

# FORMATION QUALITE LOGICIELLE

Autom : Unix (3j)



Ecole de la Qualité Logicielle



# Présentation des objectifs



# Partie 1 : Présentation des outils



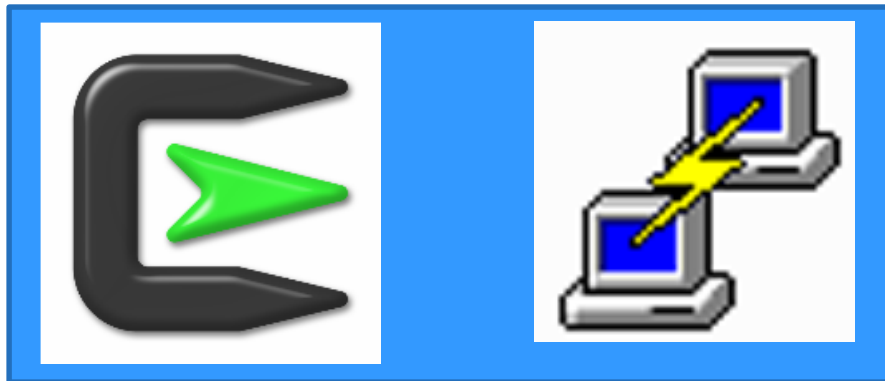
FTP



Machine(s) virtuelle(s)



SSH



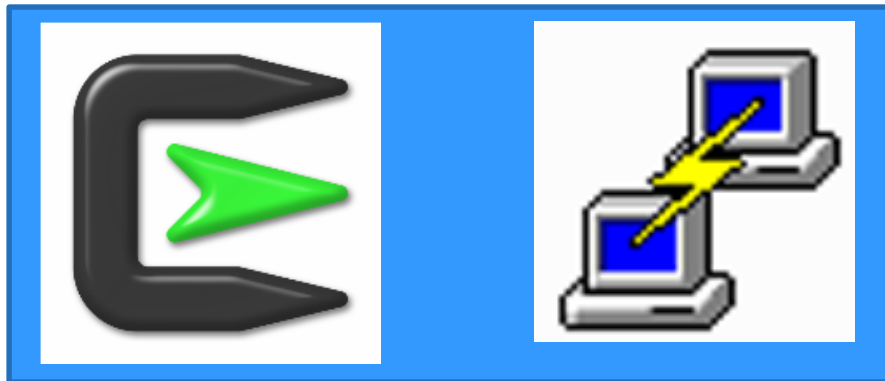
# UNIX

## Présentation des outils

FTP



SSH



ubuntu



Machine(s) virtuelle(s)

### Cygwin

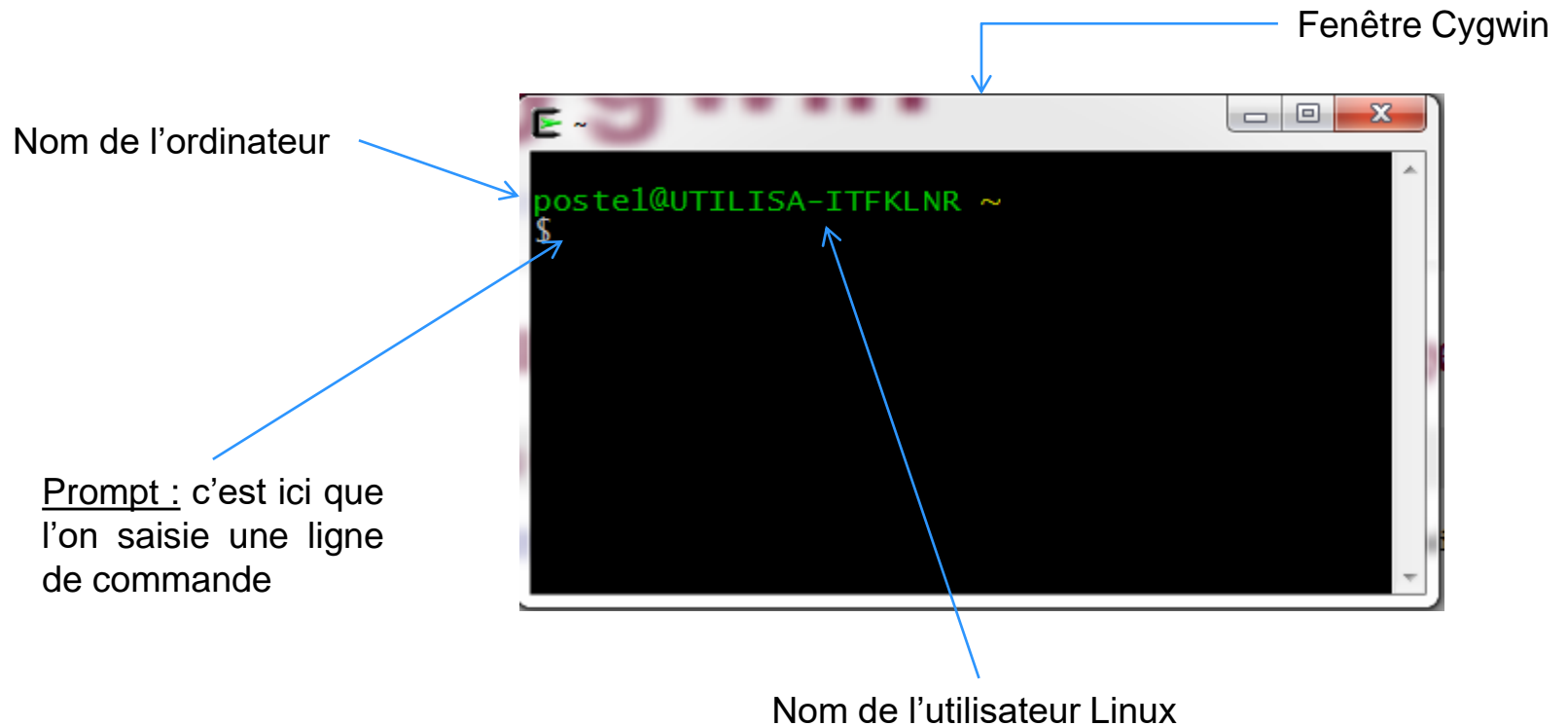
- ✓ **Est une application avec laquelle nous pouvons utiliser Linux**
- ✓ Est une collection de logiciels libres qui permet d'émuler un système Linux sur une machine Windows.
- ✓ Utiliser Cygwin sur Windows équivaut à utiliser une machine Linux
- ✓ **SSH :**
  - ✓ Possibilité de se connecter en SSH à une machine Linux
  - ✓ Possibilité de se connecter en SSH depuis une machine Linux sur Cygwin
- ✓ Pour rappel :
  - ✓ L'ajout de module(s) s'effectue via l'installateur de Cygwin

# UNIX

## Présentation des outils : Cygwin (rappel) (2/3)

Lancer Cygwin

- ✓ Dans le menu démarrer de Windows, cliquer sur « Cygwin64 Terminal »
- ✓ La fenêtre suivante apparaît



### Cygwin (linux en général) :

#### ✓ Exemples fichiers répertoire utilisateur :

- ✓ **.bash\_history** : fichier dans lequel est archivé toutes les lignes de commandes utilisées par l'utilisateur propriétaire du répertoire
- ✓ **.bash\_rc** : détermine le comportement du shell interactif
- ✓ **.profile** : paramètres liés au compte de l'utilisateur
- ✓ **.ssh** : paramètres SSH

### Putty :

- Client SSH
- Présentation rapide de l'outil Putty
- Peut aussi permettre la génération de clé de connexion



### **Filezilla :**

- Client FTP / SFTP
- Existe également une version serveur



### Virtual Box :

- Moteur de machine(s) virtuelle(s) Windows, Linux, Solaris, Mac..
- Outil gratuit, peu gourmand en ressources, facile à installer
- Utilisable depuis n'importe quel OS
- Permet de ne pas mettre en place de dual boot
- Permet de monter, gérer et détruire rapidement des machines virtuelles
- Permet la création de « templates » de machines virtuelles
- Permet une configuration très simple des machines virtuelles
- Interface graphique « User friendly »

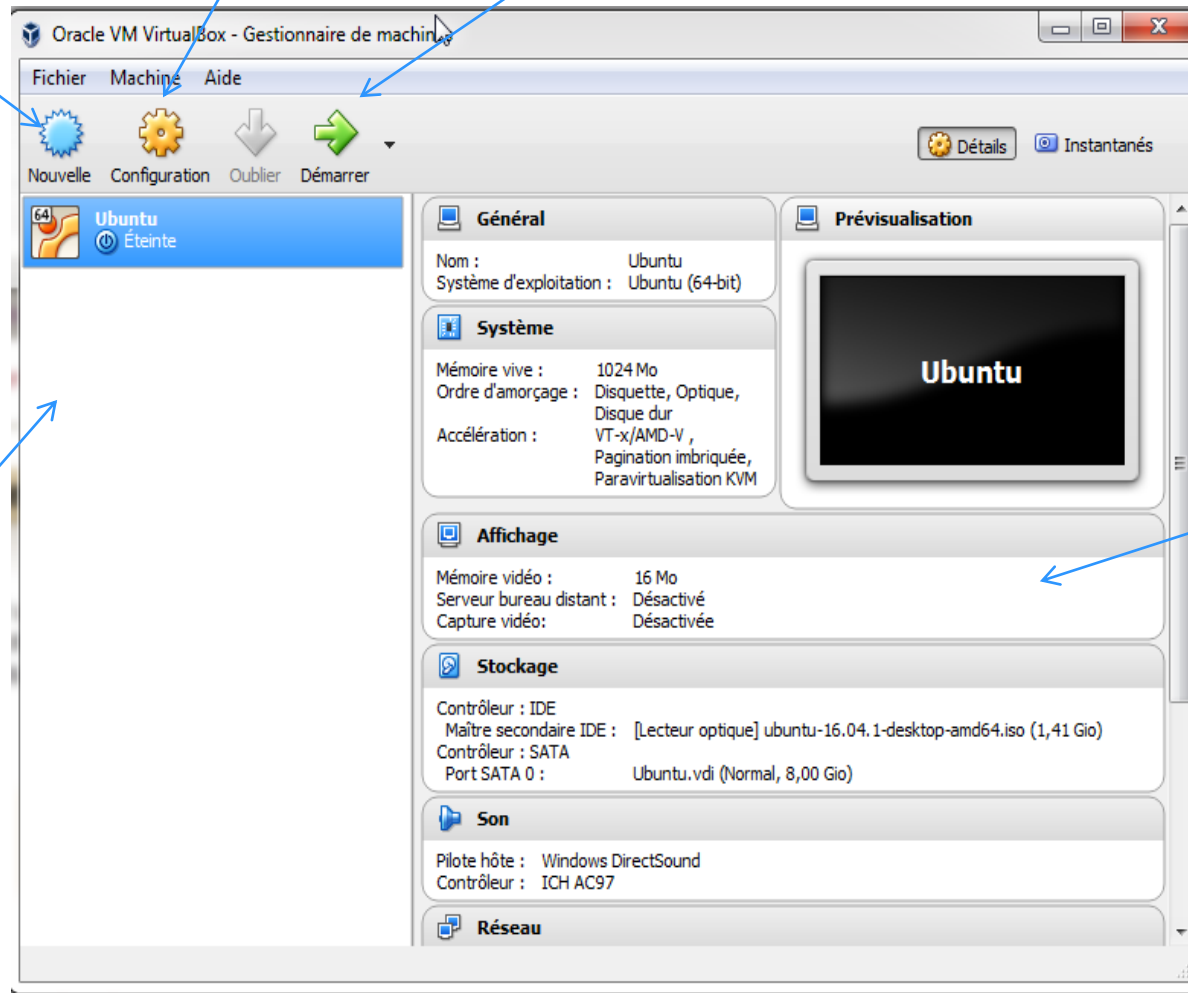
# UNIX

## Présentation des outils : Virtual Box : Interface d'accueil

Nouvelle machine

Configurer sa machine

Démarrer une machine virtuelle



Liste des machines

Caractéristiques techniques de la machine

### Menu fichier :

- Paramétrer VB (Virtual Box)
- Importer une machine virtuelle
- Exporter une machine virtuelle
- Gérer les médias et les différents réseaux connectés
- Mettre à jour VB


### Machine :

- Gérer une VM (Virtual Machine) → démarrer, éteindre, suppr, créer, cloner, grouper ...
- Afficher l'explorateur de fichiers VB
- Trier les VMs

### Aide :

- Informations, aides et versions VB

Choix de la machine : OS, technologie, version :



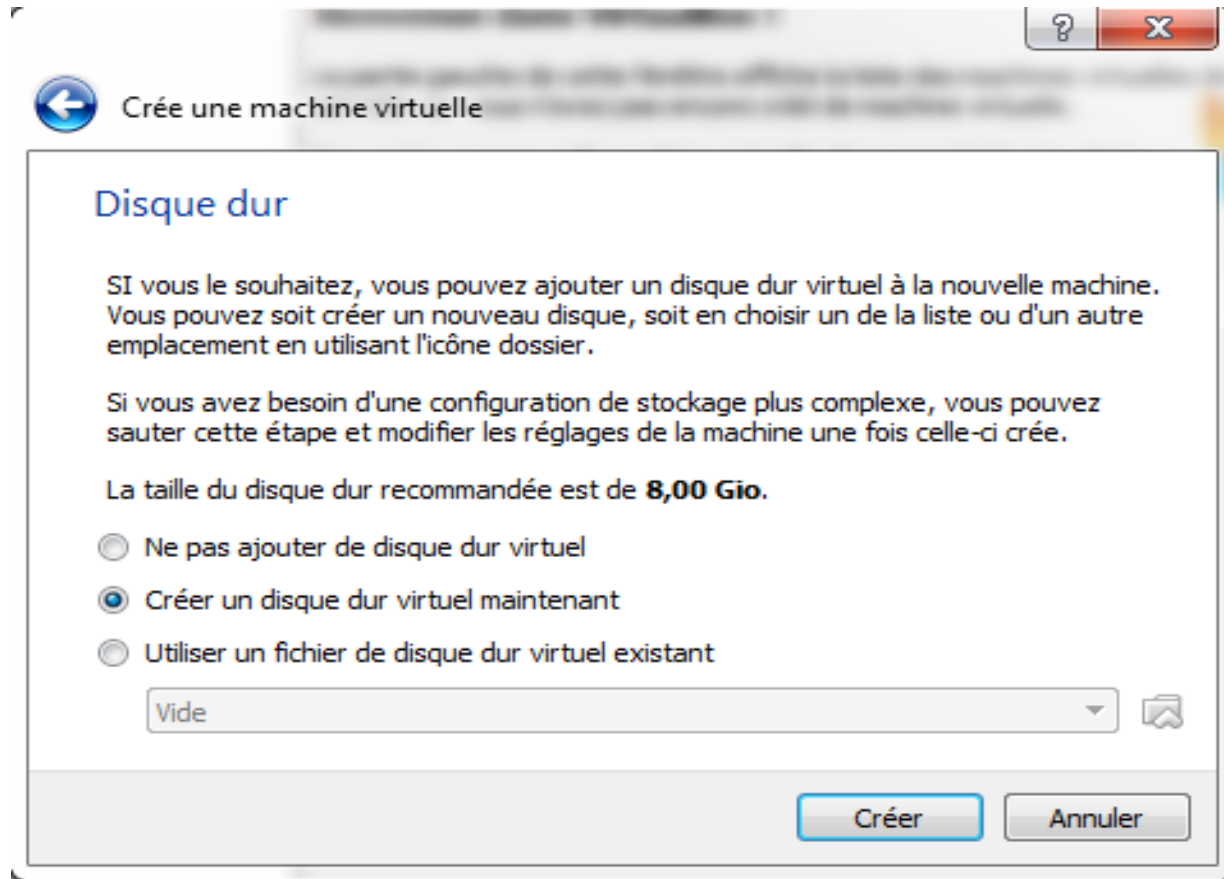
The screenshot shows the 'Crée une machine virtuelle' (Create a virtual machine) wizard in Oracle VM VirtualBox. The window title is 'Crée une machine virtuelle'. The first step is 'Nom et système d'exploitation' (Name and operating system). The instructions state: 'Veillez choisir un nom pour la nouvelle machine virtuelle et sélectionner le type de système d'exploitation que vous envisagez d'y installer. Le nom que vous choisirez sera repris au travers de VirtualBox pour identifier cette machine.' (Please choose a name for the new virtual machine and select the type of operating system you plan to install on it. The name you choose will be used by VirtualBox to identify this machine.)

The form contains three fields:

- Nom :** A text input field containing 'Ubuntu'.
- Type :** A dropdown menu showing 'Linux'.
- Version :** A dropdown menu showing 'Ubuntu (64-bit)'.

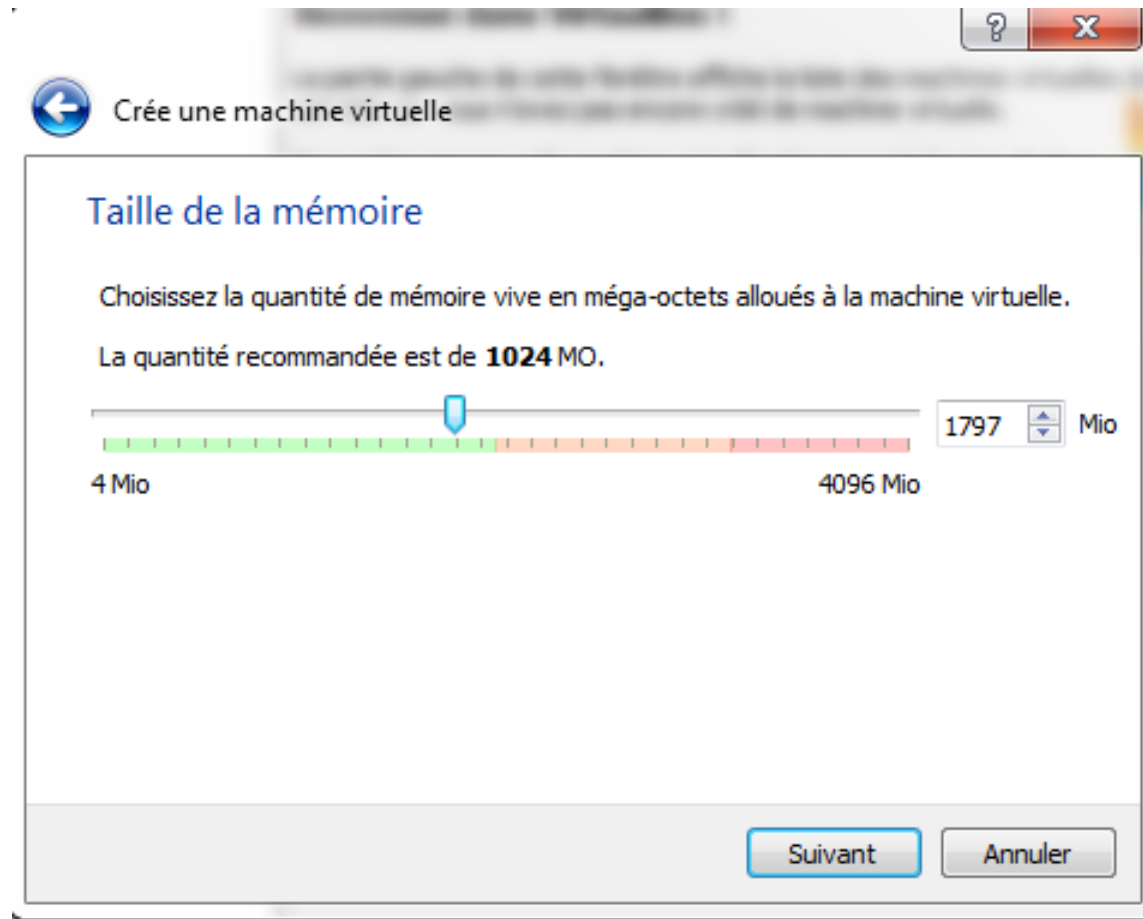
There is a small icon with '64' and a CPU symbol next to the version dropdown. At the bottom, there are three buttons: 'Mode expert' (disabled), 'Suivant' (Next, highlighted in blue), and 'Annuler' (Cancel).

### Création du disque dur de la VM :



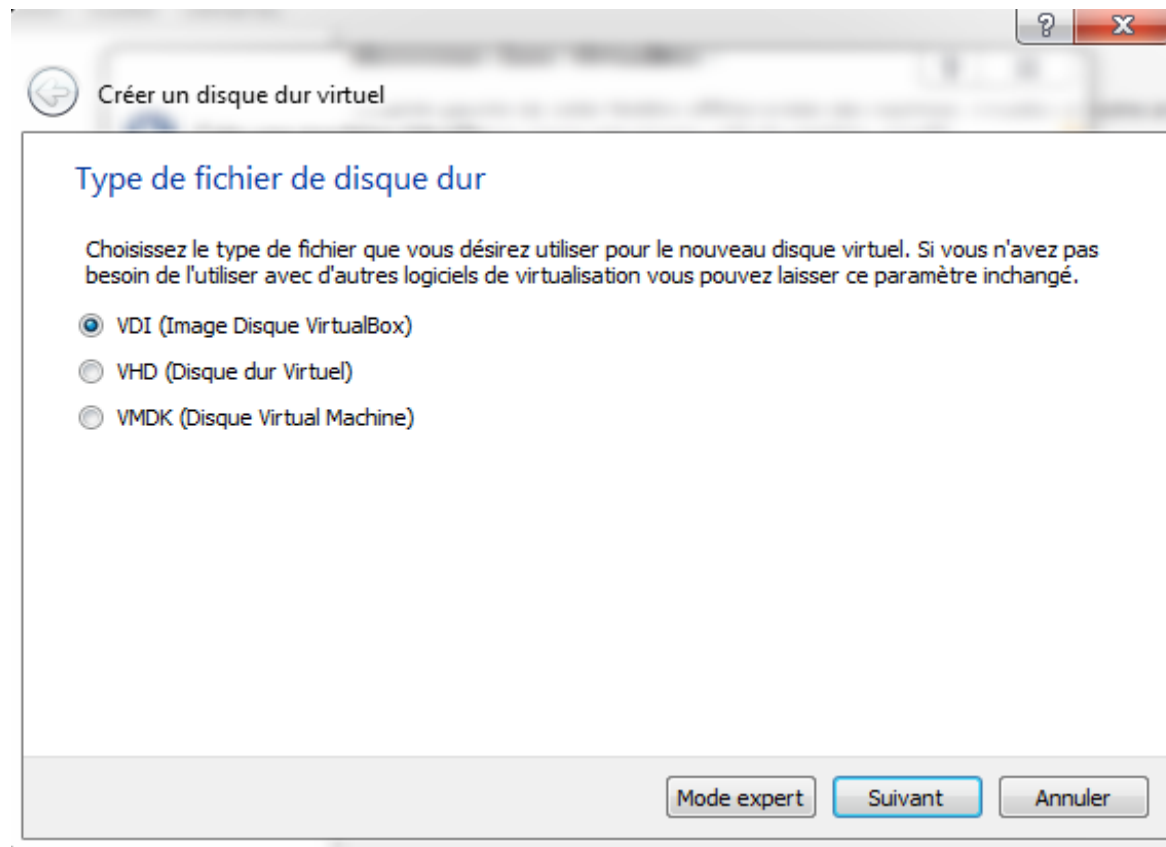
Allocation de la mémoire vive de la VM :

**ATTENTION : la mémoire ne peut pas dépasser la mémoire de la machine réelle. Il est conseillé de ne pas dépasser la zone verte**



### Choix du type du volume disk de la VM :

- VDI → image disk VirtualBox ( pour système VB → notre cas ici)
- VHD → Disque dur Virtuel ( pour système Microsoft Virtual PC)
- VMDK → Disque Virtual Machine ( par exemple pour serveur ESXi)

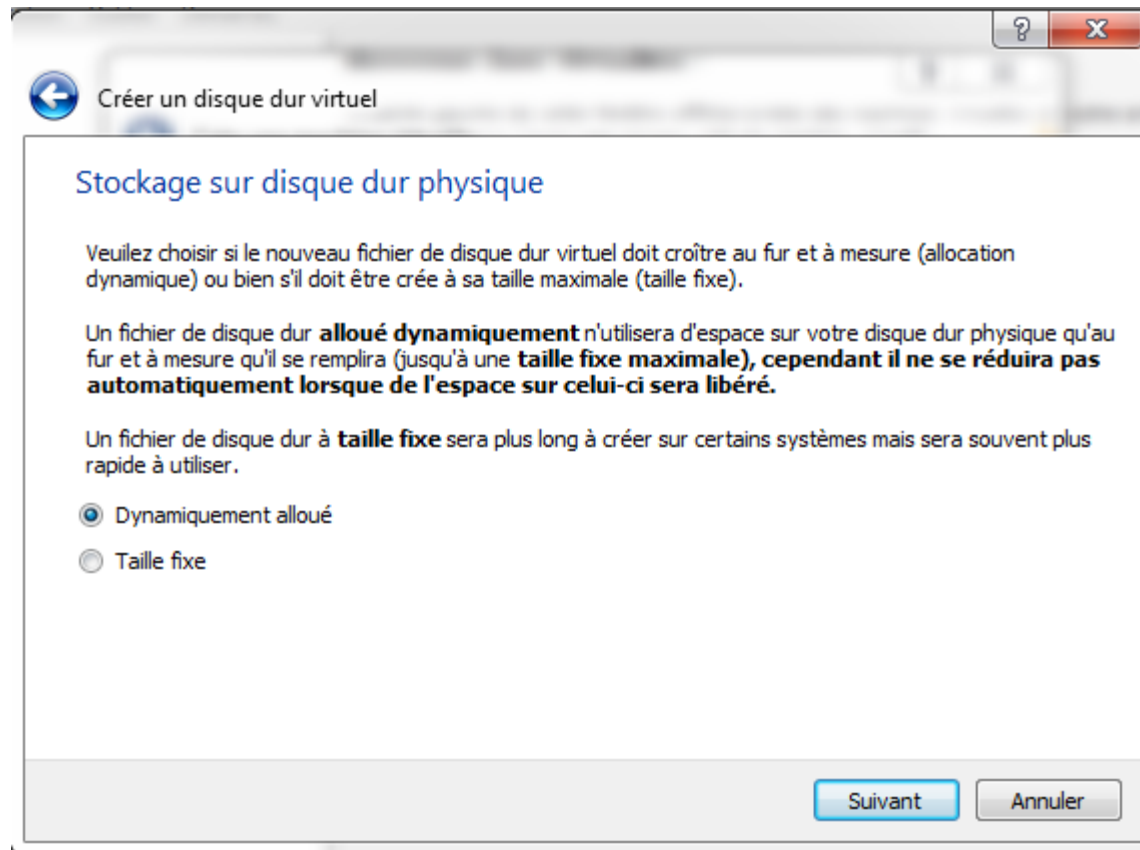




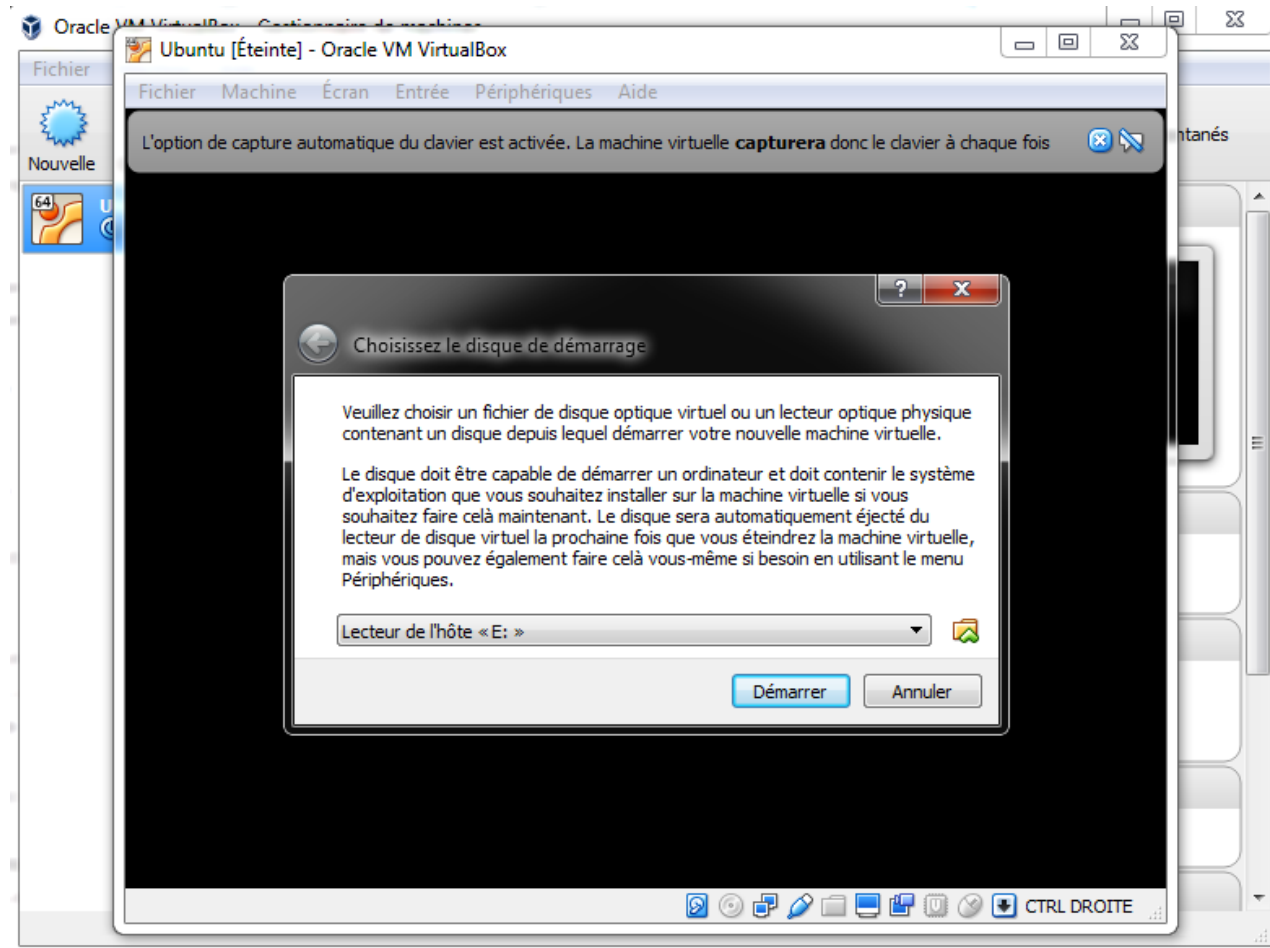
### Espace disque alloué :

**Dynamiquement alloué** : paramétrable ensuite avec LVM

**Taille fixe** : espace prédéterminé à l'avance



### Choix de l'image .iso





### Commandes basiques :

- Lancer VIM en ligne de commande : depuis la console, taper vim
- Mode insertion : dans VIM taper i puis saisir du texte

Ne pas être dans un mode spécifique :

- Enregistrer : :w (ajouter {nom\_fichier} si nouveau fichier)
- Quitter : :q
- Enregistrer et quitter : :wq
- Forcer : ajouter !
- Afficher les numéros de ligne : :set nu
- Rechercher un mot : /
- Aller en haut et bas du fichier : haut → gg ; bas → maj + g
- Copier/coller : yy puis p
- Supprimer une ligne (en réalité c'est un couper) : dd
- Remplacer une occurrence dans tout un fichier : :%s/ancien/nouveau/g

# Partie 2 : Installation d'Ubuntu serveur



### Installation Ubuntu serveur :

- Même si la machine est virtuelle, il s'agit de l'installation d'un Linux à part entière.
- La bonne pratique en entreprise veut qu'un serveur Linux soit dédié à une application.
  - Exemple : Jenkins sera installé sur un serveur, Alfresco sur un autre serveur ...
  - Attention de bien différencier application principale et tous les composants nécessaires au bon fonctionnement de cette application.
- Le nom de l'utilisateur principal de la machine est le nom de l'application principale installée
- La bonne pratique veut que le nom du groupe soit identique au nom de l'utilisateur (sauf cas particuliers)
- L'automaticien doit maîtriser l'installation et le paramétrage simple d'un serveur Linux
- Toute installation nécessite une adaptation aux environnements des clients.

- Le point de montage d'un disque Linux est /
- La sous-arborescence est le filesystem
- Le filesystem **est soit organisé en partitions, soit organisé en volumes**
- La zone swap est destinée à la mémoire virtuelle de Linux.
- Une partition sera gérée avec le programme fdisk
- Un volume sera géré avec le programme LVM (logical volume manager)

### Les différents noms de partitions/volumes sous Linux :

➤ hda → hard disk a

➤ hdb → hard disk b

➤ ...

} partitions

➤ sda → sata disk a

➤ sdb → sata disk b

➤ ...

} volumes

➤ D'autres types existent

➤ La commande **fdisk -l** affiche l'ensemble des partitions et volumes

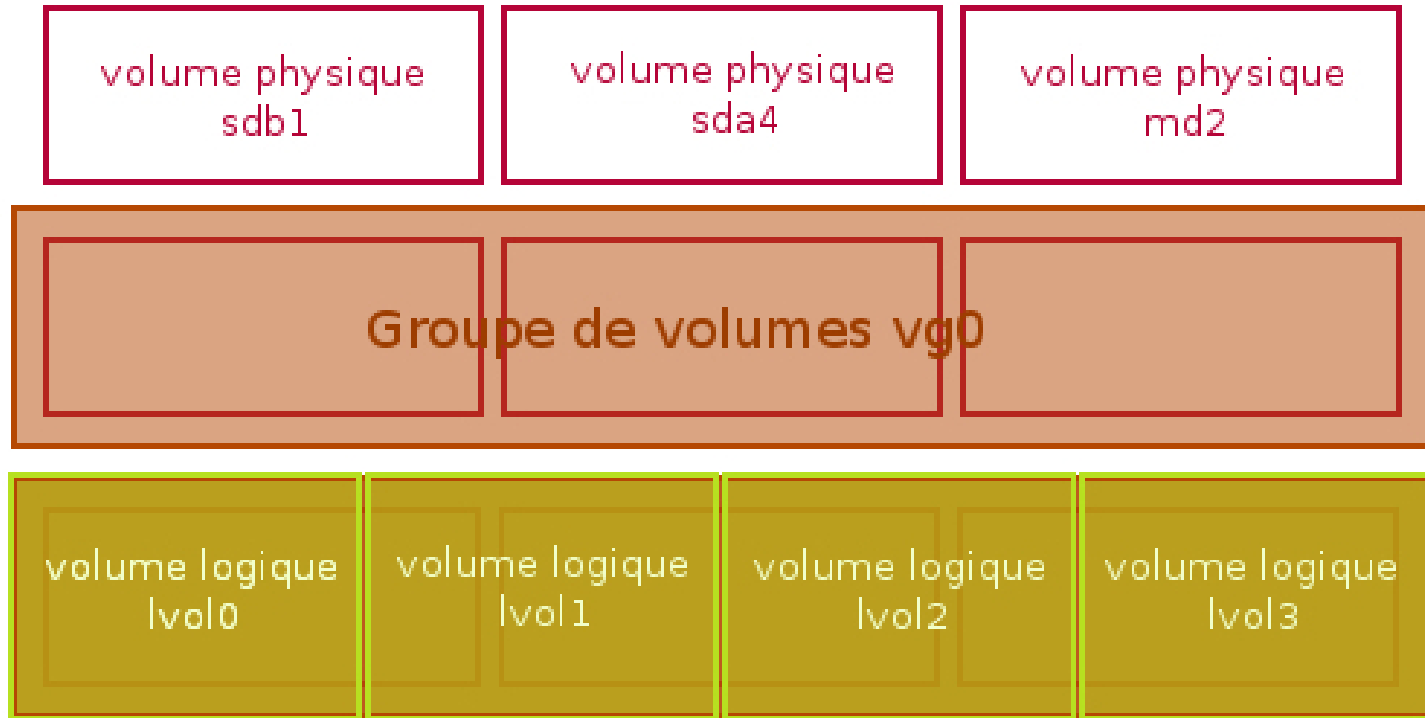
➤ La commande **df -h** indique l'espace disque occupé par volume



**VOLUME PHYSIQUE** : PV (Physical Volume) → un disque physique. Stockage réel « confié » à LVM. ATTENTION : le contenu sera effacé. Cette partie se réalise normalement avant l'installation du système Linux.

**GROUPE DE VOLUMES** : VG (Volume Group) → un ensemble de volume(s) physique(s). L'utilisation de LVM nécessite au moins un groupe de volumes.

**VOLUME LOGIQUE** : VL (Volume Logical) → c'est sur cette partie que nous allons directement agir. Un volume logique est un espace « quelque part dans un groupe de volume » où l'on peut mettre un système de fichiers. C'est donc lui qui remplace les partitions.



- Les commandes s'effectue avec root (sudo -i)
- La commande man -k ^pv indique les commandes utilisables avec LVM
- Les commandes pvcreate vgcreate permettent de créer des volumes physique et des groupes de volumes ( à ne pas utiliser dans le cadre de l'exercice)
- La commande vgdisplay indique les informations sur les volumes

Nom du volume →  
Format du volume →

Taille allouée →  
Taille disponible →

```
--- Volume group ---
VG Name                ubuntu-vg
System ID
Format                 lvm2
Metadata Areas         1
Metadata Sequence No   4
VG Access               read/write
VG Status               resizable
MAX LV                 0
Cur LV                 3
Open LV                 3
Max PV                 0
Cur PV                 1
Act PV                 1
VG Size                 2,82 GiB
PE Size                 4,00 MiB
Total PE                723
Alloc PE / Size        714 / 2,79 GiB
Free PE / Size          9 / 36,00 MiB
VG UUID                g9g1aB-g0VE-0D0Q-czTf-WGJI-3gEA-0q5D6e
```

- L'ajout de nouveau(x) dépôt(s) s'effectue depuis le fichier : `sources.list (/etc/apt)`
- En fonction des versions de Linux, différents dépôts existent. Par exemple : REDHAT → `yum`, DEBIAN → `apt` ...
- **Dépôts officiels :**
  - Sections Main et Restricted, maintenues par les développeurs d'Ubuntu
  - Sections Universe et Multiverse, maintenues par les MOTU
- **Backports** : rétro-portage, anciens dépôts
- **Proposed** : futurs dépôts, déconseillés chez un client
- **Dépôts partenaires et commerciaux** : autres dépôts : `sudo add-apt-repository "deb http://archive.canonical.com/ubuntu $(lsb_release -sc) partner"`
- Il est possible d'ajouter un CDROM, clé USB ou répertoire comme dépôt. Par exemple : `deb file:/votre/chemin/vers/le/dépôt stable main restricted`
- Certaines applications comme synaptic permettent d'installer de nouvelles applications
- Voir le fichier `source.list`

### Principales commandes apt

- apt-get update (ne pas lancer) → resynchronise l'indexation des paquets et dépendances
- apt-get upgrade (ne pas lancer) → installe les versions les plus récentes d'un paquets
- apt-get dist-upgrade (ne pas lancer)
- apt-get install → installe un paquet
- apt-get remove → désinstalle un paquet
- apt-get autoremove → désinstalle les dépendances inutiles
- apt-get purge → désinstalle les paquets et les répertoires résiduels
- apt-get check → vérifie l'existence d'un paquet
- apt-get source → initialise l'ajout de nouveaux canaux de dépendances
- apt-get clean → supprime les paquets résiduels et les paquets archivés
- apt-get autoclean → nettoie le référentiel local
- apt-get changelog → télécharge le journal différentiel d'un paquet

Il est maintenant possible d'utiliser la commande suivante apt install {nom\_paquet} Ubuntu 16)

### Installer une archive .rpm

- `rpm -i nom_du_package.v.i386.rpm`
- `rpm -u nom_du_package.v.i386.rpm` (update)

### Installer une archive .dpkg

- `sudo dpkg -i <paquet.deb>`

### Installer depuis yum

- `yum install {nom_du_paquet}`

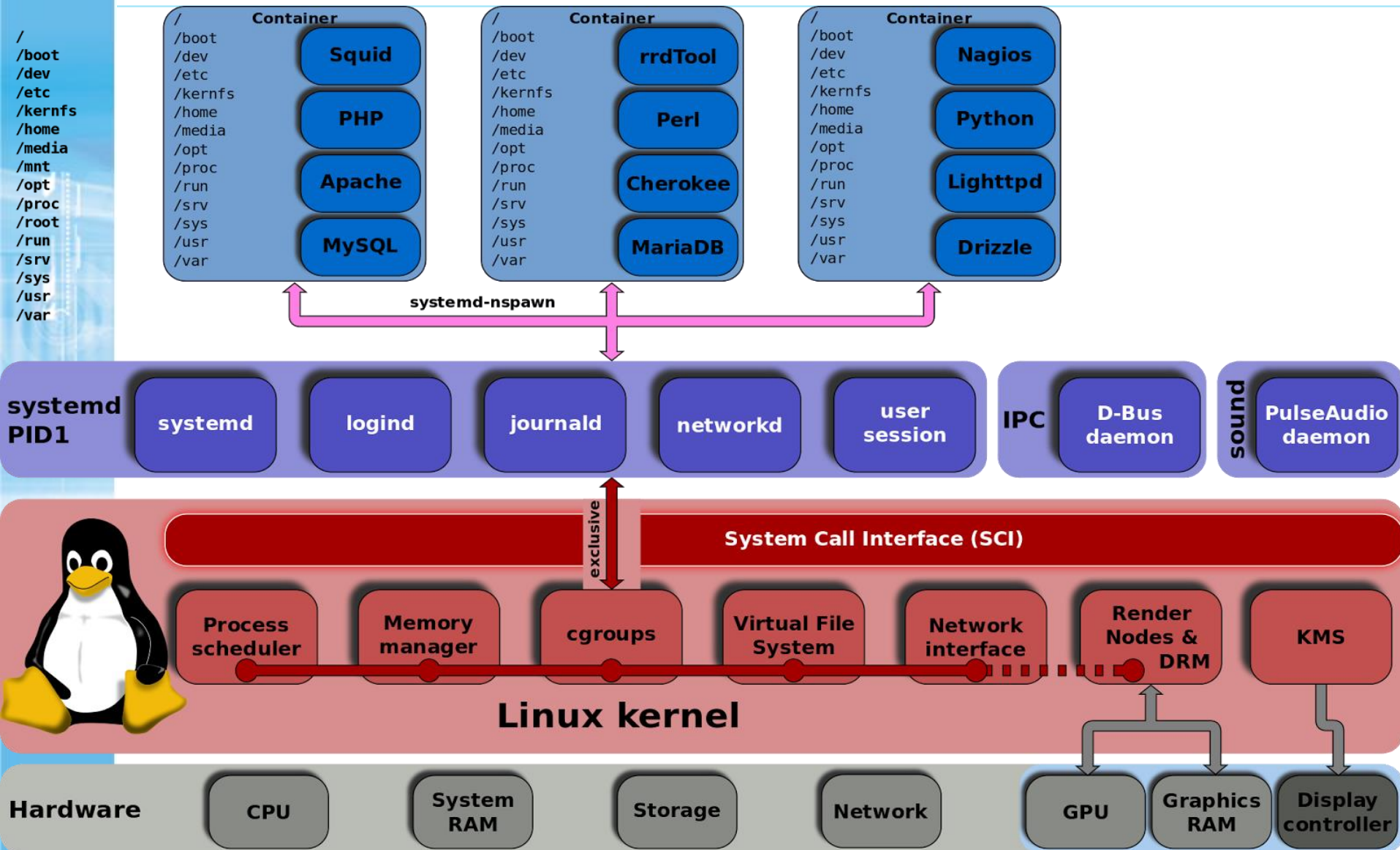
### Installation graphique depuis synaptic (GUI).

### Un programme sous Linux, c'est quoi?

- Un programme sous Linux est un simple fichier texte auquel l'on a donné des droits d'exécution.
- Principe du script (support QL).
- Un processus (ou tâche) est un objet système correspondant à l'environnement d'exécution du programme.
- Le noyau gère plusieurs processus en même temps.
- Les principaux attributs d'un processus sont le PID (Process IDentificator), le PPID (Parent Process IDentificator), le UID (User IDentificator), le GID (Group IDentificator), le TTY (terminal de contrôle du processus), Nice Value (valeur d'importance d'un processus)

# UNIX

## Installation d'Ubuntu serveur : gestion des services (1/3)





**Différentes technologies de services :**

**systemd (ancienne technologie mais encore très présente)**

**systemctl (nouvelle technologie)**

- Les services à lancer se trouvent dans le répertoire `/etc/init.d` (démons)
- La création de nouveaux scripts de démarrage peuvent être placés dans `/etc/init.d` ou dans le répertoire `bin` de votre application.
- Il est possible de paramétrer Linux pour démarrer ou éteindre des services au démarrage de la machine (`/etc/bash.bashrc`). La modification de ce script peut empêcher le démarrage de votre système Linux.
- Exemple lancement/arrêt de service.

**Commandes principales pour la gestion des processus :**

**Commande ps** (process status), affiche les processus

- Syntaxe : ps

**Commande ps -ef**, liste complète des processus (ou ps aux)

- Syntaxe : ps -ef

**Commande ps aux | grep {nom\_process}**, pour retrouver un processus en particulier

- Syntaxe : ps aux | grep {nom\_processus}

**Commande kill**, interrompt un processus

- Syntaxe : kill {id\_process}

**Commande &**, permet de lancer un processus en arrière plan

- Syntaxe : {commande\_que\_vous\_avez\_saisi} &

Les commandes **bg** et **fg** permettent de renvoyer les traitements en arrière ou premier plan

### SSH (serveur)

- SSH → Secure SHell.
- SSH serveur → Indispensable sur le serveur sur lequel vous souhaitez vous connecter
- SSH client → Outils pour se connecter à un serveur Unix distant (exemple ; Cygwin, Putty)
- Installation :
  - Classiquement, on utilise openssh-server et openssh-client
  - Soit suivre un DIE dédié. Soit utiliser un dépôt distant : par exemple sur Ubuntu, possibilité d'installer SSH avec la commande **sudo apt-get install ssh**
  - Vérifier le paramétrage du SSH serveur depuis /etc/ssh/sshd.config
  - Action sur SSH :
    - `service ssh start`
    - `service ssh stop`
    - `service ssh status`
    - `service ssh` (pour obtenir de l'aide sur les combinaisons possibles)
  - Internet regorge de documentations d'installation et de gestion du SSH
  - INFO : Toutes les applications qui utilisent des services fonctionnent avec le même système (systemd et nouvellement systemctl)

### SSH (client) :

- Le fichier de paramétrage est `ssh.config (/etc/ssh)`
- Optimiser sa customisation SSH (dans `/home/{user_name}/.ssh` :
  - le fichier `known_hosts` historise les connexions vers les serveurs distants
  - Se connecter via une clé :
    - Générer une clé : `ssh-keygen -t rsa` (suivre les instructions)
    - Dans `/home/{user_name}/.ssh`, deux clés sont générés : une `id_rsa` (clé privée et une `id_rsa.pub` (votre clé publique). La clé privée est strictement privée, la clé privée est celle que vous envoyée à la machine distante via la commande :
    - `ssh-copy-id -i id_rsa.pub login@ip`
    - Vous pouvez paramétrer l'agent `ssh` (`add-ssh`, nécessite un paramétrage spécifique sous `cygwin`)

### Transférer des fichiers en ligne de commande (avec client SSH) :

**Commande scp**, transfert des fichiers d'une station à une autre station

➤ Syntaxe : `scp {file_name} {user_name}@{ip_destination}:{path}`

`{file_name}` : nom du fichier à transférer

`{user_name}@{ip_destination}` : nom de l'utilisateur qui se connecter et ip de la machine distante

`{path}` : chemin où déposer les fichiers sur la machine distante

Options principales : -r (récursif), -P (spécifier le port) (voir avec man)

**Commande wget**, permet le téléchargement de données depuis une URL

➤ Syntaxe : `wget URL`

### **FTP (serveur) :**

- Le fichier de paramétrage est vsftpd.conf (/etc)

### **FTP (client) :**

- Client FTP (en ligne de commande) : taper la commande ftp dans la console
- Connexion à un serveur FTP : ftp://user:mot-de-passe@mon-site.domaine/mon-repertoire  
OU ftp://user:mot-de-passe@mon-site.domaine:port/mon-repertoire
- Les commandes des transferts et copies sont identiques au Shell

Des interfaces graphiques sont également utilisables.

### Exercice 1 (50 minutes) :

- Lancer VirtualBox
- Installer un nouveau serveur Linux Ubuntu nom : squash\_tm
- Réaliser le paramétrage réseau de la machine virtuelle (depuis VB + adresse ip)
- Réaliser le paramétrage SSH de la machine.
- Réaliser votre paramétrage SSH client (avec une authentification par clé)
- Installer un serveur FTP
- Vérifier l'ensemble de l'installation

Suite **(10 minutes)** : installer le paquet *synaptic* puis le désinstaller. Supprimer les dépendances résiduelles de votre machine.

### Exercice 2 (15 minutes) :

- Installer startx (interface graphique simple) (depuis dépôts apt documentation sur le web - xorg)
- Installer Firefox graphique (depuis dépôts apt)
- Vérifier l'installation des deux composants.

### Exercice 3 (30 minutes) :

- Transmettre via la commande scp l'archive compressée de Squash TM sur votre serveur (possibilité de le télécharger directement depuis internet vers votre serveur avec wget)
- Décompresser l'archive et examiner les répertoires, décrivez l'arborescence de l'application
- Lancer Squash TM depuis son script de démarrage (ne pas oublier le chmod +x) et l'ouvrir depuis Firefox

Rappel : chmod +x → droit relatif d'exécution

**Exercice 4 (5 minutes)** : ajouter le dépôt deb <http://fr.archive.ubuntu.com/ubuntu/> trusty main restricted universe multiverse. Puis exécuter les commandes apt-get clean, apt-get check et apt-get update (avec root)



### Exercice 5 (60 minutes environ) :

- Réaliser un script de mise à jour de votre environnement Linux
- Le script doit : nettoyer les dépôt apt, mettre à jour les dépôt apt, uniquement mettre à jour Tomcat.
- En suivant la documentation <https://doc.ubuntu-fr.org/cron>, paramétrer le planificateur de tâche pour que votre script se déclenche dans 5 minutes. Observez la structure de la documentation d'exploitation cron.
- Une fois réussie, paramétrer votre planificateur pour votre script se déclenche toutes les nuits à 2 heures du matin.
- Conserver le script.

### Exercice 6 (15 minutes) : Exercices sur les commandes utiles Linux (rappel) :

#### En ligne de commande :

Afficher le nom de votre système Linux

Lister les utilisateurs connectés sur la machine Linux

Connectez-vous en root, puis revenez sur le compte de votre utilisateur

Fermer le terminal

Eteindre proprement votre serveur

Afficher le manuel de la commande grep

Lister le contenu (avec les fichiers cachés) du répertoire /var/log

Dans votre répertoire \$HOME, en utilisant le chemin absolu, créer un répertoire tmp\_directory

Afficher dans la console le contenu du fichier /etc/passwd (attention la destruction de ce fichier détruit votre machine)

Trouver les fichiers appelés .profile (avec deux commandes différentes)

Compter le nombre de lignes du fichier vsftpd.conf (chercher la commande à utiliser)

Indiquez le répertoire où vous vous trouvez

Listez les processus tournant

Listez les processus liés à l'application Tomcat

### Exercice 6 (15 minutes) (suite) : Exercices sur les commandes utiles Linux (rappel) :

Dans votre répertoire \$HOME, créer deux répertoires puis les supprimer

Créer un fichier texte puis le copier dans le répertoire \$HOME/tmp\_directory

Créer en une seule ligne de commande trois nouveaux fichiers et les couper/coller dans le répertoire \$HOME/tmpsudo\_directory

Indiquez la taille totale du répertoire tmp\_directory

Supprimer le répertoire tmp\_directory

Indiquer l'espace disque utilisé

Effacer les commandes affichées en console

Exécuter la commande sudo apt-get moo

Installer la commande sl. Exécuter la commande sl -la. Désinstaller la commande sl. Purger la commande sl (normalement aucune purge n'est nécessaire).

# Partie 2 : Système de fichiers



### Exemple d'arborescence d'un système Linux

/ symbolise la racine du système Linux → point de montage

**/bin** → contient les binaires de certaines commandes utiles

**/etc** → contient les fichiers de configuration et les systèmes de bdd

**/dev** → contient des fichiers qui représentent des périphériques physiques

**/home** → **contient les dossiers des utilisateurs (le plus important pour nous)**

**/mnt** → permet de monter un volume (par exemple une clé USB)

**/lib** → contient des librairies système

**/root** → le dossier de l'administrateur système (superutilisateur root)

**/usr** → contient par exemple les applications installées

**/var** → contient des éléments variables et temporaires

➤ Plusieurs arborescences composent Linux mais / est toujours la racine.

### Présentation des répertoires :

**/bin** → contient les binaires de certaines commandes utiles

**/etc** → contient les fichiers de configuration et les systèmes de bdd

**/dev** → contient des fichiers qui représentent des périphériques physiques

**/lib** → contient des librairies système

**/opt**

**/root** → le dossier de l'administrateur système (superutilisateur root)

**/usr** → contient par exemple les applications installées

**/var** → contient des éléments variables et temporaires

**/tmp**

Un lien symbolique c'est quoi?

(rappel : man ln)

### Fichiers de configuration network

➤ /etc/network (ne pas toucher)

### Rappel sur l'utilisation de certaines commandes :

La commande **ifconfig** indique les paramètres réseaux de la station

**Exercice 1 (5 minutes)** : taper ifconfig dans la console : trouver l'ip de votre machine

La commande **ping** permet de vérifier la communication avec une machine distante.

**Exercice 2 (5 minutes)** : ping {*adresse\_ip\_de\_la\_machine\_distante*}

Depuis votre machine Linux, vérifier la communication avec votre station locale (pour connaître l'ip de votre station Windows, lancer cmd puis taper ipconfig)

Commande changement utilisateur : `su - {nom_utilisateur}`

### Fichiers de configuration utilisateurs

**passwd :**

- **nom\_utilisateur** correspond au login de l'utilisateur.
- **mot\_de\_passe** correspond au mot de passe de l'utilisateur remplacer par un x pour des raisons de sécurité.
- **uuid** correspond à l'identifiant système de l'utilisateur
- **guid** correspond au groupe principal de l'utilisateur.
- **commentaire** correspond à un commentaire textuel sur l'utilisateur qui est souvent son nom réel (Prénom et Nom).
- **home** correspond au répertoire home de l'utilisateur sur ce système.
- **shell** correspond à l'interpréteur shell par défaut de l'utilisateur.
- **exemple** : `nom_utilisateur:mot_de_passe:uuid:guid:commentaire:home:shell`

**adduser.conf** : paramétrage général des comptes utilisateurs

**ftputils** : accès FTP par les utilisateurs

**deluser.conf** : conditions de suppression des utilisateurs

En fonction des applications installées, de nouveaux fichiers de configuration peuvent s'ajouter.



### Principales variables d'environnement.

Le résultat d'une variable peut être affiché en console avec la commande **echo** (exemple :  
echo \$?)

Les variables d'environnement sont très utilisées en scripting pour les rendre multi-machines

**\$HOME** → home de l'utilisateur courant

**\$LOGNAME** → nom de l'utilisateur connecté

**\$OLDPWD** → ancien rép.

**\$PATH** → Liste des noms de répertoires dans lesquels le système cherche les commandes

**\$PWD** → nom du répertoire courant

**\$RANDOM** → nombre aléatoire

La commande **env** liste les variables d'environnement de l'utilisateur connecté.

La commande **set** affiche toutes les variables de l'environnement.

La commande **export** permet d'ajouter de nouvelles variables d'environnement.

### Partie 3 : Gestion des droits



### Linux fonctionne sur un système de droits (autorisations)

**Droits** d'accès aux fichiers et dossiers (souvenez-vous, sous Linux tout est fichier!)

- Les droits de lecture (**read**) --> lire un contenu
- Les droits d'écritures (**w**rite) --> écrire un contenu dans un fichier
- Les droits d'exécution (**e**xecute) --> exécuter un contenu (par exemple script)

### Règles sur les droits

- Ces droits sont applicables à tous les utilisateurs (même root), aux groupes et à des utilisateurs extérieurs
- Un système Linux possède des droits généralisés à tout le système, c'est le umask (commande umask)

## Gestion des droits : Gérer les droits sur les fichiers et répertoires (2/3) (rappel)

La commande `ls -l` affiche les différents droits des fichiers et répertoires

- ✓ La 1<sup>ère</sup> colonne indique le type de fichier : '-' fichier, 'd' directory, 'l' link
- ✓ Nous retrouvons rwx pour read, write, execution (utilisateurs, groupes, les autres)

-rwxr-xr-x	1	postel	None	1,9K	5 déc. 13:24	.inputrc
-rw-----	1	postel	None	37	8 déc. 15:13	.lessht
-rw-r--r--	1	postel	None	28	6 déc. 08:26	.minttyrc
-rwxr-xr-x	1	postel	None	1,3K	5 déc. 13:24	.profile
drwxr-xr-x+	1	postel	None	0	6 déc. 11:56	.vim

Droits lecture:écriture/exécution pour utilisateur (ex : vous)

Droits lecture:écriture/exécution pour groupe

Droits lecture:écriture/exécution pour les autres

Nom du propriétaire du fichier/répertoire

Groupe d'appartenance du propriétaire

Taille du fichier/répertoire

Date de création du fichier/répertoire

Nom du fichier/répertoire

**.vim est un répertoire, l'utilisateur à tous les droits, le groupe et les autres peuvent l'ouvrir**

# UNIX

## Gestion des droits : Gérer les droits sur les fichiers et répertoires (3/3)

La commande **chmod** s'utilise de manière « relative » ou « absolue »

### Manière relative :

✓ Modifie **simplement** des droits **tout en conservant les anciens droits**

#### non modifiés

- u : user
- g : group
- o : other
- a : all
- r : read
- w : write
- x : execute

#### Exemples :

- a. Donner au groupe des droits d'exécution : `chmod g+x nom_fichier`
- b. Donner au groupe des droits d'écriture et retirer aux autres des droits de lecture : `chmod g+w o-r nom_fichier`
- c. Donner à tout le monde tous les droits : `chmod a+rwx nom_fichier`

### Manière absolue :

✓ **Ecrase les droits existants** par de **nouveaux droits**

✓ La manière « absolue » est basée sur une conversion binaire/décimale

✓ Par exemple : je donne tous les droits à user, droits de lecture à group et droits de lecture aux autres

✓ User : `rwx` → 111 → 7

✓ Group : `r--` → 100 → 4

✓ Other : `r--` → 100 → 4

Dec	Bin	Droits
1	001	x
2	010	w
3	011	wx
4	100	r
5	101	rx
6	110	rw
7	111	rwx

✓ Soit : `chmod 744 nom_fichier`  
Module 2 : UNIX

### Utilisateurs et groupes (rappel)

- Les utilisateurs standards → vous, votre voisin, une autre personne, une application qui simule un utilisateur factice.
- Le superutilisateur root → il est l'administrateur du système Linux. Par exemple, il gère l'accès à certains répertoires et fichiers, il gère la configuration du système, il ouvre le système Linux sur d'autres systèmes...
- Les groupes → certains utilisateurs standards sont affiliés à des groupes d'utilisateurs.



« L'utilisateur root » possède tous les droits même celui de détruire le système Linux. Il est fortement recommandé de ne l'utiliser que pour des tâches d'administration système. Toutes autres tâches doivent être réalisées avec un utilisateur dédié!!!

### Commandes principales gestion utilisateurs et groupes :

**Commande sudo**, se connecter avec le superutilisateur root

- Syntaxe : `sudo -i`

Les manipulations qui nécessitent les droits superutilisateur sont précédées de la commande `sudo`.

**Commande su**, se connecter avec un utilisateur

- Syntaxe : `su - nom_utilisateur`

### Commandes principales de gestion utilisateurs et groupes :

**Commande chown** (change the owner), modifie l'utilisateur et le groupe qui possède le fichier ou le répertoire

- Syntaxe : `chown utilisateur:group nom_du_dossier_OU_fichier`
- Principales options
  - `-R` → récursif

**Commande adduser**, crée un nouvel utilisateur

- Syntaxe : `sudo adduser {nom_utilisateur}`

**Commande addgroup**, crée un nouveau groupe

- Syntaxe : `sudo addgroup {nom_utilisateur}`

**Commande chgrp** (change group), change le groupe auquel appartient le fichier

- Syntaxe : `sudo chgrp {nom_fichier}`



### Fichier de configuration des groupes :

**group** : regroupe les différents groupes existants

Par exemple : lpadmin:x:110:ubuntu

- nom du groupe, mdp, id système et liste utilisateurs rattachés au groupe

### Fichier de configuration sudoers :

**sudoers** : spécifie les privilèges utilisateurs et groupes (dont le superutilisateur). Les modifications sur ce fichier peuvent détruire la machine ou créer des failles de sécurité. Sécurise les accès à certains répertoires (NE PAS TOUCHER).

# Partie 3 : Les flux

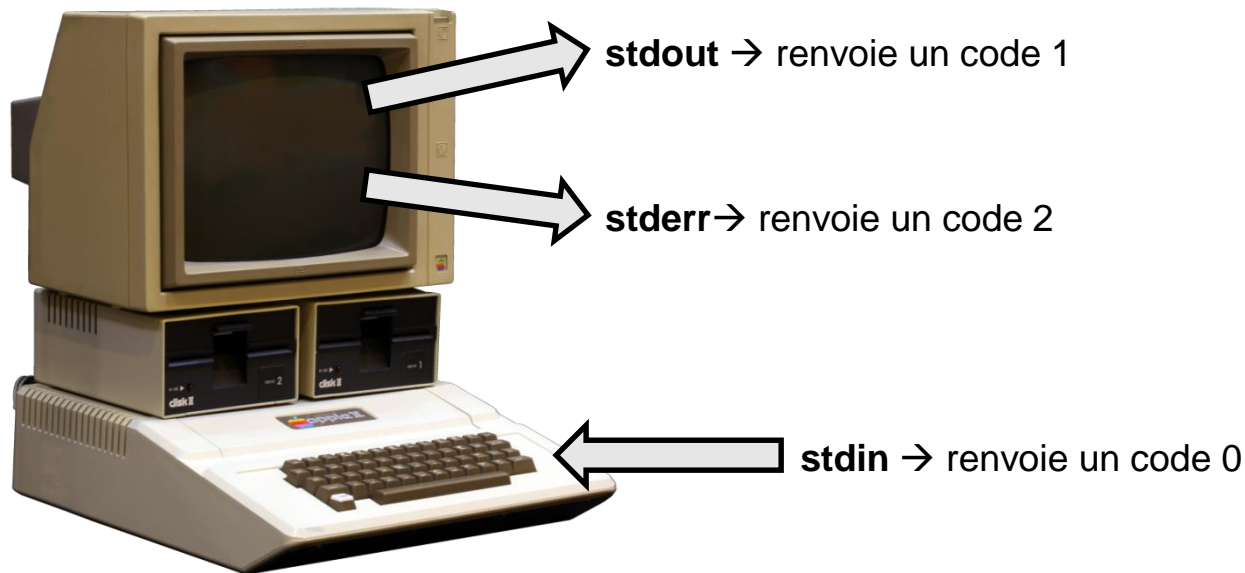


# UNIX

## Les flux : entrées et sorties de processus (1/2) (rappel)

Chaque commande possède trois flux standards pour communiquer avec l'utilisateur

- l'**entrée** standard nommée **stdin** (identifiant 0) : il s'agit par défaut du clavier
- la **sortie** standard nommée **stdout** (identifiant 1) : il s'agit par défaut de l'écran
- la **sortie** d'erreur standard nommée **stderr** (identifiant 2) : il s'agit par défaut de l'écran




- Lorsque l'on exécute une commande, le shell redirige le résultat de la commande à l'écran :

ls -l ➡

```
postal@UTILISA-ITFKLNR ~  
$ ls -l  
total 0  
-rwxrwx-wx 1 postal None 0  9 déc. 14:08 tmp  
postal@UTILISA-ITFKLNR ~  
$
```

- Il est possible de rediriger le résultat de la commande dans un fichier en utilisant le signe ➤

ls -l ➤ result ➡  ➡  
total 0  
-rwxrwx-wx 1 poste1 None 0 9 déc.  
14:08 tmp

- L'intérêt est par exemple de conserver une trace du résultat de la commande dans un fichier texte persistant.

## Les flux : redirection et concaténation de flux dans un fichier (rappel)

### ➤ Rediriger la sortie standard :

*commande* > resultat

Exemple : `ls > resultat`

### ➤ Rediriger la sortie standard en identifiant le flux à rediriger

*commande* 1 > resultat

Exemple : `ls 1 > resultat`

### ➤ Rediriger la sortie d'erreur standard

*commande* 2 > resultat

Exemple : `ls 2 > resultat`

### Rediriger un résultat vers /dev/null (« trou noir »)

*commande* > /dev/null

### ➤ Concaténer les redirections → utilisation du symbole >>

*commande* >> resultat

Exemple : `ls >> resultat`

- Il est possible de rediriger la sortie d'erreur vers la sortie standard avec la commande `2>1`
- Il est possible d'ajouter la commande de bg d'un service après une redirection.

Par exemple : `2>1 &` ou encore `> ma_log.log &`

### Exercice (facultatif car déjà fait) :

1. Dans votre \$HOME (à vérifier), créer un répertoire tmp\_directory et ajouter la commande echo pour indiquer la création du répertoire.
2. Rediriger la création du répertoire dans un fichier creation.txt
3. Concaténez le résultat de la commande `cal` dans le fichier creation.txt . Sur la même ligne, réorientez la sortie d'erreur vers /dev/null.



### Partie 4 : Manipulations sur les fichiers



## Manipulations sur les fichiers : grep

**Commande grep**, affiche une ligne de correspond à un pattern (soit dans un fichier, soit dans le résultat d'une ligne de commande

- Syntaxe : `grep pattern nom_de_mon_fichier`
- Exemple : `grep bash tmp.txt` → affiche en console le pattern bash présent dans un fichier nommé tmp.txt
- Expressions régulières avec grep (avec l'option -E) :

### Caractère spécial

### Signification

.	Caractère quelconque
^	Début de ligne
\$	Fin de ligne
[]	Un des caractères entre les crochets
?	L'élément précédent est optionnel (peut être présent 0 ou 1 fois)
*	L'élément précédent peut être présent 0, 1 ou plusieurs fois
+	L'élément précédent doit être présent 1 ou plusieurs fois
	Ou
()	Groupement d'expressions



[a-z]	Caractères minuscules de a à z.
[0-9]	Chiffres de 0 à 9.
[a-e0-9]	Lettres de « a » à « e » ou chiffres de 0 à 9.
[0-57A-Za-z.-]	Chiffres de 0 à 5, ou 7, ou lettres majuscules, ou lettres minuscules, ou un point, ou un tiret.
guitare	Cherche le mot « guitare » dans la chaîne.
guitare piano	Cherche le mot « guitare » OU « piano ».
^guitare	La chaîne doit commencer par « guitare ».
guitare\$	La chaîne doit se terminer par « guitare ».
^guitare\$	La chaîne doit contenir uniquement « guitare ».

**Commande cat**, affiche le contenu d'un fichier dans la console

- Syntaxe : `cat nom_de_mon_fichier`

**Commande tail**, affiche les dernière ligne d'un fichier dans la console. Utile pour suivre en temps réel l'écriture de logs

- Syntaxe suivre écriture de logs: `tail -f nom_de_mon_fichier`
- Afficher les 10 dernières ligne d'un fichier dans la console : `tail -n10 nom_de_mon_fichier`

De nombreuses autres options sont disponibles (rappel commande `man tail`)

**Commande head**, afficher les premières lignes d'un fichier dans la console

- Syntaxe : `head nom_de_mon_fichier`

# UNIX

## Manipulations sur les fichiers : cut, join, sort

**Commande cut**, affiche des zones spécifiques d'un fichier

- Syntaxe : `cut`
- `cut -c 1-5 {nom_fichier}` → conserve les caractères 1 à 5
- `cut -c -3 {nom_fichier}` → conserve du premier au troisième caractère
- `cut -c 3- {nom_fichier}` → conserve du troisième au dernier caractère
- `cut -d':' -f 6 /etc/passwd` → sélectionne la 6<sup>ème</sup> colonne délimitée par :

**Commande uniq**, supprime les doublons dans un fichier

- Syntaxe : `uniq nom_de_mon_fichier`

**Commande sort**, trie les données dans un fichier

- Syntaxe suivre écriture de logs: `sort nom_de_mon_fichier`
- Trier des nombres: `sort -n nom_de_mon_fichier`

De nombreuses autres options sont disponibles

Voir commande join pour fusionner des fichiers

**Commande diff**, compare deux fichiers

- Syntaxe : `diff {nom_fichier_1} {nom_fichier_2}`

**Commande sed**, affiche une ligne de correspond à un pattern (soit dans un fichier, soit dans le résultat d'une ligne de commande).

- Afficher un fichier dans la console : `sed " mon_fichier`
- Adressage par ligne : suppr de lignes : `sed '4,7 d' mon_fichier`
- Adressage par motif : suppr par motif : `/ma_regex/ d' test.txt`
- Exemple : `sed '/^Bonjour/,/^Au revoir/d' mon_fichier` supprimera toutes les lignes comprises entre 'Bonjour' et 'Au revoir'.
- Remplacement : `sed -re s/motif/substitut/` et `sed -re s/motif/substitut/g`
- Le motif peut intégrer des regex
- Pour les regex voir memento sur Internet : <https://openclassrooms.com/courses/concevez-votre-site-web-avec-php-et-mysql/memento-des-expressions-regulieres>

Pour aller plus loin, voir commande awk (langage à part entière)

**Commande tar**, archive et compresse des répertoires

➤ Syntaxe : `tar -options nom_de_larchive nom_répertoire`

➤ **Création d'une archive**

`tar -cf archive.tar nom_répertoire`

➤ **Création d'une archive compressée**

`tar -zcf archive.tar.gz nom_répertoire`

➤ **Décompression d'une archive**

`tar -xzf archive.tar.gz nom_répertoire`

➤ Principales options

- -v → mode verbose, affiche le détail de la compression/décompression/archive

Il est possible de créer des archives avec zip et autre(s)

### **Exercice (entre 5 et 10 minutes) : Création et récupération d'une archive tar.gz**

- 1) Créer un répertoire contenant 3 fichiers différents
- 2) Créer une archive tar de votre répertoire
- 3) En plus, créer une archive compressée tar.gz de votre répertoire
- 4) Avec la commande scp, récupérer l'archive tar.gz sur le poste local
- 5) Décompresser l'archive dans le \$HOME de votre environnement Cygwin

### Partie 5 : Scripting shell : les tests (rappel)



### Tester une variable :

Syntaxe : [ \$a = toto]

### Tests sur les objets du système de fichier :

[ -e \$FILE ] → vrai si l'objet désigné par \$FILE existe dans le répertoire courant

[ -s \$FILE ] → vrai si l'objet désigné par \$FILE existe dans le répertoire courant et si sa taille est supérieure à zéro

[ -f \$FILE ] → vrai si l'objet désigné par \$FILE est un fichier dans le répertoire courant

[ -r \$FILE ] → vrai si l'objet désigné par \$FILE est un fichier lisible dans le répertoire courant

[ -w \$FILE ] → vrai si l'objet désigné par \$FILE est un fichier inscriptible dans le répertoire courant

[ -x \$FILE ] → vrai si l'objet désigné par \$FILE est un fichier exécutable dans le répertoire courant

[ -d \$FILE ] → vrai si l'objet désigné par \$FILE est un répertoire dans le répertoire courant.



## Scripting shell : les tests (2/3)

### Tester des chaînes de caractère :

- [ c1 = c2 ] → vrai si c1 et c2 sont égaux
- [ c1 != c2 ] → vrai si c1 et c2 sont différents
- [ -z c ] → vrai si c est la chaîne vide (*Zero*)
- [ -n c ] → vrai si c n'est pas la chaîne vide (*Non zero*)

### Tests sur les nombres :

- [ n1 -eq n2 ] → vrai si n1 et n2 sont égaux (*Equal*)
- [ n1 -ne n2 ] → vrai si n1 et n2 sont différents (*Not Equal*)
- [ n1 -lt n2 ] → vrai si n1 est strictement inférieur à n2 (*Less Than*)
- [ n1 -le n2 ] → vrai si n1 est inférieur ou égal à n2 (*Less or Equal*)
- [ n1 -gt n2 ] → vrai si n1 est strictement supérieur à n2 (*Greater Than*)
- [ n1 -ge n2 ] → vrai si n1 est supérieur ou égal à n2 (*Greater or Equal*).

### Tester le code retour de la dernière commande (gestion d'erreur) :

- [ \$? -gt 0 ] → la variable \$? retourne code retour de la dernière commande exécutée. Si le code retour est strictement supérieur à 0, alors il y a une erreur. Si le code retour est égal à 0 alors la dernière commande est correcte.

### Expressions régulières et tests :

ATTENTION, les syntaxes peuvent changer selon les environnements, comportements très complexes

Exemple de code qui teste une expression régulière :

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

```
read a
```

```
if [[ $a = [0-9] ]]
```

```
then
```

```
    echo OK
```

```
else
```

```
    echo KO
```

```
fi
```

**Exercice 1 (10 minutes)** : Créer un script qui propose à un utilisateur de créer le répertoire de son choix dans ~. Si le répertoire existe, avertir l'utilisateur, si le répertoire n'existe pas, le créer. Mettre en place une gestion d'erreur et une log.

**Exercice 2 (10 minutes)** : Créer un script qui vérifie si un numéro de téléphone est valide (0xxxxxxxxx). Vous utiliserez des expressions régulières

**Exercice 3 (20 minutes)** : Reprendre le même exercice mais cette fois avec deux possibilités (0xxxxxxxxx) ou (+330xxxxxxxxx)

### Exercice 4 (20 minutes) : Expliquer le code suivant :

```
dir()                                # fonction dir (affichage d'un repertoire)
{
```

```
case $# in
0) rep=`pwd`;;
*) rep=$*;;
esac
```

```
for f in $rep
do
```

```
if [ -d $f ]                        # est ce un repertoire ?
then
```

```
if [ -r $f -a -x $f ]              # lisible et traversable ?
then
```

```
ls -la $f
```

```
else
```

```
echo "Le repertoire $f est inaccessible" 1>&2;
```

```
fi
```

```
# sinon on teste si l'element existe et est accessible
# en testant le resultat de ls -l
```

```
else
```

```
r=`ls -l $f 2>/dev/null`
```

```
if [ "$r" ]
```

```
then
```

```
echo $r
```

```
else
```

```
echo "L'element $f est inaccessible" 1>&2;
```

```
fi
```

```
fi
```

```
done
```

```
}
```

### Partie 6 : Scripting shell avancé (difficile)



Pré-requis : Revoir les conditions et boucles sur le support Shell de la formation QL :

Présentation et lecture d'un script shell rédigé dans les règles de l'art

**Exercice** : Lire le script. Comprendre et noter tous les points qui en font un bon script

**\$#** : nombre de variables d'arguments utilisées

**\$?** : code retour de la dernière commande utilisée

**\$1** : argument 1

**\$2** : argument2 (jusqu'à 9 arguments différents)

```
hello_world() {  
  figlet $1  
}
```

Dans cet exemple, l'argument placé derrière hello\_world sera affiché en console.

```
while getopts m:y:f:b:l:p:vu OPT
do
    case ${OPT} in
        m) # Set the month of the dataset
            ARG_DATA_MONTH="`echo ${OPTARG} | awk '{printf("%02d", $1)}'`"
            ;;
        y) # Set the year of the dataset
            ARG_DATA_YEAR="`echo ${OPTARG} | awk '{printf("%04d", $1)}'`"
            ;;
        *) # Print usage in others cases and exit false
            _Usage 2 #Usage est ici une fonction déclarée précédemment dans le script
            ;;
    esac
done
```



### Créer une fonction :

- Une fonction sert à regrouper du code par thématique afin d'être ensuite réutilisé
  - Une fonction marche deux temps : je déclare ma fonction puis j'appelle ma fonction
  - L'écriture d'une fonction suit la syntaxe suivante : *nom\_de\_la\_fonction()* { *code\_shell* }
- #!/bin/bash

```
#Je déclare ma fonction
figlet_install() {
dpkg -s figlet | grep Status
if [ $? = 0 ]; then echo "figlet est installé"
else
    sudo apt-get install figlet
fi
}
```

```
hello_world() {
figlet $1
}
```

```
#J'appelle ma fonction
figlet_install
hello_world hello_world
```

**Ecrire le script et l'essayer**

### Niveau simple (40 minutes environ) :

**Exercice 1** : Créer un script qui indique si l'argument indiqué est un fichier, un répertoire ou un lien symbolique. Si aucun argument n'est indiqué, le script demande un argument.

**Exercice 2** : Créer un script qui liste l'ensemble des utilisateurs et leur groupe d'appartenance dans un fichier texte. L'utilisateur peut choisir le nom du fichier texte

**Exercice 3** : Créer une boucle for qui liste tous les fichiers contenus dans votre répertoire \$HOME

**Exercice 4** : Créer un script compte le nombre de répertoire(s) et fichier(s) présent dans un répertoire choisi par l'utilisateur

### Niveau moyen (1 heure environ) :

**Exercice 4 :** Créer une calculatrice en shell. Indications : utilisation de la commande `let` et de la boucle `case`. Vous créerez une fonction `saisir` et une fonction `calcul`. Ces fonctions seront utilisées dans un algorithme final.

**Exercice 5 :** Créer un script dont le menu permet : Afficher l'heure, de créer un nouveau répertoire dont le nom peut être choisi, de créer une archive `tar.gz` d'un fichier, de déplacer une archive dans un répertoire choisi, de lister le contenu de l'archive, de décompresser l'archive dans le répertoire de son choix, de sortir du menu. Le code peut être écrit en `korn shell`.

### Partie 7 : Intégration d'une application sous Linux



### Test d'intégration

- Intégration manuelle
- Intégration continue
- Le rôle du testeur
- Le processus du test d'intégration manuel
- L'intégration automatisée

Exemple de documents pouvant être fournis pour une recette d'intégration :

- **DIE** (Document d'Installation et d'exploitation)
- **DAT** (Document d'Architecture technique)
- **DEX** (Document d'Exploitation)

### Mini-projet :

Installation d'un serveur d'intégration continue : Intégration de l'application Jenkins

L'énoncé sera fourni le moment venu!

# Annexes





**Il est possible d'effectuer deux commandes DIFFERENTES sur une même ligne en utilisant le symbole ;**

- Syntaxe : `commande_1;commande_2`
- Exemple : `ls;pwd`

**Il est possible de lier l'action de deux commandes en effectuant un tube (pipe)**

**L'action redirige le résultat d'une commande vers l'entrée de la commande suivante**

- Syntaxe : `commande_1|commande_2`
- Exemple : `ps aux | grep bash`

L'exemple ci-dessus liste les processus auxiliaires de votre système et affiche les lignes qui correspondent au pattern « bash ». Tester la commande.

**Il est possible d'enchaîner deux commandes à la suite si et seulement la première commande est un succès complet**

- Syntaxe : `commande_1 && commande_2`
- Exemple : `pwd && exit`

L'exemple ci-dessus indique le répertoire dans lequel vous vous trouvez puis quitte la console Cygwin. Effectuer la commande suivante `pwd && exit` et observez le résultat.