Cours de Système Introduction

Bertrand Le cun[1], Emmanuel Hyon[2] et J.-F. Pradat-Peyre[2]

Université Paris Nanterre
[1]prenom.nom@google.fr [2]prenom.nom@parisnantere.fr

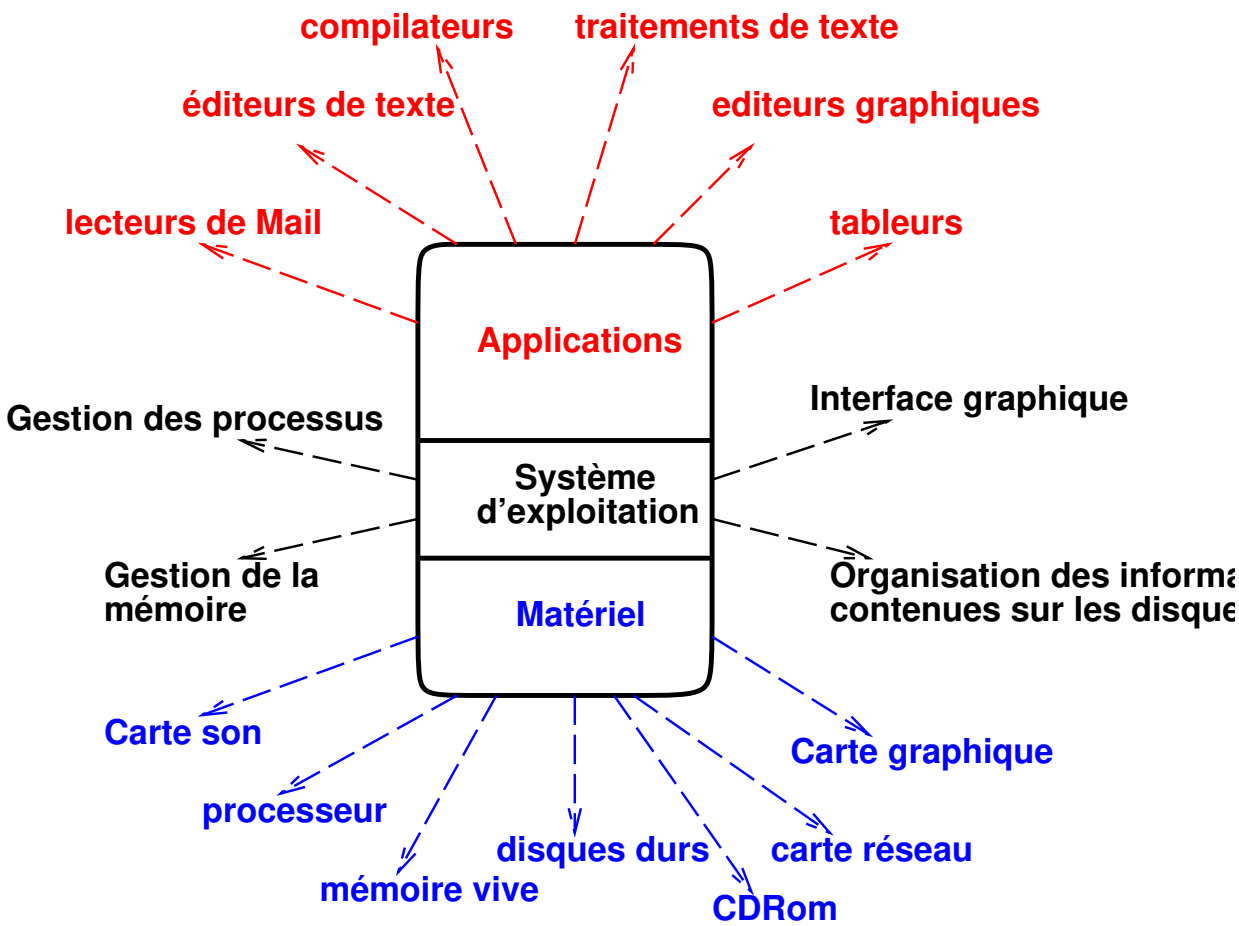
5 janvier 2020

Buts suivis

Intérêts

- Comprendre les interactions entre logiciel et matériel
- Comprendre les mécanismes internes
  - Notion de processus,
  - Système de gestion de fichiers,
  - Gestion mémoire,
  - Communication inter processus
- Comprendre la communication inter machines (réseaux)

Introduction



Définition

Un S.E.

Un système d'exploitation (Operating System O.S.) est un intermédiaire indispensable entre un utilisateur, les programmes applicatifs, et le matériel de l'ordinateur

Il permet

- l'utilisation correcte, commode et efficace du matériel
- la bonne gestion de toutes les ressources de l'ordinateur

Le système

Rôles

- Gestion des ressources
  - Rend uniforme l'accès aux différents matériels :
    RAM, Carte graphique, CD-Rom, Disque Dur, Ports : USB, Série, Firewire etc. . .
  - Rend uniforme la gestion de la mémoire
  - Définit le système de gestion de fichiers (SGF)
- Gestion du déroulement des programmes (ordonnanceur)
  Définit la notion de processus
- Gestion de la communication
  - Définit la notion d'utilisateur (contrôle d'accès).
  - Permet (ou non) la communication entre processus
  - Permet (ou non) la communication inter machines.

Historique

1953 IBM 680

1965 Multics : ancêtre d'unix.

1969 Unix (AT&T) produit universitaire

1981 MS Dos (disk operating system), premier produit Microsoft, système pour le PC-XT d'IBM

1982 QNX système temps réels pour automates industriels

1984 Macintosh, Interface graphique à la souris, inventé chez Rank-Xerox

1991 Linux, Une implémentation d'Unix

1995 Windows 95 (version améliorée de Windows 3)

1998 Symbian OS pour téléphone mobile.

2002 Windows XP (basée NT non basée sur DOS)

2008 Android (OS pour mobile basé sur linux)

2009 Windows 7 (approche modulaire)

Les systèmes

Système Propriétaire : Windows, Apple (*Darwin* Free mais *aqua* et quartz non free)

- Beaucoup d'aspects sont secrets, propriétaires...
- Les changements ou évolutions sont imposés
- Le paramétrage est limité
- Fonctionnalités cachées

Système Ouvert : Unix, Linux

- Système ouvert et système en logiciel Libre !
- Tout est public,
- Dans le cas du libre, les changements et évolutions sont discutés et votés
- Évolution, toutes les interfaces sont connues, bien souvent, les sources sont publiques

Unix

1971 V1 Unix Assembleur sur PDP11

1973 V4 Réecrit en C

1975 V6 Bell Labs, accessible à tous, naissance des version BSD.

1978 première version commerciale, la V7

1983 2 versions : SysV (AT& T) et BSD (Berkeley University).

1996 Xopen, Norme POSIX

Quelques Unix

- Commerciaux : AIX (IBM), SCO-Unix, Xenix (Microsoft), HP-UX (HP) et Solaris(SUN) ...
- Libre : NetBSD, FreeBSD, OpenBSD, Linux.
- Minix, un autre système proposé par Tannenbaum.

Linux est le plus connu

Linux

Historique

1991 Linus Torvalds envoie un mail sur la liste comp.os.minix

1994 Réorganisation totale du noyau (Kernel).

2000 Engouement de sociétés, de personnes, d'associations pour linux (remplace progressivement unix)

Qu'appelle-t-on Linux ?

- Linux est juste le noyau (qui gère les ressources).
- D'autres (sociétés, associations) s'occupent de proposer des distributions : RedHat (Fedora), Suse, Debian, Ubuntu (Canonical), Mandriva...

  voir http ://www.linux.org/
- Beaucoup de sociétés/associations proposent des logiciels gratuits pour linux

Système et ressources

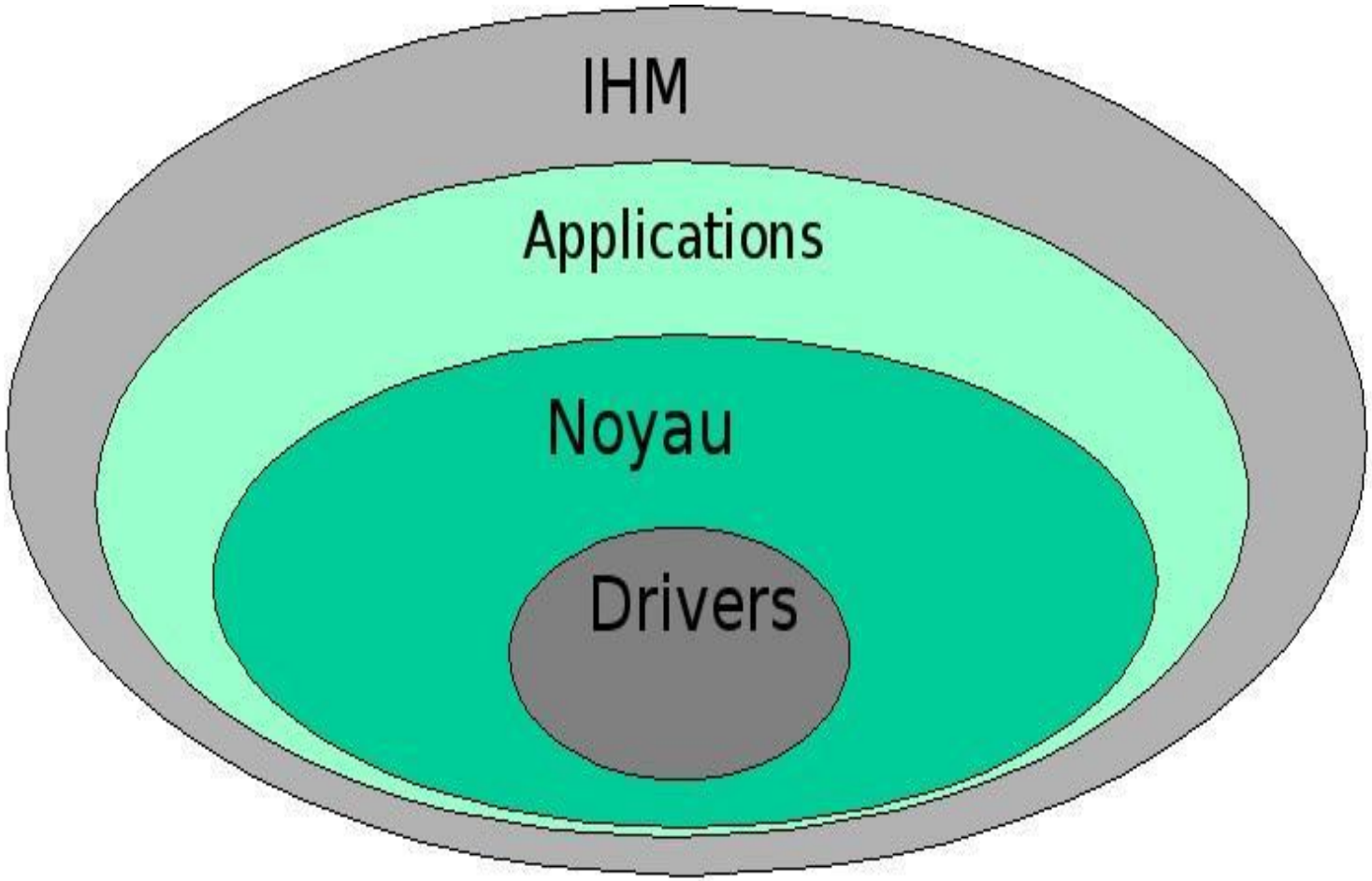
Objectif principal : Gestion des ressources

efficacité	utilisation maximale des ressources
équité	pas de programmes en attente indéfinie
cohérence	des données entre deux accès consécutifs
protection	des données contre des accès interdits

Caractéristiques d'un OS

- Mono-Tâche/Multi-Tâche
- Préemptif/non préemptif
- Mono-utilisateur/Multi-utilisateurs
- Mémoire : Avec ou sans protection mémoire Avec ou sans swap (fichier d'échange)
- File system (Système de gestion de fichiers) journalisation, RAID, allocation, etc...

Différentes parties d'un S.E.



Rôle et fonctionnement

L'O.S. est passif

Réagit à des événements provenant :

- Du matériel (*interruption* matérielle) ;
- Du logiciel (*interruption* logicielle).

Les drivers (modules)

Architecture dépendance :
Avant le constructeur de matériel fournissait son propre S.E. écrit en langage machine.
Besoin de Portabilité : Abstraction matérielle

- Écriture avec langage de programmation.
- Séparation de ce qui dépend de l'architecture : le module.
  - Programme qui peut intervenir sur un périphérique physique.
  - Interface (codifiée) de dialogue avec l'O.S.

Rôle et fonctionnement (suite)

Le noyau (kernel)

- Assure la répartition des ressources sans interventions externes (accès à tous les programmes).
- Lancé au démarrage et chargé en mémoire.
- Communication avec le noyau par des *appels systèmes*.

Les applications :

- un certain nombre de bibliothèques standard (notamment celles du C)
- utilitaires liés à l'administration du système.

L'interface utilisateur

Permettre à l'utilisateur d'interagir avec le S.E.

- Transmission d'ordres de l'utilisateur au S.E.
- Accès aux ressources.

L'apport d'Unix

Résumé

- L'unicité d'arborescence disque
- Accès au hardware
  - Composante matérielle et/ou logicielle représentée par un "fichier spécial"
  - Les fichiers spéciaux dans /dev :
    /dev/mem /dev/video /dev/cdrom /dev/audio ...

Langage de commandes

Tout SE propose des possibilités pour manipuler les fichiers et les répertoires et pour lancer des processus.

- Sous MacOS, Windows **tout à la souris**
  - click-click sur des icônes,
  - click sur des menus déroulants.
- Sous Unix, à l'origine **tout au clavier :**
  - interpréteur de commandes le shell
  - des commandes à la pelle !
    (cd , ls, rm, cp, mv, rm, etc)

Le Shell

Principes

- programme qui attend que vous tapiez des commandes,
- le prompt ou “invite ” indique cette attente.
- Interpréteur ligne une commande sur une ligne, en appuyant sur entrée (caractère fin de ligne) la commande sera exécutée,
- Beaucoup de petites choses facilitent l’utilisation du shell.
- Propose aussi un véritable langage de programmation les scripts

Le Shell bases

Notion de répertoire courant

- Comme tout processus unix, le shell a un répertoire courant.
- C'est à partir de ce répertoire que ce fait le nommage des fichiers ou répertoires.

Système de nommage

- répertoire courant "."
- répertoire père ".."
- séparateur de répertoire "/"
- répertoire HOME "~"
- répertoire racine (root), on commence par "/"

Système de nommage exemple

Référence absolue

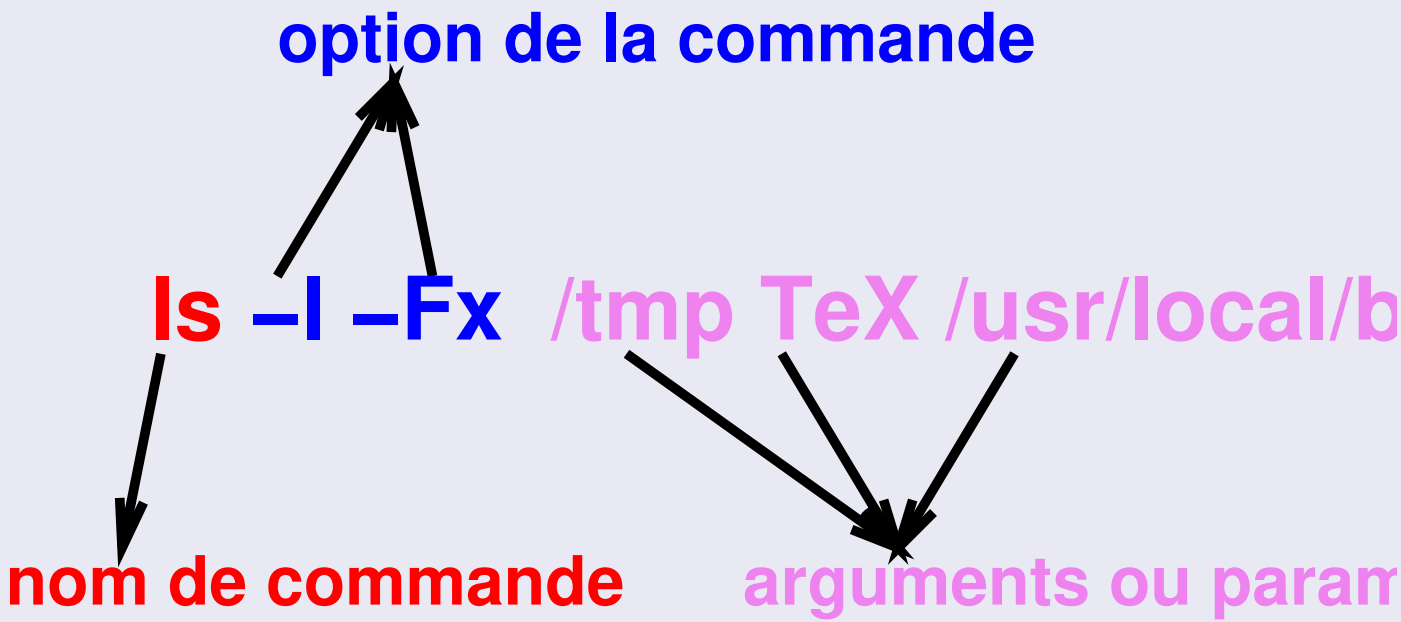
- donnée à partir de la racine :
  Exemple : `cd /home/users/info/moi/TeX/Publi`
- donnée à partir d'un HOME :
  Exemple : `ls ~/TeX/Publi` ou `ls ~moi/TeX/Publi`

Référence relative

- donnée à partir du répertoire courant.
  `cd TeX/Publi` ou `cd ./TeX/Publi`
  `cd ../user/TeX`
  `cd ../../toto/Work/latex`

Forme générale d'une commande sous unix

Commande ligne



une commande tient sur une ligne.

Les commandes

Commandes usuelles

- `cd répertoire` change le répertoire courant du shell,
- `cd` ou pwd affiche le répertoire courant du shell,
- `ls` ou ls <liste de fichiers ou catalogue> affiche le nom des fichiers et catalogues du répertoire courant.
- `cp f1 f2` copie du fichier f1 dans f2.
- `cp f1 f2 f3 ... d1` copie des fichiers f1 f2 f3 ... dans le catalogue d1.

Les commandes

Commandes usuelles

- `rm f1 f2 f3` suppression des fichiers f1 f2 f3
- `mv f1 f3` changement de nom du fichier f1 en f2.
- `mv f1 f2 f3 ... d1` changement de location des fichier f1 f2 f3 ... dans le répertoire d1.
- `mkdir d1` crée le répertoire d1.
- `rmdir d1` détruit le répertoire d1 (il doit être vide).
- `file f1 f2 f3` affiche le type des fichiers f1 f2 f3.
- `cat f1 f2 f3` affiche le contenu des fichiers f1 f2 f3.
- `more f1 f2 f3` affiche page à page le contenu des fichiers f1 f2 f3.

Les commandes et leur manuel

la commande man

- `man <commande>` donne le manuel de la commande en paramètre
- `man -k <mot>` donne la liste des pages man contenant le <mot>
- Les pages de man sont classées par section.
  - Section 1 : commandes utilisateurs
  - Section 2 : appels systèmes
  - Section 3 : fonctions de bibliothèques
  - Section 4 : *devices*
  - Section 5 : format de fichiers spécifiques
  - Section 6 : jeux
  - Section 7 : divers
  - Section 8 : commandes d'administration système

Par exemple, getopt existe dans 1 et 3

man 1 getopt

man 3 getopt

Expansion des noms de fichiers

Nommer plusieurs fichiers de noms différents

ls *.tex Affiche les fichiers du répertoire courant ayant l'extension '.tex'.

ls T*.tex Affiche les fichiers du répertoire courant commençant par 'T' et ayant l'extension '.tex'.

ls ?a*b*.tex Affiche les fichiers du répertoire courant ayant pour deuxième lettre un 'a', contenant un 'b' et ayant l'extension '.tex'.

ls ? ? ?.tex Affiche les fichiers du répertoire courant de trois caractères et ayant l'extension '.tex'.

D'autres sont aussi possibles voir le man du shell correspondant

Attention c'est le shell qui effectue les expansions de noms de fichier.
Toute commande peut admettre des motifs de noms de fichiers.

Exemple : echo *.sh

Compléter un nom de fichier

Complétion des noms de fichiers

- Expansion automatique des noms de fichiers
  Touche [TAB] étend automatiquement les noms de fichiers (et commandes) correspondant aux caractères tapés.

```
<Mo-lecun-4-68- Unix -> ls
Net.F        disk.F       main.dvi    main.tex
OS.F         display.F    main.log    psfig.tex
cmdfrm.F     main.aux     main.ps
<Mo-lecun-4-69- Unix -> ls ma<TAB>
main.aux  main.dvi  main.log  main.ps   main.tex
<Mo-lecun-4-69- Unix -> ls main.
```

Voir les commandes déjà exécutées

Historique des commandes

- le shell maintient une liste des commandes que vous avez tapées.
- les flèches haut et bas permettent de naviguer dans cet historique.
- commande “history” affiche cette liste.
- commande “ !ma” ré-exécute la dernière commande dont le nom commence par ma
- commande “ !23” ré-exécute la commande numéro 23
- commande “ !-1” ré-exécute l’avant dernière commande

Modes d'exécution des commandes

Avant-plan

- Lorsqu'une commande est lancée par le shell, elle prend "la main" sur le shell.
- Le shell ne peut plus lire d'autres commandes au clavier
- Le shell est en attente que la commande se termine

Arrière-plan

- Possibilité de lancer une commande en arrière plan
- Ajouter un & à la fin de la commande
- Juste après avoir lancé la commande, le shell attend de nouvelles commandes

Redirections

Entrée et sortie et sortie erreur standard

- Toute application a
  - une entrée standard : pour lire des données
  - une sortie standard : pour afficher des données
  - une sortie erreur standard : pour afficher des erreurs d'exécution
- généralement ces entrées/sorties standards sont le terminal actif

Rediriger les E/S d'une application

- Ajouter `> fic` à la fin d'une commande permet de rediriger les affichages de la commande dans le fichier fic
- Ajouter `< fic` à la fin d'une commande permet de rediriger l'entrée standard de la commande à partir du fichier fic. La commande lira ces données dans ce fichier (et non au clavier).
- Ajouter `>& fic` à la fin d'une commande permet de rediriger les affichages d'erreurs de la commande dans le fichier fic

Le pipe

le pipe

- permet de rediriger la sortie d'une commande vers l'entrée d'une autre.
- Exemples :
  - `ls | wc -l` : compte le nombre de fichier du répertoire
  - `ls -l | more` : permet de voir les fichiers du dossier courant de manière paginée
  - `ls /bin /usr/bin /usr/local/bin | grep X | sort | more` : permet de voir de façon triée et paginée les fichiers contenus dans les dossiers /bin /usr/bin /usr/local/bin contenant un X dans le nom.

Composition de commandes

Séquence de commandes

- pour un shell : une ligne une commande
- ; composition : le caractère ; permet de mettre plusieurs commandes sur une même ligne.

```
cd ; ls -l
```

Composition conditionnelle

|| ou &&

sont les exécutions conditionnelles

- && : exécuter une commande que si la première est vraie,
- || : ne pas exécuter une commande si la première est vraie.

exemple : `cd /var/log/mail && ls -l`

exemple : `cd /var/log/mail || cd /var/mail`

Tout programme C est une “commande”

Tout programme exécutable (le binaire) peut être vu comme une commande. En effet la syntaxe du lancement du programme exécutable est similaire à une commande.

exemple : `./monprograme`

ou

`monprogramme`

Tout programme C est une commande (2)
Interaction avec environnement

Passage des arguments

Prototype :

```
main(int argc, char *argv[])
```

- `argc` est le nombre d'arguments (nom du programme compris),
- `*argv[]` est un tableau de chaînes de caractères :
  - `argv[0]` : adresse d'une chaîne contenant le nom du programme,
  - `argv[1]` : adresse du mot suivant.

Exemple

```
int main(int argc, char *argv[]) {
if (argc > 1) printf("Bonjour %s",argv[1]);
}
```

Manipulation des Shells

Les shells

Deux grandes Familles

- Les sh (sh, ksh, bash)
  Syntaxe sh surtout pour les instructions comme les boucles, les tests, etc.
- Les csh (csh, tcsh)
  Syntaxe csh essayant de ressembler au C

Les variables

Variables en sh

- Définition : nom_variable=donnée
- Utilisation : $nom_variable

```
blec@vanuatu:~$ toto=coucou
blec@vanuatu:~$ echo $toto
coucou
blec@vanuatu:~$
```

Variables et motif de fichiers

```
blec@vanuatu:~$ toto=G*
blec@vanuatu:~$ echo $toto
Games GNUstep
blec@vanuatu:~$
```

Les variables

Mettre plusieurs valeurs dans une variable

- On doit encadrer ces valeurs par des “” ou des ’ ’.
- Exemple sh :

```
blec@vanuatu:~$ toto="coucou il fait beau"
blec@vanuatu:~$ echo $toto
coucou il fait beau
blec@vanuatu:~$
```

Autre exemples

```
blec@vanuatu:~$ toto="G* P*"
blec@vanuatu:~$ echo $toto
Games GNUstep ParallelBB.pdf Perso PRISM
blec@vanuatu:~$
```

Différence guillemets et apostrophes

Exemple

```
:~$ toto=coucou
:~$ tutu="$toto moi"
:~$ echo $tutu
coucou moi
:~$ tata='$toto moi'
:~$ echo $tata
$toto moi
:~$
```

Visibilité d'une variable

Visibilité des variables simples

- une variable n'est visible que pour un shell, et que pour ce shell
- Nécessité d'avoir des variables "globales" dont les valeurs sont valides pour tous processus fils d'un processus

  Variables d'environnement
- Beaucoup sont utilisées par d'autres applications

Exemple

```
vanuatu:~> toto="la variable toto est definie"
vanuatu:~> bash
vanuatu:~> echo $toto
toto: Variable pas definie.
vanuatu:~>
```

Les variables d'environnement

sh

- Définition : export nom_variable=donnée
- Utilisation : $nom_variable
- Exemple

```
:~$ export toto="la variable toto sera definie"
:~$ echo $toto
la variable toto sera definie
:~$ bash
:~$ echo $toto
la variable toto sera definie
:~$
```

Variables d'environnement

Exemples

- PATH contient les répertoires où le shell ira chercher les commandes
- SHELL contient le shell actuellement utilisé
- LOGNAME contient le nom de login de l'utilisateur courant
- HOME contient le répertoire de l'utilisateur
- GROUP contient le groupe utilisateur
- HOST contient la machine actuelle utilisée
- DISPLAY indique quel terminal X utilisé

Groupement de commandes

Utilisation des ‘ ‘

Récupérer le résultat d’une commande

- on peut exécuter une commande et récupérer son résultat dans une variable

```
:~:> f=‘ls  | grep t‘
:~> echo $f
CourSystRes.tgz intro.aux intro.tex main.out main.tex
:~>
```

Attention il s’agit du signe ‘ (touche AltGr 7).

Les Alias

Mettre sous un mot un groupe de commandes

bash : `alias ll='ls -l'`

Il s’agit du signe apostrophe (touche 4)

Pas d’alias en sh. ksh et bash ont des alias mais sans paramètres.

Instructions des shells

Véritable langage de programmation

- Les shells sont des véritables langages de programmation mais orientés fichiers
- Généralement, une suite de commandes mise dans un fichier représentant un script
- Facilitent
  - lecture d'un répertoire
  - lecture des attributs de fichiers (droits, type, etc)
  - permet de lancer des commandes, programmes

Les fichiers scripts

Forme du fichier

- un fichier script est un fichier texte (ascii), une suite de caractères.
- la première ligne contient le binaire permettant d'interpréter le fichier.
- pour sh

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

- il existe sous unix d'autres langages de script : Perl, Tcl, Ruby...
- Équivalent du .bat sous windows

Lancement

- Le nom du script est une commande (si fichier est executable).
- Sinon c'est un paramètre de la commande `sh`

Les arguments des fichiers scripts

Des variables stockent les arguments du script

- $0 le nom du script $1 $2 $3 respectivement le premier, deuxième troisième argument. $* tous les arguments
- $# le nombre de paramêtres

```
:~$ cat truc.sh
#! /bin/sh
echo nombre de parametre $#
echo le script est $0
echo premier argument est $1
:~$ sh truc.sh titi
nombre de parametre 1
le script est truc.sh
premier argument est titi
:~$
```

Affichage à l'écran

La commande `echo` permet d'afficher des messages à l'écran

La saisie au clavier

Sh : `read name` permet la lecture d'une chaîne au clavier dans la variable `name`

Variables numériques

Les variables numériques ont un traitement à part. Pour interpréter une expression : `$(( expre ))`

```
:~$ a=$(( 4 + 5 ))
:~$ echo $a
9
:~$ echo $(( $a * 5 ))
45
:~$
```

Les Tests en sh

Forme

```
if liste ; then liste ; fi
if liste ; then liste ; elif liste ; then list ; else liste ; f
```

Exemple

```
 if [ $a = "rere" ] ; then echo toto ; else echo tutu ; fi
if [ -a myfile ] ; then echo myfile already exists ; fi
```

Quelques opérateurs de tests en sh ou bash

A mettre entre [ ]

operateur	vrai si	# operandes
-a (-e)	le fichier existe	1
-n	opérande est non vide	1
-z	opérande est vide	1
-d	opérande est un dossier	1
-f	opérande est un fichier	1
-eq	opérandes sont des entiers égaux	2
-ne	opposé de -eq	2
=	opérandes sont des chaines égales	2
!=	des chaines différentes	2

Tests différents si valeurs entières ou chaînes de caractères

Les boucles en sh

Plusieurs types de boucle

```
for X in bleu rouge vert jaune
do
echo $X
done


X=0
while [ $X -le 20 ]
do
echo $X
X=$((X+1))
done
```

Les tableaux

Variable tableau (bash)

Comme dans tout langage de programmation, il existe des variables indicées comme des tableaux

```
:~$ names=( Jennifer Tonya Anna Sadie Molly Millie)
:~$ echo $names
Jennifer
:~$ echo ${names[1]}
Tonya
:~$ echo ${names[*]}
Jennifer Tonya Anna Sadie Molly Millie
:~$ echo ${#names[@]}
6
```

Conclusion

La documentation des shells

- Pas une doc exhaustive
- `man` pour plus d'information
- Manque syntaxe des fonctions,
- Beaucoup de scripts dans /etc/init.d
- Il faut pratiquer