
  - **jobs**: commande interne permet d'avoir la liste de tous les jobs y compris ceux en arrière plan ou bien arrêtés.
  - Par exemple, un job de numéro j arrêté peut reprendre l'exécution en tapant: %j
- Un job qui tourne en arrière plan est arrêté quand il tente de lire le terminal. Par contre, il peut écrire sur le terminal à moins que la commande stty tostop ait été utilisée.
- Un ensemble de commandes internes permet de manipuler les jobs. Ce sont les commandes `bg`, `fg`, `kill`, et % .

Entrées Sorties des Processus

- Fichiers standards Entrée/Sortie (E/S)
  - Toute exécution de commande (plus généralement de processus) s'effectue avec assignation par défaut de 3 unités standards E/S.
- Par défaut, les trois flux de données précédents sont assignés au terminal de l'utilisateur `/dev/tty...`, un flux associé à un élément :
  - d'affichage (écran en général)
  - de saisie (le clavier en général)
  - d'affichage des messages d'erreurs

Redirections Simples: <>>>

- Par défaut, un processus est connecté aux trois flux E/S.
- On peut pour la durée de l'exécution d'un processus changer les flux E/S par des directives.
- > *fichier* prend un fichier comme sortie standard.
  - Exemple: `ls > t` voir ex/ls-redirect.txt
- < *fichier* prend un fichier comme entrée standard
  - Exemple: `wc < t` voir ex/ls-redirect.txt
- >> *fichier* comme >, mais ajoute à la suite du fichier existant.
  - Exemple: `ls >>t`
  - Avec t existant déjà voir ex/ls-redirect_add.txt

Rediriger la Sortie Standard "erreur"

- Utilité: Séparer la sortie standard de la sortie erreur.
  - Exemple: Un programme analyse des données et produit des résultats dans un fichier normal. Toutes les erreurs sont redirigées vers un autre programme (exemple mail) qui prévient l'utilisateur.
- Exemple: `cc toto.c 2>t`
  - Redirige les erreurs (sortie standard 2) vers le fichier t
  - `echo "ooo" >&2`
  - `do_something 2>&1`
  - `some_command >file.log 2>&1`
  - `rm -f $(find / -name core) &> /dev/null`
  - `exec monprog 3>&1`

Connexion des E/S Entre Commandes

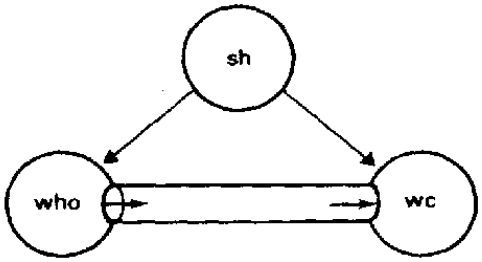
- Exemple
  - `$ who > tmp`
    `$ wc -l < tmp`
- Equivalent (presque) à
  - `$ who | wc -l`

Redirection entre processus: tubes (pipes)

Définition:

Connecte la sortie standard d'un processus (programme) sur l'entrée d'un autre.

- `<commande 1> | <commande 2>`
- Les processus s'exécutent en parallèle
- Chaque partie séparée par '|' est de la forme ¡cmd¿ ¡arguments¿....



Redirection entre processus: tubes (pipes)

- **Fonctionnement**: Le noyau créé un fichier virtuel appelé *tube* de qq Ko (dans /dev).
  - La commande de gauche envoie les données de sa sortie standard à l'intérieur.
  - Si le tube est plein, la commande à gauche est bloquée en attente.
  - du côté de la commande à droite, il lit les données au fur et à mesure sur son entrée standard.
  - Si il n'y a plus de données, il se met en attente.
- Un tube est un fichier en mémoire. Il n'y a pas d'E/S disque.

Redirection entre processus: tubes (pipes)

- Exemple 1 : rechercher le nombre de lignes dans un répertoire
  `ls | wc -1`
  - ls donne les noms des utilisateurs sur une lignes.
  - wc -1 compte le nom de lignes d'un fichier. @n voir ex/ls-wc.txt
- Exemple 2: rechercher tous les processus de caillou
  `ps -elf | grep caillou`

Enchaînement de processus par les tubes

- Propriété fondamentale
  - Combine des commandes pour obtenir un résultat

Exemple

```
$ capture | filtre_immo | sort | tail 10
$ capture | avg_compute
```

Re-direction en T

Problème

On ne peut rediriger un flux qu'une seule fois, comment rediriger la sortie standard vers la sortie standard et simultanément vers un fichier.

- Re-direction en T
  - La commande `tee` permet de recopier son entrée standard vers sa sortie standard et également vers un fichier dont le nom est passé en argument.
  - Exemple: `who | tee liste | wc -l > nombre`
  - Fournit d'une part la liste des utilisateurs dans le fichier `liste` et leur nombre dans le fichier nombre.
  - Équivalent (presque) à `who > liste ; cat liste | wc -l > nombre`

La Sortie Standard Vers Arguments: xargs

- Prendre les données de l'entrée standard et le placer en argument d'une document
  - Entrée standard redirigée en argument

Exemple

```
$ find . | xargs  ex/find-xargs
Difference ? : ex/find-xargs `find .`
Difference ? : for i in `find .`
 do ex/find-xargs $i
 done
```

Un outil sur les flux: l'outil sed

- Pour modifier un flux de texte, pour modifier le contenu d'un fichier par une commande, il y a *sed*.
- Autrement, il faut taper avec un éditeur comme *vi*.
- Éditeur de flot de textes
- Utilise les expressions régulières
- Exemple: `$ sed {e 's/80.86/processeur Intel' note_néophyte.txt`

Paramètres Positionnels

- Paramètres

Exemple

```
$ cat exemple
echo $1 est égal $2 ?
$ exemple toto titi
toto est égal titi ?
```

- Paramètres positionnels :
  - les paramètres d'appel sont accessibles au moyen des noms réservés $n, où n est la position du paramètre (1 <:).
- Tester la présence d'un paramètre `echo ${1:?"1er param. absent"}`

Paramètres (2)

- Autres variables accessibles ;
  - $0 le nom de la procédure.
  - $* la liste des paramètres positionnels.
    - "$*" equivaut à "$1,$2,... ${n}"
  - $@ la liste des paramètres positionnels.
    - "$@" equivaut à "$1","$2",... "${n}"
  - $# le nombre des paramètres.
    - `if [ $# -ne 3 ] ; then echo "Veut 3 args et non $#" ; fi`
  - $$ le numéro du processus qui exécute la procédure.
  - $? code retour après exécution d'une procédure.

set

Définition

Assigne les paramètres positionnels

Exemple: ex/set.txt

```
set a=‘ls ex*‘
echo a= $a
echo "positionnel " $*
```

Résultat

```
$ ./set
a=
positionnel  a=ex exemples.pdf exemples_run.tex exemples
```

- Cas de $-

shift

Décalage des paramètres positionnels vers la gauche

Exemple: ex/shift.txt

```
while [  $# -gt  0 ]
do
echo  $*
shift
done
```

Résultat

```
$ ./shift un deux trois
un deux trois
deux trois
trois
```

- soit
  - $1 ⇐ $2, $2 ⇐ $3 etc.
- Exemple: ex/shift.txt

Contrôle d'une Séquence de Processus

- Permet d'aligner des traitements plus complexe qu'une simple séquence de commandes.
  - Exemple: un script lance l'exécution d'une impression, si ça s'est bien passé, il lance une autre tache, sinon envoie un message.
- Rappel:
  - exit $< n >$
  - Normalement 0 si ok, sinon code d'erreur.

Code de retour & Conditions logiques

Outre ;, il existe d'autres modes de contrôle d'enchaînements de commandes

- `commande1 && commande2`
  - Si la commande1, se termine mal alors commande2 n'est, pas activée.
  - Si la commande1 se termine bien alors commande2 est exécutée et son code retour est celui de l'ensemble.
- `commande1 || commande2`
  - N'exécute la seconde commande2 que si la commande1 se termine mal.
  - Sinon commande2 n'est pas exécutée.

Conclusion des Processus

- La gestion des processus,:
  - c'est les créer, les détruire, les arrêter, ...
  - c'est de connaître son état de fonctionnement
  - c'est de les connecter entre eux, et/ou au fichier

SECTION: Langage de Scripts

Quoting & Double Quoting

- Le *Quoting & Double Quoting* servent à isoler un ensemble de mots représentant un seul argument
- Le *Quoting & Double Quoting* se distinguent par la substitution de variable ou non

Exemple

```
$ NB=10
$ echo 'nombre = $NB'
nombre = $NB
$ echo "nombre = $NB"
nombre = 10
```

Back-quoting: Substitutions de Commandes

- Objectif: Récupérer le résultat d'une exécution d'une commande
- Fonctionnement:
  - N'importe quelle commande shell peut être placée entre ‘ (back-quoting)
    - Appelé (quote à l'envers, "back quotes").
    - Le résultat de l'exécution peut être affecté à une variable.
  - Forme equivalente en bash: `$(...)`

Exemple 1:

```
$ sudo chown ‘id -u‘ /somedir
```

Exemple 2:

```
$ NB-UTIL=‘ls | wc -l‘
$ echo $NB-UTIL
$ 2
```

  - En *bash*, équivalent `$(ls | wc -l)`

Ordre de Substitutions de Commandes

- Fondamental pour comprendre le fonctionnement du Shell
  - En cela, se distingue d'un langage de programmation.
- L'ordre de substitution est le suivant ;
  - substitutions des variables,
  - substitutions des commandes,
  - séparation des arguments,
  - génération des noms de fichiers.

Exemple:

```
cmd='ls -l'; list="ficl fic2"
$cmd $list part* `echo fic3`
```

- Subs. variables `ls -l ficl fic2 part* ‘echo fic3‘`
- Subs. commandes `ls -l ficl fic2 part* fic3`
- Séparation des paramètres
- Génération, si `part*=parta partb`
  ⇒ `ls -l ficl fic2 parta partb fic3`

Ordre de Substitutions de Commandes

Exemple:

```
1 set a=`ls ex*`
2 echo $a
3 echo "positionnel " $*
```

Exemple:

```
1 echo "votre repertoire est `pwd`."
2 echo  "Vous travailler sur la machine `uname -n`."
```

Résultat d'une commande

- Toute commande doit fournir un état de son fonctionnement.
- La convention sous Unix est de retourner:
  - 0 si tout est ok.
  - une valeur comprise entre 1..255 si problème, et le code identifie le problème.
- le code de retour d'une commande (i.e. script) est donnée par la commande `exit <no>`
- Shell utilise cette valeur pour enchainer les commandes
- Rappel:
  - `<cmd1> || <cmd2>`
  - `<cmd1> && <cmd2>`.
- mais aussi dans les structures de contrôles comme `if, while`, étudiés plus loin.

Commande **expr**

Commande qui fournir la capacité de manipulations et de comparaisons de chaînes ou d'expressions arithmétiques.

Définition

```
expr <expression arithmétique>
```

- Donc, de base le langage Shell ne sais pas calculer des expressions arithmetiques !
  - C'est le cas, car c'est un langage d'orchestration de commandes!
  - Tout peut se compenser par de nouvelles commandes !
- `echo ‘expr 2 + 3‘`

Structure de Contrôles: **if**

- Instructions conditionnelles

Syntaxe

```
if <condition>
  then
  <commandes>
 [else
<commandes> ]
fi
```

- Remarque ligne `If <condition> ; then`

Exemple:

```
then
   rm $1
else
echo $1 inexistant
fi
```

Structure de Contrôles: **if**

Exemple:

```
1  echo "Votre age? "
2  read age
3  echo "Vous avez $age an(
       s)"
4  if [ "$age" -lt 0 \
5   -o "$age" -gt 100  ]
6  then
7    echo "Age incorecte !"
8    exit
9  fi
10
11 if [ $age -lt 8 ]
12 then
13  echo "oh un bebe !"
14 else
15  if [ $age -gt 79 ]
16  then
17   echo "oh, un vieux"
18  else
19   echo "Profitez en !"
20  fi
21 fi
```

Structure de Contrôles: **if**

Exemple:

```
1  for file in *          # Empty word list
2  do
3  if [ -f $file  -a   ! -x $file ]
4  then
5  echo $file n'est pas executable
6  fi
7  done
```

Erreurs d'Espaces: Cas d'école

- Erreur 1: Les espaces dans les chaînes
  - `a="bon jour"`
  - `b="ok"`
  - `if test $a = $b`
  - `...`

    $\Rightarrow$ `if test bon jour = ok`
    - `if test "$a" = "$b"`
    - `If [ "$a" = "$b" ]`
- Erreur 2: Les espaces
  - `if test $a= $b`
  - test equivaux: `if [ $a = $b ]`
  - `if [ -f $fic ]` equivaux `if test {f $fic`

Erreurs Fréquentes: Cas d'école

- `if test $1 = toto`
  - Pourquoi ça peut ne pas fonctionner ?
  - Explication
    - `if test = toto`
    - `if test "$1" = toto`

Structure de Contrôles: **case**

- Commande UNIX construite pour fournir les capacités de manipulation de chaînes ou d'expressions arithmétiques.

Syntaxe:

```
case <chaine> in
 <motifi>)
    <commande 1>.. ;;
 <motif2>)
   <commande2>..;;
 <motifn>)
   <commande n>;;
 *)
  <cmd defaut>;;
esac
```

Exemple:

```
case $TERM in
hp*) echo terminal hp
          echo bon ;;
[oO]*)          echo Oui ;;
vt*|Vt*) echo terminal dec;;
*) echo terminal inconnu ;;
esac
```

Structure de Contrôles: **case**

Exemple:

```
1  echo -n "est ce correcte? [O/N] "
2  read answer
3  case "$answer" in
4  [Oo]*)
5  echo "En etes vous sur ?" ;;
6  [Nn]*)
7  echo "Ne soyez pas negatif !" ;;
8  *)
9  echo "Pas d'avis? "
10 ;;
11 esac
```

Instructions répétitives **for**

Syntaxe

```
for <variable> in <liste>
do
  <commandes>
done
```

Exemple:

```
for i in part*
  do
   mv $i $i_old
  done
```

Structure de Contrôles: **expr, if, case**

Exemple:

```
1  for name in a b c d
2  do
3  echo "bien $name"
4  done
5
6  for name in $* or for name in $@
7  do
8  echo Hi $name
9  done
```

Instructions répétitives: **while**

Syntaxe

```
while <condition>
do
  <commandes>
done
```

Exemple

```
until
do
sleep 5
done
```

Syntaxe

```
do
  <commandes>
done
until <condition>
```

Exemple:

```
1 num=0   #Initialize num
2 while [ $num -lt 10 ] #
      Test num with test
      command
3 do
4 echo $num
5 num=`expr $num + 1` #
      Increment num
6 done
```

Instructions répétitives: **while**

Exemple:

```
1  #!/bin/bash
2
3  num=0    #Initialize num
4  while  [ $num -lt 10 ] # Test
       num with test command
5  do
6  echo $num
7  num=`expr  $num + 1` #
       Increment num
8  if [ $num -eq 4 ]
9  then echo "Vrai"
10   break
11 fi
12 echo "Bla bla"
13 done
```

- `break [n]` permet de sortie de il boucles englobantes.
- `continue [n]` reprend la nième boucle englobante à l'itération suivante.

Instructions répétitives: **while**, Cas d'école

- Lecture d'un fichier

Exemple

```
while read i
do
        echo $i
done < fichier.txt
```

Fonctions

- Permet de découper un programme en unité "logique"
- Exemples:

Exemple

```
bon  () {
 echo "Bonjour"
}
```

Fonctions: Résultat

```
$ bon
Bonjour
$ bon
Bonjour
```

Fonctions

- Une fonction peut aussi être vue comme un script
  - Si un script s'appel S et une fonction porte le même nom, lorsqu'on invoque S, c'est la fonction qui prime.
  - A l'aide des fonctions, on peut surcharger des commandes de base.
  - Exemple: Redéfinir la fonction rm avec sauvegarde dans une poubelle

Paramètre des fonctions

- Chaque fonction peut recevoir des paramètres
  - Paramètres positionnels $1, $2, ...

Exemple

```
function bon() { echo "Bonjour $1, ca va ? "; }
$ bon John
$ Bonjour John, ca va ?
```

Paramètres d'une fonction

- Définition d'une fonction comme procedure:
  - `function bon() { echo "Bonjour $1, ca va $2 ? "; }`
- Appel de la fonction
  - `bon toto titi`
- Ne pas confondre $1 du script et $1 d'une fonction

Exemple du script paramScriptFct

```
echo "Script = $1"
function bon () { echo "Fonction = $1" }
bon toto
```

Résultat

```
$ paramScriptFct jour
Script = jour
Fonction = toto
```

- Prédominance du paramètre de la fonction sur celui du script

Fonctions

Exemple:

```
1  table ()
2  {
3  echo "Table de $1"
4  num=0   #Initialize num
5  while [ $num -lt 10 ] # Test num with test command
6  do
7   res=`expr $1 \* $num`
8   echo $1 " fois " $num " = $res"
9   num=`expr $num + 1` #Increment num
10 done
11 }
12 table 4
13 table 6
```

Argument variable et retour d'une valeur

- Passage d'une valeur par variable globale

```
$ r=1
$ function plus_un() { r=‘expr $r + 1‘; }
$ plus_un
$ echo $r
```

L'instruction **return**

- Équivalent à un exit du Shell.
  - soit une valeur de 0 à 255
  - ne peut retourner une chaine de caractères

Exemple

```
function plus_un() { return $1+1 }
r=1
plus_un $r # b=plus_un      ne marche pas
echo $?, $r
```

- $? contient la valeur du `return`, soit une valeur de 0 à 255

Exemple

```
function maFct() { return 0 }
maFct
if [ "$?" == 0 ]
then ...
```

Comment Retourner une Chaîne de Caractères ?

- Une fonctionne peut que retourner une valeur de 0 à 255
- Plusieurs solutions possibles
  - echo d'une chaine

Exemple echo d'une chaine

```
function maFct() { echo "maValeur" }
retval=$( maFct )
if [ "$retval" == "true" ]
then ...
```

  - Partage d'une variable

Exemple

```
varPartage=""
function maFct() { varPartage="maValeur" }
maFct
if [ "$varPartage" == "true" ]
then ...
```

Exemple de surcharge de la fonction **rm**

Redefinition de la fonction:

```
function rm() {
 if [ "$PWD" == "/storeU/poubelle"  ]; then
   echo "Poubelle Vidée"
   $(which rm) $*
 else
   for word in "$@"
   do mv -n "$word" /storeU/poubelle/; done
   echo "-> /storeU/poubelle"
 fi
}
```

En Résume du Langage de Script

- Un langage complet pour orchestrer les commandes
- Souvenez-vous, une commande par ligne, sinon il faut ;
- Attention aux espaces dans les instructions car interprète chaque instruction comme une commande. Chaque argument doit être séparé par des espaces !
- Souvenez-vous qu'un ensemble de mot est considéré comme un argument si on utilise les " ou '

SECTION: Le Monde du Shell

Efficacité Sous Shell

- Certes Shell demande de taper des commandes. Mais on devient très efficace si l'on maîtrise:
  - L'utilisation des *alias*
  - L'écriture de scripts
  - L'utilisation des flèches du clavier. Rappel les précédentes commandes.
  - L'utilisation de "l'historique des commandes". Rappel d'anciennes commandes.

Historique des Commandes

- Objectif: Rappeler une commande antérieure ou voir l'historique.
- Une ligne constitue un événement dans la liste "historique" et ces événements sont numérotés.
  - Pour voir la liste, tapez history.
  - Les commandes utiles:
    - !! Remplace l'événement précédent (rappel la commande précédente).
    - ! n Remplace l'événement numéro n.
    - !str Remplace l'événement le plus récent contenant str.
    - Exemple: !cc la dernière commande débutant par cc

.

Les 'alias'

- Alias: Utilisé pour abréger la longueur des commandes et personnaliser des commandes selon l'habitude de l'utilisateur.
  - La commande interne alias:
    - sans arguments, il permet d'afficher la liste des alias. Le premier mot d'une commande est testé contre cette liste pour faire une substitution d'alias.
    - Exemple 1: `alias ff='find / -name * -print'`
    - Exemple 2: `alias ll='ls -l'`
- Supprimer la définition d'un alias: `unalias`

Développer sous Unix

- Ecrire une nouvelle application:
  - Utiliser un langage comme Java et un IDE pour écrire complètement une application.
  - Ecrire que ce qui est nécessaire (Shell, VBA, etc.)
- Sous Shell, c'est :
  - Utiliser des commandes existants
    - Chaque commande est très générique.
    - Il faut très bien uniquement sa fonction.
  - Ecrire les commandes manquants (éventuellement)
    - Utilisation d'un langage approprié (ex. C)
  - Ecrire le script qui combine l'ensemble de ces commandes

Développer sous Unix

- Tout faire en Shell ?
  - Shell vise l'intégration de commandes: Lancer des commandes avec un type de synchronisation.
    - Hors de ce cadre, possible, mais mieux vaut utiliser des outils spécialisés
    - Exemple: awk pour filtrer du texte
    - Exemple: sed manipule des flux de textes
  - Existe d'autres langages
    - Exemple: awk, Tcl, Perl, Python, Ruby, ...

Inclusion d'Autres Interpreteurs dans un Script

- Il existe beaucoup d'autres interpréteurs: *awk*, *sed*, *gnuplot*, ...
- Le langage doit être interprété
- L'inclusion permet de mixer les possibilités

Exemple

```
echo "bonjour"
export A='toto'
python -  << FINI
from pdb import set_trace
reserve=out=''
...
print out
FINI
```

Outils pour Développer sous Unix

- Les compilateurs
  - Langage de programmation: gcc, pascal, java, ...
  - De textes: Latex, troff etc.
  - De courbe: gnuplot, etc.
- De gestions d'un developpement
  - git, cvs, svn,... synchronisation de plusieurs version de sources
    - permet de developpement collaboratif
  - ddd/gdb/ ... débugger
  - profile
  - make
    - Se base sur une description dans Makefile
  - ...

Makefile

- Idée: *Commande*$_i$ qui transforme des données pour la *Commande*$_{i+1}$.
  - Donc un flot de transformations.
  - Un document passe par une succession de transformations.
- Comment se souvenir d'appliqué une chaîne de commande à chaque modification du document source ?
- L'outil `make` fait ca:
  - Le principe: Compare les dates du document source et du document cible.
  - Si la date du document est postérieur au document cible, alors appliquer la ou les commandes pour produire le document cible.

Syntaxe

```
<cible>: <depend>
  <commande à appliquer>
```

Exemple de Makefile

```
all:        exam.ps lecteur
exam.dvi: exam.tex
  latex exam
  dvidvi -m "2:0,1(14.85cm,0cm)" exam.dvi tmp.dvi
  mv -f tmp.dvi exam.dvi
exam.ps: exam.dvi
  dvips -t landscape -f exam.dvi > exam.ps
lecteur:        lecteur.o util.o
  $(CC) $(CFLAGS) -o lecteur lecteur.o util.o $(LFLAGS)
clean:
  cleantex -a exam
  rm -f a.out core *.o tmp.dvi *.ps lecteur serveur posteur
```

Makefile

- en tapant:
  - en tapant `make`, il recherche par défaut un fichier appelé Makefile
  - il applique la première cible
- on peut déclencher une cible en tapant: `make <nom de la cible>`
- Sur le principe de `make`, il existe de nombreux nouveaux outils

La communication sous Unix

- Unix/connectivité
- Internet
  - Réseau temps réel (ou connecté)
    - Offre de nombreux services
    - Composé de milliers de sous réseau
  - Historique
    - Initialement Universitaire aux Etats-Unis
    - Actuellement planétaire
  - TCP/IP
    - Protocole: packet-switched readline
    - Noeud à Noeud
    - Des routeurs

Services Interactifs

- FTP Transfert de fichiers interactif
- Gopher Banques d'informations textuelles distribuées.
- IRC Discussion de type CB distribuée
- MUD Jeux multi-utilisateurs distribués.
- NFS Partage de fichiers.
- NNTP Lecture de news sur serveur distant,
- POP Lecture de mail sur serveur distant.
- TALK Discussion interactive.
- TELNET Appel de système distant.
- WWW Services multimédia interactifs distribués

Services non interactifs

- mail Courrier électronique privé. Permet d'échanger des messages avec des correspondants du monde entier.
- news Forums de discussion. Permet d'échanger des points de vue dans des conférences à thèmes spécifiques avec d'autres utilisateurs.
- uucp Transmission de fichiers.
- mailservers Serveurs d'informations via courrier électronique
  - Exemple: faxserv@aphanet.ch permet d'envoyer des fax par courrier électronique.
- ...

Mail

- Le système mail sert à l'échange de courrier électronique privé entre correspondants connectés au réseau ALPHANET, Internet, CBMNET. FidoNet, Compu$erve, X.400, UUCP et divers autres réseaux.
  - L'interface utilisateur comprend les commandes mail et elm. Il est également possible d'utiliser ce service pour créer des listes de distributions permettant, d'atteindre des personnes aux intérêts ciblés (mailing list).
  - `mail`
    - Interface ligne. Très simplifié
    - Se termine par . en début de ligne.
  - `elm`:
    - Interface VT100
- Transfert de mail sous le standard multi-plate-forme MINE

Spécification d'adresses

- Le routage mail sur ALPHANET est très simple. Toute adresse soumise au système de message doit être de la forme générale suivante: `user@computer.domaine1.domain2.domam3.topdomain`
  - `laurent.yeh@prism.uvsq.fr`
  - Le nombre de sous-domaines peut être quelconque.
- TOP domaines
  - Les TOP domaines sont les domaines terminant une adresse mail. Il existe énormément de TOP domaines, dont par exemple:
    - edu Education aux Etats-Unis (universités).
    - gov Gouvernement des Etats-Unis.
    - com Organisations à, but lucratif ou général.
    - org Organisations à but non lucratif.
    - net Organisations de réseaux.
    - ch, us, fr, ..

News (forums de discussions)

- Messagerie électronique publique
  - Forum d'échanges qui sont regroupés en newsgroups
    - Exemple: `comp.sys.lang.shell`
    - Plus de 1GB par jours
    - Parle de serveur NNTP
  - Exemple de catégories
    - Misc sujets divers
    - Comp forums concernant l'informatique
    - sci sciences
    - ...
  - `tin`
  - `knews`, logiciel commercial, www

Communication à distance

- `ping` permet de savoir si un site distant est atteignable et en combien de temps.
- `traceroute` trace le parcours de paquets IP jusqu'à un site distant.
- `netstat` permet de connaître les session TCP ouvertes sur la machine courante.
- `whois` permet d'obtenir des informations sur des domaines internet.

Communication entre utilisateurs

- `finger` permet de connaître les utilisateurs connectés actuellement au système. Exemple:
- `who` ermet de connaître les utilisateurs en ligne.
- `write <nom de login>` permet de dialoguer en temps réel avec un autre utilisateur en session si celui-ci accepte le message, (mesg n: refus, mesg y : acceptation).
- `talk <nom de login>`

Transférer d'un compte vers un autre

- Généralement sur des machines distantes
- `ftp` (File Transfert Protocol)
  - Permet d'échanger des fichiers entre sites distants
  - De nombreux versions ftptool,
  - Exemples de sites (ftp.ibp.fr, ftp.uvsq.fr, . . .)
- `xarchie`
  - Base de données mondiale pour rechercher des fichiers
  - `telnet nic.switch.ch`
- `scp <source> <destination>`
  - Remplace rcp
  - Exemple: scp fic.txt yeh@darwin.uvsq.fr:/home/yeh/arc

Synchronisation d'arborescences de fichiers

- But: synchroniser une arborescence A avec une arborescence B de fichiers
  - Ne recopie vers B que les fichiers de A qui ont été modifies
  - Basé sur une comparaison de dates de modifications

Exemple

```
rsync -av rep1/  yeh@darwin.uvsq.fr:rep2/
```

- S'appuie sur `ssh`, donc securisé

Session à distance

- Commandes permettant d'ouvrir des sessions sur des systèmes distants (réseau local ou réseau Internet).
- `ssh <nom de machine>` comme telnet, ouvre une session. Mais plus sur. A l'heure actuelle ssh.
  - permet d'exécuter des commandes shell à distance.
  - Remplace: rlogin, telnet, rsh

Exemple

```
ssh yeh@darwin.isty.uvsq.fr
```

  - Possibilité d'authentification par clé RSA/DSA

X-Windows (X11 X-Windows System)

- Développe au MIT 1984
- Objectifs:
  - Rend indépendant plusieurs aspects d'un système de fenêtrage. Indépendant:
    - Matériel
    - Du Système d'Eploitation
    - De la localisation
    - Du "look-and-feel"

Exemple demo fiat en cours

```
Affichage de  "clients" en provenance de plusieurs machines sur un écran un
```

Architecture X-Windows

- Client/Serveur
  - Exemple: Machine serveur puissant servant plusieurs clients. Affichage chaque résultat sur l'ordinateur de bureau d'un client.
- Serveur
  - Exécute un noyau (ex. XFree86)
  - Plusieurs instances exécutées
  - Communication codée avec un client
  - Désigné par la variable d'environnement DISPLAY
    - Exemple: DISPLAY=darwin.uvsq.fr:0.0
- Client
  - Window-managers (gestionnaire d'interface)
    - Gère l'aspect pour l'utilisateur
    - Mode d'interaction (ex, aspect des fenêtres)

Architecture X-Windows

- Copier Coler à la X-Windows
  - Double bouton de la souris ou troisième bouton.
  - S'effectue au niveau du client X-Windows
- Travailler sur une session à distance avec XWindows
  - SSH -X yeh@darwin.uvsq.fr
  - Si ca ne marche pas, mauvaise configuration, voir solution du transparent suivant

Travailler en X-Windows sur une machine distante non sécurisée

- Normalement, ssh -X yeh@persee.prism.uvsq.fr permet de travailler en X windows à distance
- Si ce n'est pas le cas, voici un déroulé (non securisté)

Exemple

```
$ xhost +persee.prism.uvsq.fr  # J'authorise persee à affiche
$ ssh yeh@persee.prism.uvsq.fr # Je suis sur le serveur
export DISPLAY=192.22.33.11:0.0
# Je tape Ctrl-D pour quiter la session distante
gedit # J'essaye !
# Normalement, la fenetre est ouverte sur ma machine
```

- Les programmes s'executent sur le serveur, mais l'affichage est redirigé vers $DISPLAY.

Super Utilisateur

- Pour protéger l'utilisateur lambda contre la casse du système unix, il existe deux niveaux d'exécution: Normal et Super utilisateur
  - Normal qui ne peut tout faire
  - Super utilisateur (root) qui peut administrer le système et donc le casser !

Pour être super utilisateur, il faut:

1. appartenir au groupe `sudo`.
   Exemple: `$ sudo usermod -a -G sudo yeh`
2. simplement précéder une commande par sudo
   Exemple: `$ sudo mkdir /monroot`
3. taper Exemple: `$ sudo su` pour continuer une session en super utilisateur

Conclusion

- Communiquer et utiliser un environnement de fenêtrage est possible dans le monde Unix.:
- C'est:
  - simple et naturel de travailler à distance !
  - avoir un environnement bureautique équivalent à MacOS ou Windows
- Un système:
  - très solide qui n'a pas besoin d'un rachat d'une machine plus puissante tout les X années ! Même de très vielles machines peuvent booster !
  - qui ne dégrade pas ses performances au cours du temps
  - où tout est gratuit
  - pour développer ...

SECTION: Annexes

printf

Définition

```
printf <chaîne format> <arg>...
```

- Plus puissant que echo, permet de formater les arguments
- les types connus: s, c, f, d, x

Exemple

```
$ printf "[%s]\n" "Laurent Y"
[Laurent Y]
$ printf "[%20s]\n" "Laurent Y"
[           Laurent Y]
$ printf "[%.2f]\n" "28,987665"
[28,99]
$ printf "[%8.2f]\n" "28,987665"
[   28,99]
$ printf "[%x]\n" "28"
[1c]
```

printf

Définition

```
printf <chaîne format> <arg>...
```

- placer des caractères de contrôles: \n, \t

Exemple

```
$ printf "[%s]\tenseigne\t%s\n" "Laurent Y" "Intro Unix"
[Laurent Y] enseigne Intro Unix
```

- répétition du format jusqu'à épuisement des arguments

Exemple

```
$ printf "%s\t%s\n" "1" "2 3" "4" "5"
1       2 3
4       5
```

Commande expr pour expressions régulières

Définition

`expr <chaîne> : <expression-régulière>`

- teste si la chaîne correspond à l'expression régulière fournie.
- Exemple: `if expr "$1" : [0-9]+`
- Exemple:

```
$ string="yeh laurent 2 16.5 Algo"
$ expr "$string" : '.*[0-9].[0-9]'
18
$ expr "$string" : '.* \([0-9]*.[0-9]\)'
16.5
```

Calculer sur une Colonne

Le probleme

Comment prendre les valeurs d'une colonne. Exemple de l'exercice donné en cours sur le disk free d'un étudiant

- Le plus naturel est d'utiliser `sed`
- Le plus simple est d'utiliser la commande `cut`.
- `awk` commande qui permet de faire des traitements encore plus complexes

Solution 1: Basé sur une commande

```
for i in ‘df -k | awk ’{print $3}’‘
do echo $i
done
```

Utilisation d'une commande

Solution 2: Utilisation du Set

```
df -k | while read line
  do  set $line
    echo $2
  done
```

Utilisation d'un tableau

- Bash sait manipuler des tableaux
- Exemple: `$ t=(un deux trois)`
- Exemple: `$ echo ${t[1]}`

Solution 3: Basé sur les tableaux

```
df -k |
while read line ;
  do
    lineCols=( $line ) ; # Split suivant espace
    echo "$lineCols[1]"
  done
```

Tracer une exécution

- Mise au point d'un programme bash
- Peut se faire par des `echo`
- mais plus simple en utilisant l'option `-x`

Exemple du fichier split_lines

```
df -k | while read line ; do
    lineCols=( $line ) ;
    echo "$lineCols[1]"
  done
```

Résultat

```
$ bash -x split_lines
+ df -k
+ read line
+ lineCols=($line)
+ echo de
```

Tracer une exécution

- On peut tracer une sous partie grace à `set -x` pour activer et `set +x pour desactiver`

Exemple du fichier split_lines

```
df -k | while read line ; do
  set -x
    lineCols=( $line ) ;   echo "$lineCols[1]"
  set +x
  done
```

Résultat

```
$ bash -x split_lines
+ df -k
+ read line
+ lineCols=($line)
+ echo de
```

Conclusion

- UNIX shell est fondamental pour travailler avec un système.
  - Beaucoup à découvrir
- Beaucoup d'extensions ou d'améliorations: csh, tcsh, ksh, bash,
- Forte adaptation au noyau UNIX et à la production de programmes C.

Bibliographie

- Christophe Blaess, Langage de scripts sous UNIX, Eyrolles.
- W. Richard Stevens, Advanced programming in the UNIX Environnement, Addison-Wesley Profession al Computing Séries, 1992.