

Les systèmes UNIX

Aspects utilisateurs en ligne de commande

Université Grenoble-Alpes
Licence MIASHS

2023-2024

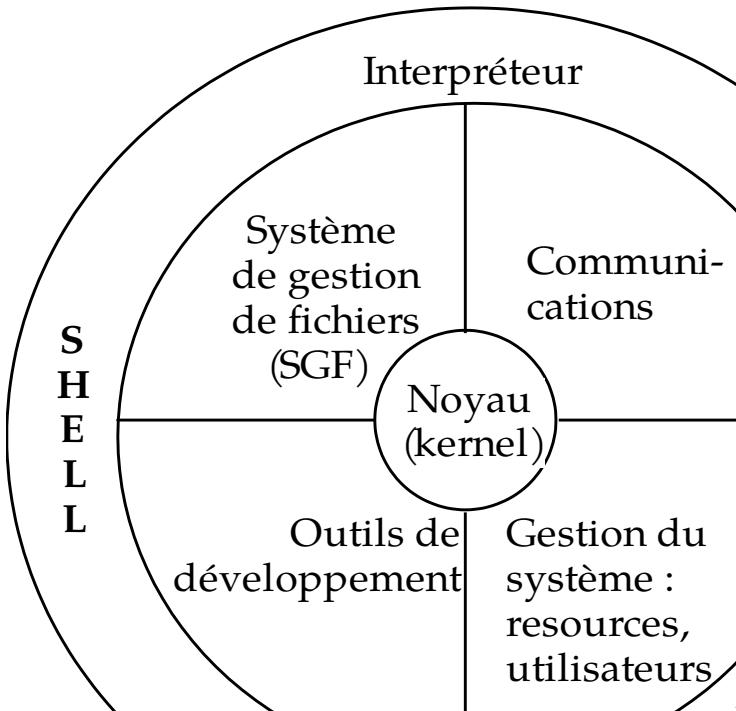
Introduction

- Un peu d'histoire
 - Conçu pour des mini-ordinateurs au début des années 70 chez Bell
 - Conçu par des informaticiens, pour des informaticiens
 - Conçu pour être évolutif et ouvert
 - Grande diffusion : moyens et gros systèmes mais aussi petits systèmes : Linux, MacOS, Android)
- Les différentes versions
 - Versions constructeurs (IBM-AIX, HPUX, Ultrix, ...)
 - BSD et dérivées (SunOS)
 - Linux
- Norme POSIX pour l'interface de programmation

Introduction

- Objectif du cours
 - Pratique d'Unix du point de vue utilisateur
 - Pratique de la programmation du langage de commande (shell)
- Bibliographie
 - UNIX et Linux - Utilisation et administration
Jean-Michel Léry - Pearson Education – 3ème édition 2011
 - Unix - Les bases indispensables
Michel Dutreix – ENI – 3ème édition 2015
 - Unix: Programmation et communication
Rifflet J.M. et J.B. Yunès, Dunod 2003

Architecture générale d'Unix



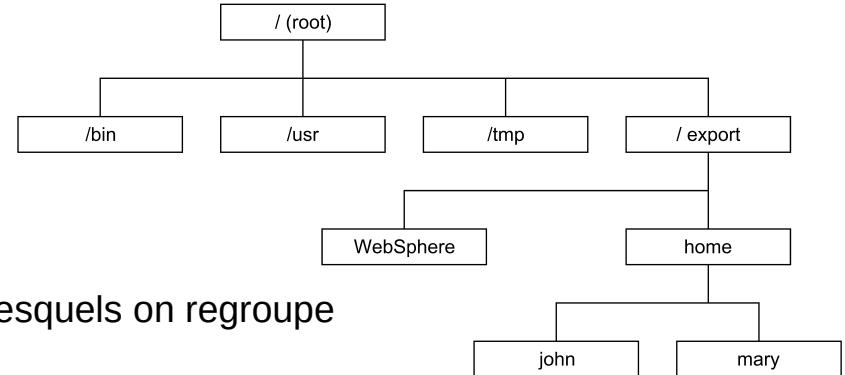
- Multi-tâches (multi-processus) et multi-utilisateurs
- Très grande facilité d'intégration en réseau
- Interface texte ou graphique

Plan du cours

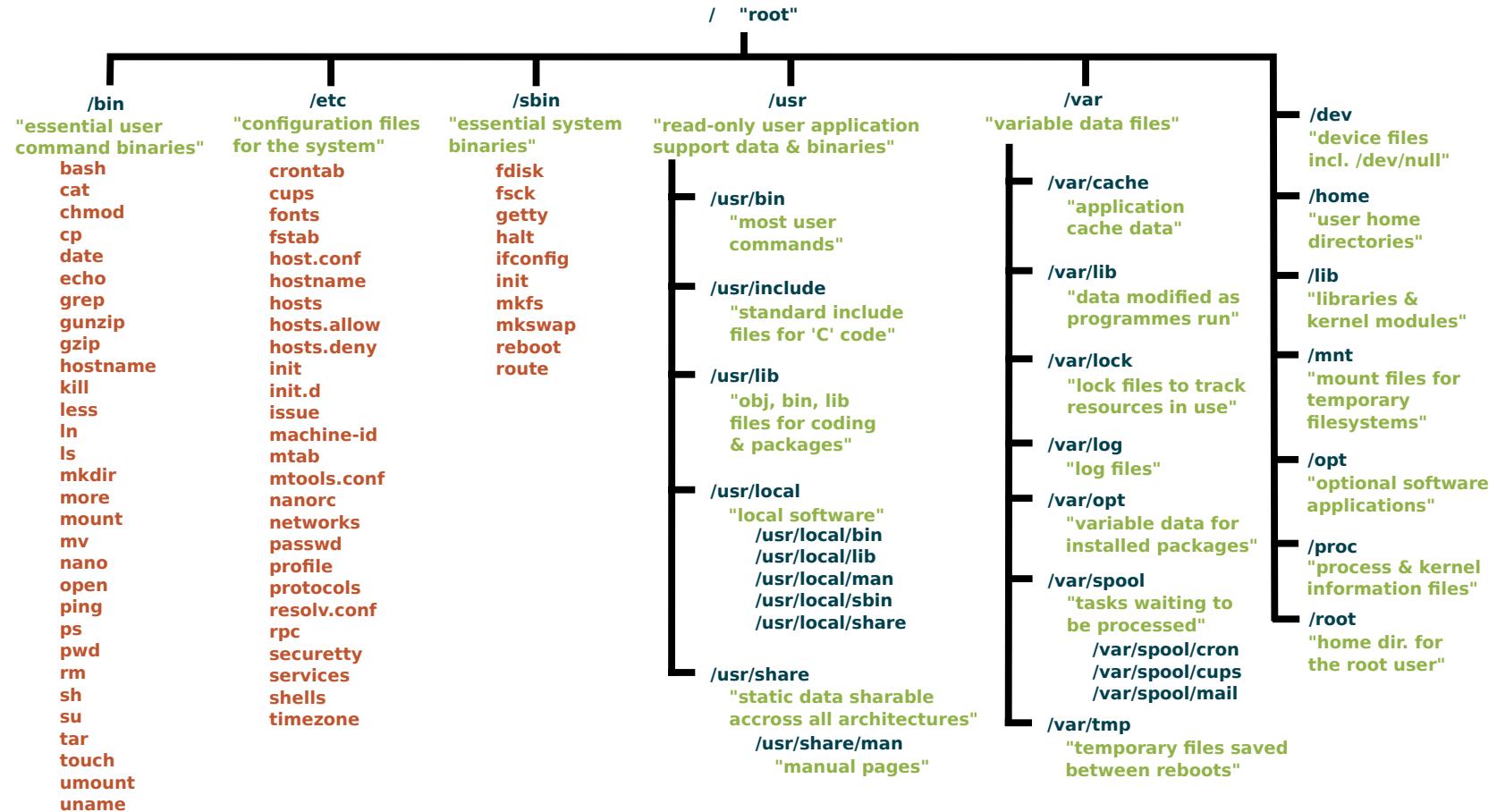
- Le SGF : système de gestion de fichiers
 - Structure arborescente
 - Utilisateur et protections
 - Commandes de base
- Les processus
 - Principe, initialisation du système
- Le langage de commande
 - Généralités
 - Environnement et variables
 - Composition des commandes
 - Écriture de scripts : paramètres, structures de contrôle
 - Fonctions et procédures

La vue du système de fichiers

- Les données sont rangées dans des fichiers
 - Un fichier possède un nom
 - Un fichier a un contenu (les données)
 - Les répertoires sont un type particulier de fichiers dans lesquels on regroupe plusieurs fichiers
- Vue arborescente avec une seule racine (root) « / »
 - Les répertoires (directories) sont des nœuds
 - Les autres fichiers (files) sont des feuilles
- La norme de la hiérarchie des systèmes de fichiers (FHS) :
 - https://fr.wikipedia.org/wiki/Filesystem_Hierarchy_Standard

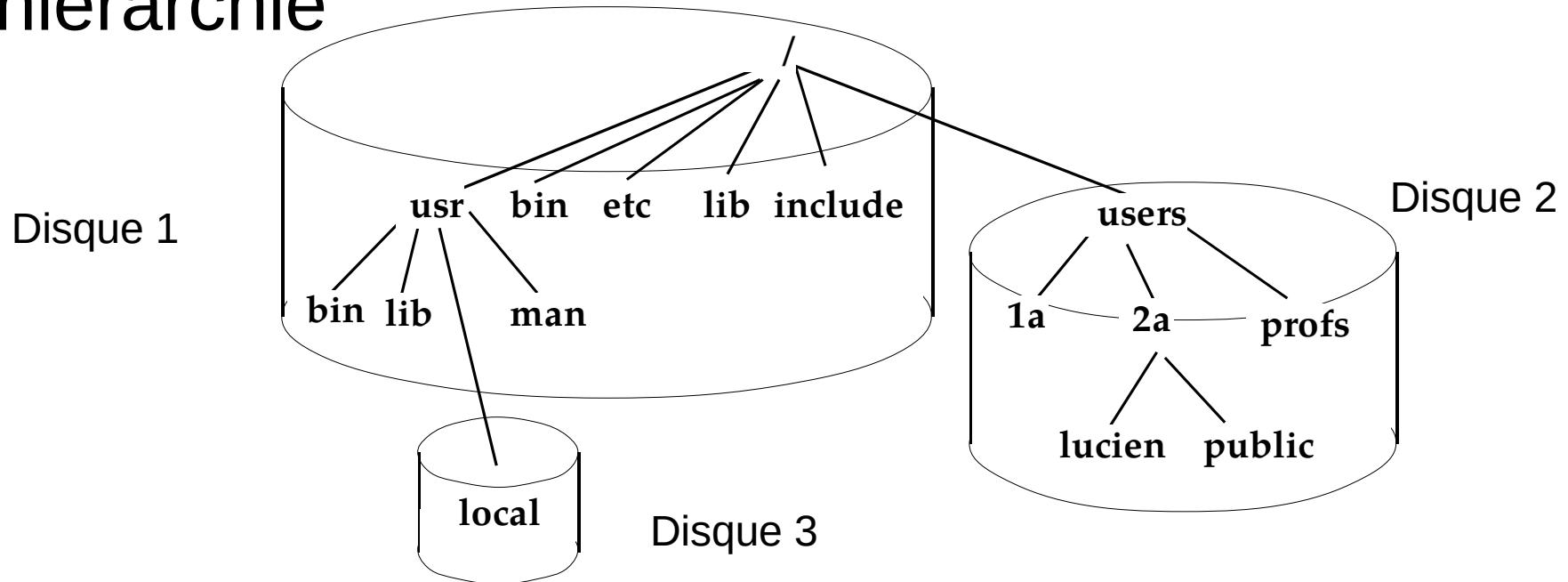


Exemple d'organisation des fichiers sous un système Linux



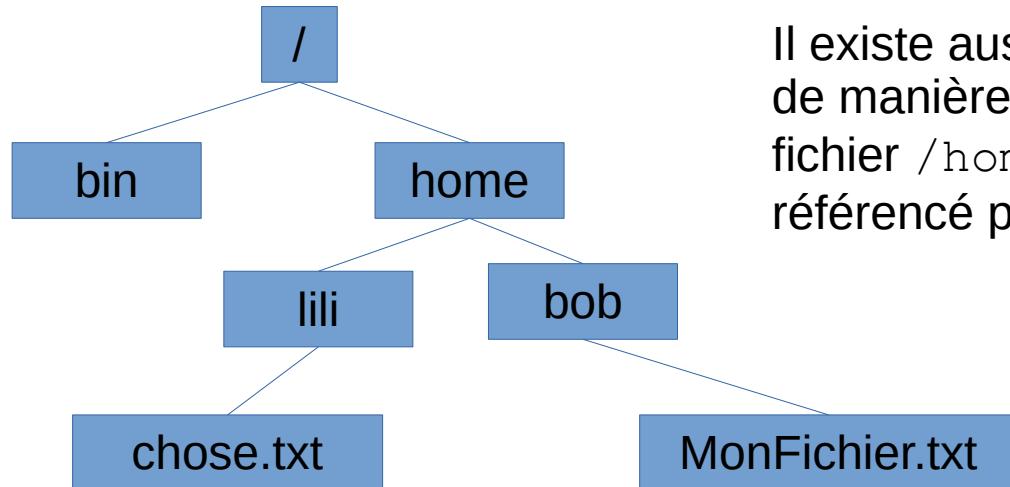
Les points de montages

- Les périphériques contenant des fichiers peuvent être attachés à différents endroits de la hiérarchie



La notion de chemin (path)

- Chaque fichier est identifiable par son chemin :
 - Ont utilisé « / » pour passer d'un répertoire à l'autre
 - Exemple : /home/bob/monFichier.txt



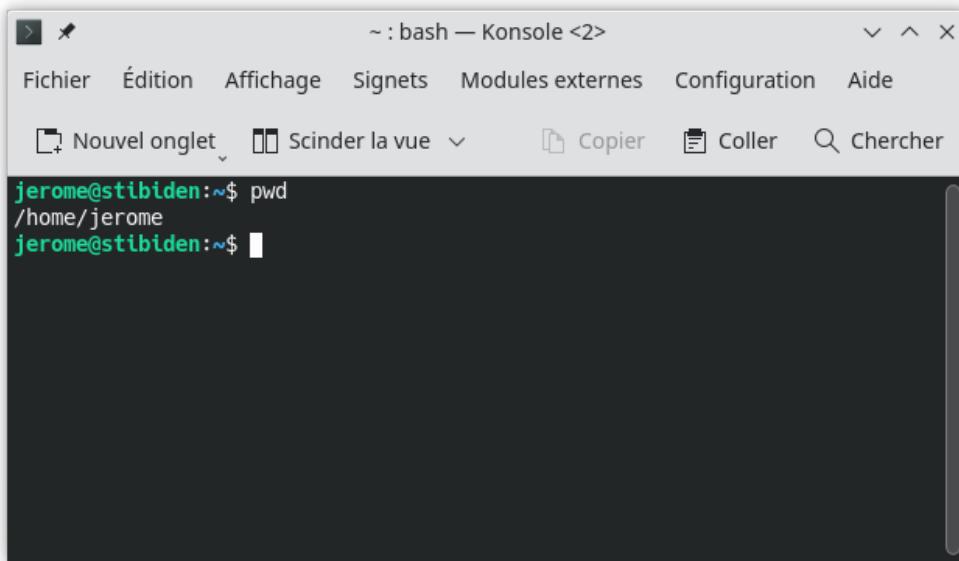
Il existe aussi une manière de référencer un fichier de manière relative : relativement à /home/bob, le fichier /home/lili/chose.txt peut être référencé par ../lili/chose.txt

.. permet de revenir au répertoire parent

Le « home directory »

- Unix est un système multi-utilisateurs
 - Chaque utilisateur possède son répertoire personnel, appelé « home »
 - Exemple : /home/dcissm1/alice
- Lorsque l'on ouvre un terminal, on se retrouve, au départ, dans notre répertoire personnel

Le terminal



- Un interpréteur de commande permet de « dialoguer » avec le système
 - Appelé aussi shell
 - Il existe plusieurs shells : **bash**, sh, ksh, csh, etc.
- Logiciel dans lequel on va exécuter un interpréteur de commande
 - Konsole (KDE)
 - Terminal (MacOS)
 - Gnome-terminal (Gome)
 - etc.

Syntaxe générale des commandes

- Principe de fonctionnement :
 - Un shell lit, interprète et exécute une commandes entrée par l'utilisateur
 - Les résultats sont (éventuellement) affichés à l'écran.
- Syntaxe

```
nom-de-la-commande -options --options-longues parametre1  
parametre2
```

- Les options peuvent avoir des arguments
- En général nom-de-la-commande référence un fichier exécutable (i.e un programme)

Exemple de la commande « Print Working Directory »

```
toto@trux:~$ /bin/pwd  
/home/toto
```

chemin vers la commande
généralement, /bin/ peut être
omis, pwd aurait suffit car le shell
va automatiquement chercher ce
qu'il y a dans /bin

Résultat : ici c'est le « *working directory* », i.e.
l'endroit où l'on se trouve.
Quand on exécute un terminal, le *working directory*
est le *home directory* de l'utilisateur

Partons en balade dans le système de fichiers

- `ls` (list) permet d'afficher la liste des fichiers
 - `ls` : liste des fichiers du répertoire courant « working directory »
 - `ls /bin` : liste des fichiers du répertoire `/bin`
- `cd` (change directory) permet de changer de répertoire courant
 - `cd /bin` : pour aller dans `/bin`
 - `cd` : revenir dans son « home directory »

Exemple

- Deux approches pour lister le contenu de /bin

Sans « bouger »

```
ls /bin
```

En allant sur « place », puis en revenant à la « maison »

```
cd /bin  
ls  
cd
```

Les caractères et chemins spéciaux

- Ces chemins sont évalués à l'exécution
 - ~ : home directory
 - . : répertoire « courant »
 - .. : répertoire « parent »
- Les caractères jokers dans les chemins
 - ? : remplace n'importe quel caractère (1 seul)
 - * : remplace n'importe quelle séquence de caractères (une même la chaîne vide)
 - [abcd] : soit a, b, c, ou d
 - [!efg] : ni e, ni f, ni g

Les fichiers « cachés »

- Leur nom commence par un point .
 - `~/.ssh` : rép. dans lequel on range ses clés ssh
 - `~/.` : le dossier parent de mon home directory
 - `~/.` : mon home directory (équivalent à ~)
- Pour afficher tous les fichiers (y compris ceux qui sont cachés) :

```
ls -a
```

A l'aide !

- Questions : c'est quoi ls ?, quelles sont les options ?, où est le programme ?, etc.
 - Un résumé : whatis ls
 - Le bon vieux manuel : man ls
 - Le paramètre --help : ls --help
 - info (navigateur)
 - Google :-)

```
jerome@stibiden:~$ whatis ls
ls (1)                      - list directory contents
jerome@stibiden:~$ which ls
/usr/bin/ls
jerome@stibiden:~$ man ls
jerome@stibiden:~$ info ls
jerome@stibiden:~$ ls --help
```

Quelques commandes pour manipuler les fichiers

- Création
 - Répertoire : `mkdir DIRECTORY`
 - Fichier (vide) : `touch FILE`
 - Lien symbolique (raccourci) : `ln -s TARGET LINKNAME`
- Copie :
 - `cp SOURCE DEST` (copie le fichier « source » vers un fichier nommé « dest »)
 - `cp SOURCE DIRECTORY` (copie le fichier « source » dans le répertoire « directory », la copie garde le nom)
- Déplacement/renommage :
 - `mv SOURCE DEST` (renomme, déplace le fichier « source » en « dest »)
 - `mv SOURCE DIRECTORY` (déplace le fichier « source » dans le répertoire « directory »)

Quelques commandes pour manipuler les fichiers

- Suppression :
 - `rm FILE...` (supprime le fichiers passés en paramètre)
 - `rmdir DIRECTORY...` (supprime les répertoires passés en paramètre, ces rép. Doivent être vides, sinon utiliser `rm -rf ...`)
- Affichage du contenu des fichiers (texte)
 - `cat` , `more` (affichage paginé), `less` (comme `more` mais avec plus de fonctions, d'où son nom ;-)

Utilisateurs et groupes

- Liste des utilisateurs : cat /etc/passwd ou getent passwd

- Chaque utilisateur est décrit par:

- Un identifiant : uid (user identifier)
- Un nom : username
- Un groupe « principal » : gid
- Un champ « gecos » : son nom complet par exemple
- Son répertoire personnel (homedir)
- Le programme lancé à la connexion (le shell par exemple)

- Les différents types d'utilisateurs

- root, l'admin : Il possède l'uid 0 et son groupe principal est appelé aussi root avec gid 0
- Ceux dont uid < 1000 (ca dépend des systèmes) : utilisateurs spéciaux réservés pour le système

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
...
bob:x:1000:1000:Bob Marley:/home/bob:/bin/bash
```

Les groupes

- Un utilisateur possède un (et un seul) groupe principal et peut appartenir à plusieurs groupes (secondaires)
- Liste des groupes
 - cat /etc/group
 - getent group
 - Format :
 - groupname:x:gid:liste des utilisateurs secondaires

```
root:x:0:  
daemon:x:1:  
bin:x:2:  
sys:x:3:  
adm:x:4:syslog,jerome
```

Gestion des utilisateurs et groupes

- Utilisateurs
 - Ajout (adduser, useradd) , modification (usermod), suppression (deluser)
 - Changement de votre mdp : passwd
 - Infos : id, id USERNAME
- Groupes
 - Ajout (addgroup), modification (groupmod), suppression (groupdel)
- La gestion des utilisateurs et groupes requiert d'être super-utilisateur (root)
 - su -
 - sudo lecommande

Les droits sur les fichiers

- Unix permet la possibilité de gérer des permissions sur les fichiers
 - 3 niveaux : l'utilisateur propriétaire (u), le groupe (g) d'utilisateurs, les autres (o)
 - 3 types de droits : lecture (r), écriture (w), exécution (x)
- Extrait d'un résultat de la commande `ls -l /` :

Type de Fichier	Les droits u g o	nb de liens	Groupe	Date de dernière modification	
	l rwxrwxrwx	1	root	7 févr. 11 2021	bin -> usr/bin
	drwxr-xr-x	156	root	12288 janv. 20 09:26	etc
	-rwxr-xr-x	120	root	4096 févr. 16 2021	choses.txt

I : lien
d : répertoire
- : fichier « normal »

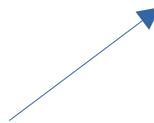
Utilisateur propriétaire

Taille (octets)

Gestion des droits

- La gestion des droits se fait via la commande chmod

```
chmod [OPTIONS] [ u g o a ] [ - + = ] [ r, w, x ] fichier
```



Qui ?
(u)ser
(g)roup
(o)ther
(a)ll ->

Action?
- enlever
+ ajouter
= garder

droits?
r : read
w: write
x : execute

Quelques autres commandes

- La date
 - date
- Effacer le texte du terminal
 - clear
- Qui suis-je ?
 - whoami

Gestion des processus

- Processus = l'exécution d'un programme
- Unix est multitâches
 - Les processus sont identifiés par un entier : pid (processus identifier)
 - Chaque processus est créé par un processus père
 - Cela donne une hiérarchie de processus (c.f. diapo suivante)
 - PPID → parent PID
 - Les processus possèdent les droits de l'utilisateur qui l'exécute

Les processus

- Au démarrage :
 - Le noyau s'exécute, initialise le matériel, puis le premier processus est lancé (init ou systemd)
- Extrait de la hiérarchie de process: pstree

```
systemd─AppRun─12*[{AppRun}]
         └─NetworkManager─2*[{NetworkManager}]
         ├─accounts-daemon─2*[{accounts-daemon}]
         ├─agent─2*[{agent}]
         ├─at-spi-bus-laun─dbus-daemon
         │   └─3*[{at-spi-bus-laun}]
         └─systemd─(sd-pam)
             ├─dbus-daemon
             ├─dconf-service─2*[{dconf-service}]
             ├─kiod5─2*[{kiod5}]
             ├─krunner─kate─7*[{kate}]
             │   ├─konsole─bash─pstree
             │   └─6*[{konsole}]
```

Toute commande exécutée dans le terminal crée un fils du processus de l'interpréteur de commande (bas) qui est lui-même fils du processus terminal (Konsole ici)

Gestion des processus

- Lancer un processus en « tâche de fond »
 - Permet de récupérer la « main » sur terminal
 - Avec &, exemple : `xeyes &`
 - Avec `Ctrl+z` puis `bg`
 - Ces commandes donnent en retour le pid du processus
- Lister les processus
 - `ps` (ceux du terminal), `ps aux` (lister tous les processus)
 - `pstree`, `htop`, etc.

Gestion des processus (suite)

- Les signaux pour envoyer des « messages » aux processus

```
kill -NUM_SIGNAL PID
```

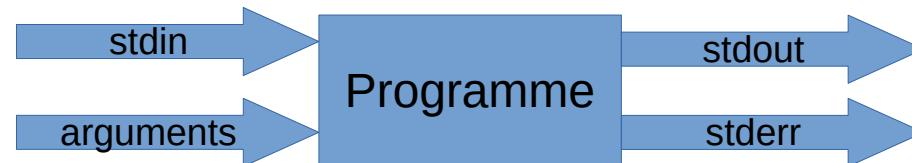
- Liste des signaux : man 7 signal

SIGTERM (15) : arrêt « propre »

SIGKILL (9) : arrêt « brutal »

Redirections E/S

- Chaque processus est associé à 3 flux standards
 - Généralement utilisés pour communiquer avec l'utilisateur
 - Entrée standard (clavier par défaut) : stdin
 - Redirigée avec <
 - Sortie standard (l'écran, le terminal) : stdout (identifiant 1)
 - Redirigée avec > (ou 1>)
 - Exemple : ls -l / > liste.txt
 - Sortie d'erreur (l'écran, le terminal) : stderr (identifiant 2)
 - Redirigée avec 2>



Redirections (suite)

- Redirections :
 - Par défaut > écrase le fichier destination. Si on veux ajouter il faut utiliser >>
 - Si on veux combiner à la fois les deux sorties : `ls 1>stdout_stderr.txt 2>&1`
- Les tubes (pipes)
 - On peut « brancher » la sortie d'un processus vers l'entrée d'un autre grâce aux « tubes » |
 - Utilise une mémoire partagée entre deux processus (on verra cela plus tard...)
 - Exemple : `ls -1 / | wc -l`
 - La sortie standard de la commande `ls -1` est envoyé à l'entrée standard de la commande `wc -l`
 - Compte le nombre de fichier (non cachés) du répertoire /

Le shell BASH

- Acronyme de Bourne Again Shell
- Shell par défaut sous linux
- Fonctions
 - Permettre d'exécuter des commandes de manière « interactive »
 - Analyse syntaxique, redirection vers écran
 - Permettre d'exécuter des scripts (mode « batch »)
 - Des sortes de « programmes » regroupant des commandes

Les variables

- Définition/affectation
 - `maVariable=quelquechose`
- Substitution (i.e. lecture)
 - `$maVariable` (exemple : `echo $maVariable`)
 - Concaténation : `echo "blabla${maVariable}blabla"`
- Visibilité : le shell courant (en cours d'exécution)
 - Visibilité aux processus fils : `export maVariable`
- Liste des variables
 - Les variables « locales » : `set`
 - Les variables d'environnement ou exportées : `printenv`

Les variables (suite)

- Les tableaux
 - `montab=("un" "deux" "trois")`
 - `echo "3eme val=${montab[2]}"`
 - `echo "tout le tableau=${montab[@]}"`
- Longueur d'une variable
 - `${#variable}`
- La substitution de commandes (bon ce n'est pas des variables)
 - `$(date)` ou `$`date`` : dans une expression, cela est remplacé par le résultat de l'appel à la commande date
- Utilisation avancée :
 - <https://linuxhandbook.com/variables-bash-script/>

Evaluation

- Par défaut, les données sont des chaînes de caractères.
 - Si on veux évaluer des expressions arithmétiques, il faut utiliser une des syntaxes suivantes :
 - `let a=3+4`
 - `A=$((3+4))`
 - Attention :
 - `a=3+4`
 - `echo a # affiche 3+4`

Les variables « usuelles »

- \$PATH : représente l'ensemble des répertoires à partir desquels le shell va « chercher » les commandes à exécuter
 - Format : chemin1:chemin2:etc
 - Modification/ajout d'un chemin : export PATH=/mon/nouveau/rep:\$PATH
- \$HOME : contient le chemin vers le home directory
- \$_ : la dernière commande exécutée
- \$\$SHELL : l'exécutable du shell
- ...

Les variables spéciales

\$? : code de retour de la dernière commande. 0 si cela s'est bien passé

\$\$: pid du shell

\$! : pid de la dernière commande exécutée

\$0 : nom du script

\$1 ... \$9 : les paramètres de la ligne de commande

\$* : la concaténation de tous les paramètres

\$# : nombre de paramètres

Ordre des initialisations

- Certains « scripts » sont exécuté au démarrage d'un shell
 - Ces fichiers servent à initialiser entre autre certaines variables
- Mode interactif, login shell
 - /etc/profile
 - Puis ~/.bash_profile, ~/.bash_login, ~/.profile
- Mode interactif mais pas login shell (ce que l'on utilise souvent)
 - /etc/bash.bashrc, ~/.bashrc

Composition des commandes

- Séquentiel (;)
 - `cp f1 f2 ; rm f1 ;`
- Conditionnelle et (`&&`) : si OK alors ...
 - `cc coucou.c && mv a.out monProg`
- Conditionnelle ou (`||`) : si NonOK alors ...
 - `cc coucou.c || echo "Il y a un pb de compilation!"`

Les scripts

- Ce sont des fichiers texte contenant des instructions shell à exécuter (i.e. une sorte de programme)
- Ils commencent par :
 - `#!/bin/bash`
- Commentaires :
 - `#un joli commentaire`
- Fin d'exécution
 - `exit N ; #(ou N est un code de retour)`

Scripts : entrées/sorties

- Sortie : echo
 - echo bonjour ; # affiche un bonjour avec retour à ligne
 - echo -n bonjour ; #idem mais sans retour à la ligne
 - echo -e bonjour\tl3\tmiashs
- Entrée : read
 - read v1 v2 ... vn
 - lit les n prochains mots et les affecte aux variables v1, ..., vn
 - vn contient les derniers mots (si le nombre de mots > n)

Les paramètres

- Un script peut prendre des paramètres
 - `./monScript.bash truc chose bidule`
- Les paramètres sont stockés dans les variables `$1, ..., $9`
 - Comment faire si il y a plus de 9 paramètres ?
 - On utilise la commande intégrée `shift` qui permet de décaler les paramètres
 - Après un `shift`, `$1` contient le deuxième paramètre, ..., `$9` le dixième
 - On peut décaler plusieurs fois pour obtenir les paramètres suivants
- Quelques variables spéciales sur les paramètres
 - `$#` : nb de paramètres, `$*` : toute la chaîne de paramètres, `$@` : le tableau des paramètres

Exemples d'utilisation des paramètres

```
#!/bin/bash

for i in $*;
do
    echo $i
done;
```

```
#!/bin/bash

# This script can clean up files
# that were last accessed over 365 days ago.

USAGE="Usage: $0 dir1 dir2 dir3 ... dirN"
if [ "$#" = "0" ]; then
    echo "$USAGE"
    exit 1
fi
# $#=0 means false
while (( "$#" )); do
    if [[ $(ls "$1") = "" ]]; then
        echo "Empty directory, nothing to be done."
    else
        find "$1" -type f -a -atime +365 -exec rm -i {} \;
    fi
    shift
done
```

Les conditionnelles

- Utilisent le code de retour de la commande
 - 0 → VRAI,
 - !=0 → FAUX

```
if <commande>
then
  <instruction>
fi
```

```
if test $# -eq 0
then
  echo Pas de paramètres
fi
```

```
if <commande>
then
  <inst1>
else
  <inst2>
fi
```

```
if cc -o tp tp.c
then
  tp
else
  echo Erreurs...
fi
```

```
if <com1>
then
  <inst1>
elif <com2>; then
  <inst2>
...
else
  <instN>
fi
```

La commande test

- La commande test permet de réaliser des expressions conditionnelles
 - Sur les fichiers
 - `test -r <path>`, `test -w <path>`, `test -x <path>`, `test -f <path>`, `test -d <path>`, etc.
 - Sur les chaînes
 - `test <ch1> = <ch2>`, `test <ch1> != <ch2>`, `test -z <ch1>`, `test -n <ch1>`, etc.
 - Sur les nombres
 - `test <nb1> -eq <nb2>`, `test <nb1> -lt <nb2>`, `test <nb1> -gt <nb2>`, etc.
- Exemples
 - `test "01" = "1"` → FAUX
 - `test "01" -eq "1"` → VRAI

La commande test (suite)

- Les compositions d'expressions
 - -a (et), -o (ou)
 - ! (negation), (<expr>)
- La commande test est aussi accessible via []
 - [\$# -gt 1]
 - Il faut penser aux espaces entre [et \$# (et aussi entre 1 et]))

Les itérations

```
while <commande>  
do  
  <instructions>  
done
```

```
while [ -r "$1" ]  
do  
  cat $1 >> liste  
  shift  
done
```

```
for f in tp1.c tp2.c  
do  
  cc -c $f 2> trace  
done
```

```
for <var> [ in <liste> ]  
do  
  <instructions>  
done
```

```
for f in *.c  
do  
  cc -c $f 2> trace  
done
```

```
until <commande>  
do  
  <instructions>  
done
```

```
until [ ! -r "$1" ]  
do  
  cat $1 >> liste  
  shift  
done
```

```
echo "Paramètres :"  
for f  
do  
  echo $f  
done
```

Exemple de script

```
#!/bin/bash
if test $# -eq 0
then
    echo Usage: testFicMult nomFichier1 ... nomFichierN
else
    for i in $*
    do
        if [ -d "$i" ]; then
            echo $i est un repertoire
        elif [ -f "$i" ]; then
            echo le fichier $i existe
        else
            echo $i n'existe pas
        fi
    done
fi
```

Recherche de fichiers

```
find <chemin(s)> <critère(s)> <action(s)>
```

- Permet la recherche récursive dans le(s) répertoire(s) indiqué(s) par (chemin(s))

- Les principaux critères (critère(s)) sont :

```
-name '<motifs>'  
  
-size <[+|-]taille>  
  
-mtime <[+|-]date>  
  
-user <nom|UID>  
  
-newer <fichier référence>
```

- les principales actions (action(s)) sont :

```
-print  
  
-ls  
  
-exec <commande shell avec {} pour spécifier le fichier trouvé> \;  
  
-ok <commande shell avec {} pour spécifier le fichier trouvé> \;
```

Exemples :

```
find /home /usr -name 'ab*' -print 2> /dev/null  
  
find . -name "*.o" -ok -exec rm {} \;  
  
find . ! -user root -print
```

Recherche dans les fichiers texte

`grep <regexp> [fichier ...]`

- Affiche uniquement les lignes, des fichiers passés en argument, correspondantes à l'expression régulière regexp.
- Les options :
 - v inverse le résultat de la commande (affiche seulement les lignes ne correspondant pas à regexp)
 - c retourne le nombre de correspondances
 - n affiche les numéros des lignes correspondantes
 - l affiche les noms des fichiers contenant des lignes correspondant à regexp
 - i : ne tient pas compte de la casse des caractères
- Exemple

`grep -ni "ab.." **`

Manipulation de texte : cut

- Projection : sélection verticale
 - `cut -d<délimiteur> -f<champ(s)> [fichier]`
 - Affiche les champs spécifiés avec l'option `-f` et séparés par le délimiteur indiqué après l'option `-d`
 - `cut -c<colonne(s)> [fichier]`
 - Affiche les colonnes de caractères indiquées après l'option `-c`
- Pour afficher les 3ème et 6ème colonnes du fichier `liste.txt` :
`cut -d":" -f3,6 liste.txt`
`cut -c1-10 liste.txt`

Autre commandes

- Word Count : compter les mots, lignes ou caractères d'un fichier texte
`wc monFichier.txt`
- sort : trier les lignes d'un fichier
- uniq : éliminer les doublons (sur fichier trié)

Bibliographie

- https://fr.wikibooks.org/wiki/Le_syst%C3%A8me_d%27exploitation_GNU-Linux
 - Chapitre 5