Module 02 : Installation, configuration d'une machine sous GNU/Linux (Debian) et premiers pas {.unnumbered .unlisted}

- **Mise en place** de l'environnement de test ;
- **Rappel** des bases ;
- Philosophie d'UNIX **en pratique** ;
- Être **à l'aise dans le shell**, sur une **distribution Debian** et savoir **agir en suivant les principes d'UNIX**.

Créer une machine virtuelle Debian

- Installer Oracle VM Virtualbox ou équivalent
- Télécharger l'image de Debian (amd64)
- Créer une machine virtuelle nommée `server` sous Debian (10 GiO)

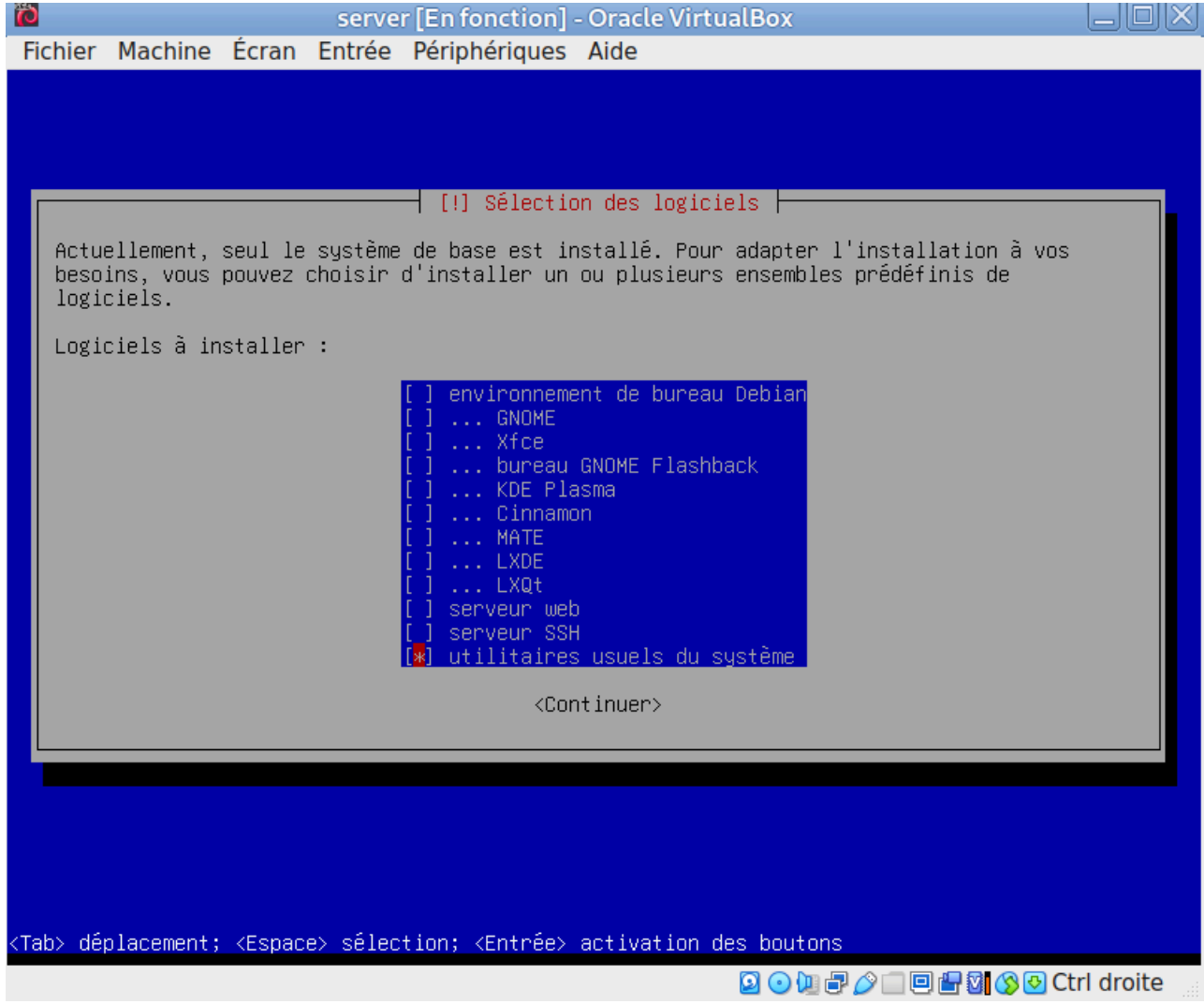
Créer une machine virtuelle Debian (rôle serveur)

{.marp-bg-img}

Important

- **Cocher "Skip Unattended Install"** ;
- Démarrer la machine virtuelle, choisir **Install** (et non Graphic Install) ;
  - Créer un mot de passe `root`, **conservez bien le mot de passe root** (dans un gestionnaire de mot de passe),
  - Créer un utilisateur et mot de passe,
  - **Ne pas installer d'environnement graphique**,
  - **Ne pas installer de serveur SSH ou web**.

> Consulter le manuel utilisateur d'Oracle Virtual Box



Informations sur la distribution installée

- Connaître la version du noyau Linux :

```
man uname
uname -a
```

- Connaître sa version de Debian, son cycle de développement et sa philosophie
- Connaître sa distribution :

```
cat /etc/issue
```

Paquets et gestionnaire de paquets `apt`

- APT (Advanced Package Tool) est le gestionnaire de paquets principal sur Debian et dérivés
- Utilisé pour **installer**, **mettre à jour** et **gérer** les logiciels.
- Vérifie et **résout les dépendances** automatiquement ;

Commande principales apt : inspecter

> Toujours : **inspecter**, faire/défaire

Inspecter :

- `show` - **affiche** les détails du paquet
- `search` - **cherche** un paquet (recherche sur nom et description du paquet)
- `list` - **liste** les paquets disponibles depuis les dépôts
- `list --upgradable` - **liste** les paquets pouvant être mis à jour
- `list --installed` - **liste** les paquets installés

```
apt show openssh-server
apt search openssh
apt list --installed
#Afficher le nombre de paquets installés
apt list --installed | wc -l
```

> `wc` (*word count*, `man wc`) pour compter les lignes, mots et bytes dans un fichier

Commande principales apt : agir

> Toujours : inspecter, **faire/défaire**

Agir :

- `update` - met à jour **la liste des paquets** disponibles
- `upgrade` - **met à jour le système** en installant/mettant à jour les paquets
- `install` - installes les paquets
- `remove` - supprime des paquets
- `reinstall` - reinstall packages
- `autoremove` - automatically remove all unused packages
- `full-upgrade` - met à jour le système en supprimant/installant/mettant à jour les paquets

Gérer les dépôts apt

> Voir cours dev natif pour plus de détails sur l'architecture des dépôts apt

Fichier principal :

```
/etc/apt/sources.list
```

Voir la liste des dépôts :

```
cat /etc/apt/sources.list
ls /etc/apt/sources.list.d/
```

Gérer les dépôts apt : ajouter un dépôt (source de paquets, distributeur)

Pour ajouter un dépôt, **ajouter une ligne** dans `sources.list` **ou créer un fichier** `.list` (reco) dans `/etc/apt/sources.list.d/` :

```
deb http://deb.debian.org/debian bookworm main contrib non-free
```

- `main` : **paquets libres et entièrement conformes aux standards Debian**
- `contrib` : paquets libres dépendant de logiciels non libres
- `non-free` : paquets qui ne respectent pas les critères Debian, souvent propriétaires (et souvent nécessaires !)

Mettre à jour (recharger) la liste des paquets :

```
sudo apt update
```

Signature

Attention, lors de l'ajout d'un dépôt APT, le système demandera à pouvoir vérifier la signature des paquets pour garantir leur authenticité et éviter l'installation de logiciels non fiables. Il faudra donc importer **la clé GPG** du dépôt ou indiquer son emplacement pour que APT puisse effectuer cette vérification. (Voir cours Dev natif et démo publication avec apt)

1. Récupérer et copier la clé publique du distributeur :

```
sudo cp depot.gpg /usr/share/keyrings/
```

2. L'associer au dépôt enregistré :

```
#Association clef pour vérification auto par apt
deb [signed-by=/usr/share/keyrings/depot.gpg] http://mon.depot/debian bookworm main
```

Gérer les dépôts apt : supprimer un dépôt (source de paquets, distributeur)

Supprimer un dépôt :

```
sudo rm /etc/apt/sources.list.d/mon_depot.list
sudo apt update
```

Installer le serveur ssh (openssh-server)

En *su* :

```
apt show openssh-server
apt install openssh-server
#Démarrer le serveur
systemctl start ssh
#Vérifier que le serveur ssh est actif
systemctl status ssh
```

Configuration réseau de la machine virtuelle

Via NAT et redirection de Port (*Port Forwarding*) :

- Aller dans Configuration VM > Réseau (Vue avancée) ;
- Redirection des ports :
  - Port hôte (de la machine hote) un port libre, `2222` par exemple,
  - Port VM `22` (port réservé au protocole SSH par l'Internet Assigned Numbers Authority(IANA),
  - IP invité (laisser vide).

> Le réseau NAT (*Network Address Translation*) est une méthode utilisée pour permettre à une machine virtuelle (ou tout autre appareil) de communiquer avec l'extérieur (Internet ou d'autres machines) en utilisant l'adresse IP de l'hôte, tout en gardant son adresse IP privée. À titre d'exemple, c'est cette configuration réseau qui est utilisée par défaut par Docker pour ses conteneurs (*bridge mode*).

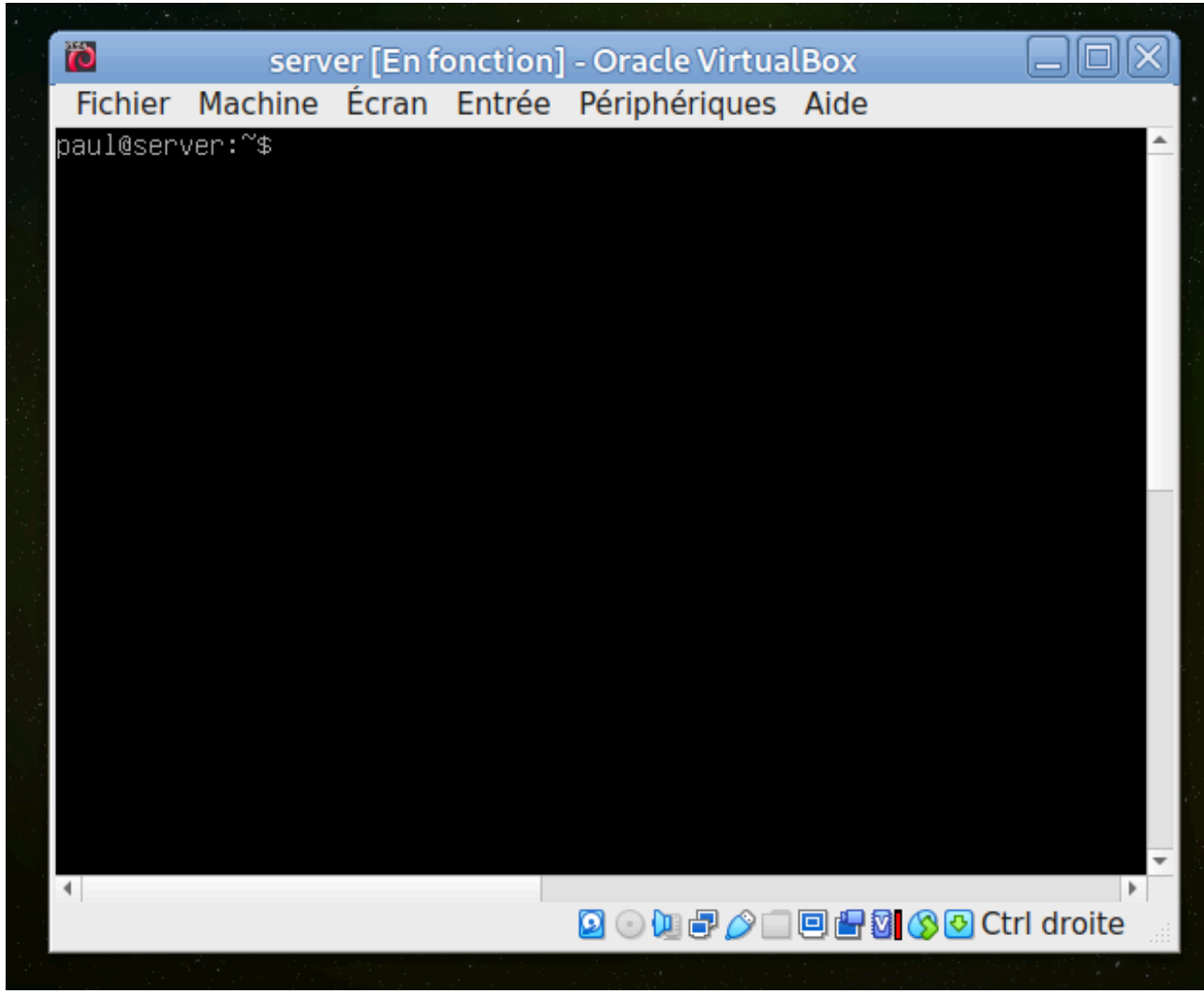
Communiquer avec la machine virtuelle depuis votre client

{.marp-bg-img}

Configuration réseau via la méthode NAT et le *Port Forwarding* (Configuration de la VM).

Après installation de SSH sur `server`. Depuis la machine hôte :

```
#Depuis la machine hôte
ssh -p 2222 user@localhost
```



Lancer et gérer la VM sans interface graphique (nous sert plus à rien !)

Avec Oracle VirtualBox, on peut lancer une VM en mode *headless* avec la commande suivate :

```
#Depuis la machine hôte
#Lancer en mode headless
vboxmanage startvm "nom de votre vm" --type headless
#Lister les vm en cours d'execution
vboxmanage list runningvms
#Arrêt propre
vboxmanage controlvm "nom de votre vm" acpipowerbutton
#Shutdown
vboxmanage controlvm "nom de votre vm" poweroff
```

S'ajouter dans le groupe *sudo users*

Sur Debian :

```
su
apt install sudo
usermod -aG sudo <votre user>
# Se reconnecter pour voir les changements
su - <votre user>
# Checker
groups
sudo -v
```

Lister les users

Base de données clés/valeurs chiffrée :

```
cat /etc/passwd
```

Nettoyer les lignes avec `cut` :

```
man cut
```

> `q` pour quitter le manuel. Puis `Ctr+L` pour nettoyer l'écran (ou `clear`).

```
cat /etc/passwd | cut -d: -f1
#Lister que les users "réels"
grep '/home' /etc/passwd | cut -d: -f1
```

Le système de fichiers UNIX

```
ls /
```

```
/bin
/sbin
/usr/bin
/usr/sbin
/opt
/etc
/var
/boot
/lib
/home
/root
/dev
/proc
```

Le système de fichiers UNIX détaillé

```
/bin     - (binaries) commandes binaires essentielles pour tous les utilisateurs
/sbin    - (system binaries) commandes binaires système et d'administration
/usr/bin - (user binaries) commandes binaires supplémentaires pour les utilisateurs
/usr/sbin- (user system binaries) commandes binaires système supplémentaires pr admins
/opt     - logiciels (opt)ionnels et paquets tiers
/etc     - (et cetera, puis editable text config) fichiers de configuration du système
/var     - fichiers variables/temporaires (logs, spool, cache, bases de données)
/boot    - fichiers nécessaires au démarrage du système (noyau, initramfs, grub) DANGER
/lib     - bibliothèques partagées (shared lib .so) et modules noyau essentiels
/home    - répertoires personnels des utilisateurs
/root    - répertoire personnel de l'utilisateur root
/dev     - (devices) fichiers spéciaux représentant les périphériques
/proc    - (processus, processor) système de fichiers virtuel exposant l'état du noyau et des processus
```

Système de fichiers stable, peut varier légèrement (détails) d'une distribution à l'autre.

Alias

Les alias sont extrêmement utiles ! Macro pour composer les programmes/commandes de base en **nouvelles commandes**, adaptées à votre usage (incluant les options utiles).

Un alias = une macro textuelle, appliquée *avant* le parsing de la ligne (**expansion**) par le *shell*.

- **Lister** tous les alias du shell courant :

```
alias
```

- **Créer** un alias :

```
alias nom_alias="liste de commandes"
```

Alias permanents

Par défaut, les alias définit avec `alias` ne vivent que dans le shell interactif courant, et disparaissent à la fermeture du shell.

Pour les enregistrer, **utiliser le fichier** `~/.bash_aliases` (chargé par défaut par `~/.bashrc`, votre fichier de config bash personnel) :

```
#fichier .bash_aliases
alias realusers="grep '/home' /etc/passwd | cut -d: -f1"
```

Recharger votre configuration (et vos alias) :

```
source ~/.bashrc
```

Limites des alias

- **Expansion simple** : un alias est une simple substitution de texte, faite avant l'exécution ;
- **Pas de paramètres** : un alias ne peut pas recevoir d'arguments ;
- **Pas de structure de contrôle** : pas de if, for, tests, conditions, etc. ;
- **Ne s'appliquent pas aux scripts** (par défaut) : ils ne sont développés (*expansés*) que dans les shells interactifs.

> **Les alias ne sont pas activés dans les scripts**, Bash désactive leur expansion en mode non-interactif

Dès qu'il faut utiliser des paramètres, des conditions, ou transmettre l'environnement, on bascule vers **une fonction shell**, beaucoup plus puissante et fiable. Abordée dans la suite du cours.

Exercices : se créer ses commandes (man, ls, cut, grep, tr)

0. **Inspecter** la sortie de `man -f ls cut grep tr` pour savoir ce que fait chaque commande;
1. **Créer** un alias `lls` qui affiche la liste longue triée par taille décroissante (en utilisant `ls -l`);
2. **Créer** un alias `dirs` qui n'affiche que les répertoires du répertoire courant (en utilisant `ls`, `grep` ou `cut`);
3. **Créer** un alias `big` qui affiche uniquement les 5 plus gros fichiers du répertoire courant;
4. **Créer** un alias `owners` qui affiche uniquement propriétaire et nom de fichier (en utilisant `ls`, `cut` et `tr`);
5. **Enregistrer** ces alias dans votre fichier `~/.bash_aliases`.

> Conseil : dès que vous rencontrez un nouveau programme --> `man <nouveau programme>`
> Dans la suite de la formation, nous allons apprendre à créer **nos propres fonctions (programmes ?)** en sh/bash pour créer des commandes encore plus complexes

Correction

```
#affiche tous les fichiers fmt long
alias ll="ls -al"
#affiche la liste des utilisateurs humains
alias realusers="grep '/home' /etc/passwd | cut -d: -f1"

#1. affiche les fichiers du plus gros au plus petit
alias lls="ls -Sahl"

#2. affiche que les répertoires
alias dirs="ll -A | grep ^d | cut -d\" \" -f9"

#3. affiche les 5 plus gros fichiers
alias big="ls -aSl | grep \"^-\" | head -n5"

#4. affiche les noms de propriétaire et les noms de fichiers
#On est bloqués avec cut ici, qui ne peut pas matcher plusieurs caractères espaces
#ce qui provoque un bug si les tailles de fichier ne sont pas sur le même nombre de caractères
#On utilise tr pour convertir/éliminer des caractères
alias owners="ls -l | tr -s ' ' | cut -d ' ' -f3,9"
#avec awk
alias owners="ls -al | awk '{print $3,$9}'"
```

> Il peut y avoir plusieurs façons de faire !

Quand peut on utiliser un alias ou non

Les *alias* (bash) sont conçus pour **aliaser des commandes** et ne sont **développés**(*expansés*) **qu'en position de commande**.

Position de commande signifie :

- **au début d'une ligne** ;
- après `;`, `|`, &&. L'opérande de gauche est considérée comme une nouvelle commande et donc un début de ligne ;
- ou après un alias dont la **valeur se termine par un espace**.

> un alias qui se termine par un espace indique au shell de continuer l'expansion des alias sur le mot suivant.

Exemples

```
alias ll='ls -l'

ll                # début de ligne OK
ll | ll           # après |         OK
echo x | ll       # après |         OK
cmd1 ; ll         # après ;         OK
true && ll        # après &&        OK

echo ll           #                 NON !
sudo ll           #                 NON !

alias sudo="sudo "
sudo ll           # après alias avec espace OK
```

Utiliser le manuel (man)

```
man man
```

Conventions : **savoir** lire le synopsis (**signature de chaque commande**), important !

```
Les conventions suivantes s'appliquent à la section SYNOPSIS et peuvent être utilisées comme un guide pour les autres sections.

texte gras          à taper exactement comme indiqué ;
texte italique      à remplacer par l'argument approprié ;
[-abc]              tous les arguments entre [ ] sont facultatifs ;
-a|-b               les options séparées par | ne peuvent pas être utilisées simultanément ;
argument ...        argument peut être répété ;
[expression] ...    toute l'expression située à l'intérieur de [ ] peut être répétée.
```

> **Comprendre la syntaxe** et l'organisation de la documentation d'UNIX vous **facilitera aussi la vie dans toutes les autres technologies** car leurs auteur·ices se sont très fortement inspirés de ces conventions pour écrire leur propre doc !

Utiliser le manuel (man) : naviguer et parcourir

On utilise souvent la documentation pour **rechercher** ou **retrouver** une information précise. **La navigation la plus utile et importante** est la **recherche par pattern**. Dans `man` :

- `/pattern` : **Recherche par pattern** (comme dans vi) ;
- `n` : match *suivant* ;
- `N` : match *précédent*.

Extrait de l'aide `man man` puis `h` :

```
  /pattern          *  Search forward for (N-th) matching line.
  ?pattern          *  Search backward for (N-th) matching line.
  n                 *  Repeat previous search (for N-th occurrence).
  N                 *  Repeat previous search in reverse direction.
  ESC-n             *  Repeat previous search, spanning files.
  ESC-N             *  Repeat previous search, reverse dir. & spanning files.
  ESC-u                Undo (toggle) search highlighting.
  ESC-U                Clear search highlighting.
  &pattern          *  Display only matching lines.
```

Obtenir rapidement une description courte de commandes :

```
man -f ls grep tr cut paste
```

Bien comprendre la syntaxe et l'usage des options

- Les options (`[options]`) permettent de **moduler** le comportement des commandes ou programmes du shell.
- Elles existent souvent sous **deux** formes :
  - Forme **longue** :
    - Mot complet **précédé de deux tirets**, ex. `--all` ;
    - L'argument peut être après un espace ou après un `=` , ex. `--regexp "motif"` ou `--regexp="motif"`
  - Une forme **abrégée** :
    - Lettre unique précédée d'un tiret, ex. `-a`
    - L'argument peut être collé à l'option ou séparé par un espace, ex. `-d:` ou `-d :`
- Les options abrégées peuvent souvent être **concaténées en une seule chaîne**. Par exemple `ls -a -S -r` peut s'écrire `ls -aSr` . Cette notation **compacte** est très pratique pour les commandes fréquemment utilisées.

Exemple avec `grep`

```
# Forme longue
grep --regexp="motif" fichier
grep --regexp "motif" fichier
grep --regexp         "motif" fichier

# Forme abrégée
grep -e"motif" fichier
grep -e "motif" fichier
grep -e       "motif" fichier
grep -e"motif" -A2 fichier
```

De l'usage de man, comment s'en servir ?

Une fois que l'on sait rechercher et naviguer dans la documentation avec `man`, comment s'en servir ?

- **Premier réflexe** quand on rencontre une nouvelle commande ;
- **Tester votre compréhension**, dans un environnement sûr. Lorsque l'on teste, on a un retour immédiat. Lorsque l'on finit par comprendre par essai/erreur, cette information est fortement consolidée en vous. Et cela procure une grande **satisfaction** d'y parvenir avec la documentation !
- **Revenir**, si `man` n'est pas toujours le meilleur premier point d'*entrée*, c'est le meilleur point pour **rappeler** une information;
- Si vous ne trouvez pas l'information, **faire A/R avec recherche Google/LLM et revenir** dans man pour confirmer, consolider;
- En **apprendre plus autour du connu** (enrichir vos connaissances, découvrir d'autres options utiles) ;
- Retrouver une information **instantanément**, sans connexion internet, sans envoyer 1000 requêtes HTTP ;
- **Inspirez-vous (critique aussi)** en pour **écrire votre propre documentation, vos spécifications** (apprécier la concision, niveau de détail, insuffisant parfois ? Pourquoi ?)

man ?

Tous les programmes n'ont pas forcément de page `man`, `man` est (généralement de grande qualité. Certaines pages sont mieux fournies que d'autres (documentation collaborative). Cela dépend des auteur·ices des pages, de l'usage des programmes, de la traduction, etc. Vous pouvez écrire vos propres pages man !

> Recommandé : consulter les pages `man` en anglais.

Utiliser le *shell* : savoir gérer les flux et manipuler du texte

Sous UNIX, **tout est fichier** et toutes les **interfaces** entre programmes sont des **chaînes de caractères**. Il est donc essentiel pour vous d'être à l'aise avec **la manipulation du texte**, c'est à dire :

- **Maîtriser les quotations** (`'...'` vs `"..."`) pour contrôler l'interprétation des variables et des caractères spéciaux ;
- **Connaître et utiliser les caractères spéciaux** (`$`, `*`, `?`, `|`, `>`, etc.). Ces caractères ont un **rôle particulier** dans le *shell* et doivent être maîtrisés pour éviter des bugs ;
- **Savoir parser, filtrer, déplacer et transformer du texte** avec des commandes comme `grep`, `sed`, `awk`, `cut`, `tr`, etc. ;
- **Maîtriser (les bases) des expressions régulières** (*regex*). Indispensable car la plupart des programmes permettent l'usage de *motifs textuels*.

> Le shell est en environnement où toutes les données sont traitées comme du texte. Il n'y a qu'un "type" : text !

Simple quotes et double quotes

En Bash (et en général dans tous les *shells* Unix et langages de programmation), les **simples quotes** `'...'` et les **doubles quotes** `"..."` n'ont pas le même comportement :

- **Simple quotes** '...' :
  - Tout ce qui est entre simples quotes est pris littéralement.
  - **Aucun caractère spécial n'est interprété** (pas de `$`, pas de `\`, pas de commandes).

```
# Affiche littéralement la variables d'environnement $HOME
echo '$HOME'
```

- **Double quotes** "..." :
  - Les double quotes permettent **l'expansion de variables**, des **commandes**, et **certaines séquences d'échappement**.
  - Certains **caractères spéciaux n'y sont pas interprétés** : `*` `?` `[ ]` (pour le *globbing*), mais pas l'expansion de variable comme `$var` ou `$(cmd)`

```
# Affiche la valeur de $HOME, par ex /home/jdoe
echo "$HOME"
```

Quoting : exemples

```
# Affiche $PATH
echo '$PATH'
# Affiche \\ (pas d'échappement)
echo '\\'
# Affiche \ (échappement)
echo "\\"
# Affiche contenu du path, où les bins sont recherchés (expansion de variables d'environnement)
echo "$PATH"
# Toute variable référencée avec $ est expansée avant l’exécution de la commande
echo $PATH
# Affiche : fichier_2025.txt (expansion de commande)
echo "fichier_$(date +%Y).txt"
# Affiche tous les fichiers markdown du répertoire courant
ls *.md
# Affiche les fichiers files0.txt, file1.txt, etc.
ls file[0-9].txt
# Affiche les fichiers dataXxxx.log dont le 5e caractère X n'est pas 'a'
ls data[^a]*.log
# Affiche le fichier "*.md" s'il existe
ls "*.md" //Erreur, aucun fichier "*.md" !
```

Regex à connaître

Concept	Exemple	Signification	Match (capture)
`.`	`a.b`	n'importe quel caractère	acb, auc, a+c, a?c
`?`	`ab?c`	**0 ou 1** occurrence du caractère ou du groupe précédent”	ac, abc
`+`	`ab+c`	**1 ou plusieurs** occurrence du caractère ou du groupe précédent	abc, abbc, abbbbc
`*`	`ab*c`	**0 ou plusieurs** occurrence du caractère ou du groupe précédent	**ac**, abc, abbc, abbbbc
`^`	`^abc`	**début** de ligne	correspond aux lignes commençant par "xyz"
`$`	`xyz$`	**fin** de ligne	correspond aux lignes finissant par "xyz"
`[ ]`	`[a-z]`	**intervalle** de caractères	un caractère lowercase (a à z)
`[^ ]`	`[^0-9]`	**caractère non compris** dans l'intervalle	toutes les lettres sauf chiffres
`|`	`ab|cd`	Booléen (OR) : **match expression avant ou après**	ab, cd
`{}`	`a{1,3}`	1, 2 ou 3 occurrences du caractère ou du groupe précédent	ab, aab, aaab

Un **groupe de capture** est constitué avec les **parenthèses**. Par ex. `(ab)+`, `(ab)` est le groupe, `+` s'applique au groupe entier : `abab`, `ababab`

Le type d'expressions que vous devez savoir écrire et comprendre

```
# Afficher les lignes commençant par abc
grep "^abc" fichier.txt

# motif suivi d’un chiffre
grep "motif[0-9]" fichier.txt

# Remplacer tous les nombres par NUM
sed 's/[0-9]\+/NUM/g' fichier.txt

# Imprimer les lignes commençant par ERROR
awk '/^ERROR/ {print $0}' log.txt

# Filtrer les messages du kernel pour ne suivre que les événements des périphériques USB ou Ethernet.
dmesg | grep -E "(usb|eth)"

# Affiche la liste des images png dont le numéro est pair
for i in {0..20}; do printf "%02d.png\n" "$i"; done | grep -E '[02468]\.png$'
```

TP : Se créer un dépôt `dotfiles`

1. **Installer** git sur la VM ;
2. **Créer** un dépôt contenant vos *dotfiles* (`.bash_aliases` , `.bashrc` , `.vimrc,` `.nanorc`);
3. **Générer** des clés SSH avec `ssh-keygen` ;
4. **Publier** la clé publique sur votre serveur git (Github, Gitlab, etc.) pour vous authentifier en SSH ;
5. **Publier** sur le dépôt *remote*.

> Maintenez à jour vos dotfiles et les centraliser sur un dépôt distant pour pouvoir y accéder depuis n'importe quelle machine au besoin.
> **Ne pas publier d'informations sensibles !**

Fichiers spéciaux STDIN, STDOUT, STDERR

En Unix, **tout est fichier**, y compris les **flux standards** utilisés par les programmes. Trois fichiers spéciaux permettent aux programmes de communiquer avec le monde extérieur :

Nom	Numéro descripteur	Usage
STDIN	0	Entrée standard : là où le programme lit ses données
STDOUT	1	Sortie standard : là où le programme écrit ses résultats
STDERR	2	Sortie d'erreur : là où le programme écrit ses messages d'erreur

Ces flux peuvent être redirigés vers des fichiers ou vers d'autres programmes avec les **opérateurs de redirection de flux**.

> Flux (*stream*) : séquence de données (*bytes*) qui peut être lue ou écrite progressivement sans avoir besoin de tout charger en mémoire.

Opérateurs de redirections de flux

Opérateur	Description	Exemple
`>`	Redirige la **sortie standard** vers un fichier (écrase le fichier existant)	`ls > fichier`
`<`	Redirige la **lecture depuis un fichier** vers l'entrée standard	`mysql -uroot -p < script.sql`
`|`	Envoie la **sortie standard** d'un programme vers l'**entrée standard** d'un autre	`cat fichier | grep "motif"`
`>>`	Redirige la **sortie standard** vers un fichier (*append mode*, ajoute à la fin)	`echo "some log" >> app.log`
`<<`	**Here document (*Heredoc*)** : fournit un bloc de texte sur plusieurs lignes	`cat << EOF ... EOF`
`<<<`	**Here string** : fournit une chaîne en entrée standard	`grep "motif" <<< "ligne de texte"`
`2>`	Redirige la **sortie d'erreur** (descripteur 2) vers un fichier (écrase)	`ls /inexistant 2> fichier`
`2>>`	Redirige la **sortie d'erreur** vers un fichier (ajoute à la fin)	`ls /inexistant 2>> fichier`
`&>`	Redirige **STDOUT et STDERR** vers le même fichier (écrase)	`commande &> fichier`
`&>>`	Redirige **STDOUT et STDERR** vers le même fichier (ajoute à la fin)	`commande &>> fichier`

Exemples

```
#Syntaxe Heredoc (qu'on retrouve dans certains langages comme en PHP !)
grep "motif" <<EOF
ligne 1
ligne contenant motif
ligne 3
EOF
#/dev/null est le fichier où rediriger ce qu'on veut jeter
ls /foo 2>/dev/null
```

> `/dev/null` est un fichier spécial dans Unix/Linux souvent appelé *trou noir*. Tout ce qui est écrit dans `/dev/null` est **perdu à tout jamais** !

Command substitution

Insère la sortie d’une commande dans une autre commande.

```
$(commande)
```

Exemple :

```
echo "Nous sommes le $(date +%F)"
echo "Utilisateurs connectés : $(who | wc -l)"
#supprimer tous les conteneurs avec docker
docker rm -f $(docker ps -aq)
```

Process substitution

La substitution de processus permet de faire référence à l'entrée ou à la sortie d'un processus en utilisant un nom de fichier. La substitution de processus envoie la sortie d'un processus (ou de plusieurs processus) vers l'entrée standard d'un autre processus.

```
#entrée
<(command_list)
#sortie
>(command_list)
```

`command_list` est exécutée de manière asynchrone, et son entrée ou sa sortie apparaît *comme* un nom de fichier. Ce nom de fichier est passé en argument à la commande courante comme résultat de l'expansion.

> Notez qu'aucun espace ne doit apparaître entre le symbole < ou > et la parenthèse ouvrante, sinon la construction serait interprétée comme une redirection.

Exemple : `diff <(sort f1.txt) <(sort f2.txt)`

Process substitution exemples

Préparer un fichier de log à traiter :

```
cat <<EOF > app.log
ERROR
WARNING
ERROR
ERROR
ERROR
WARNING
WARNING
ERROR
WARNING
ERROR
EOF
```

On souhaite repartir les entrées du fichier de log dans **deux fichiers distincts** pour analyse. Pour cela, on peut utiliser le mécanisme de *process substitution* avec le programme `tee` :

```
# La commande lit app.log (redirige stdin), duplique le flux, et envoie chaque copie vers un processus différent
# l'un extrait les lignes contenant ERROR, l'autre celles contenant WARN, chacun étant redirigé vers son fichier.
tee >(grep ERROR > erreurs.log) >(grep WARN > warnings.log) < app.log
# Idem mais en redirigeant la sortie de tee pour la rendre silencieuse
tee >(grep ERROR > err.log) >(grep WARN > warn.log) < app.log >/dev/null
```

Philosophie d'UNIX

> **Important à comprendre** pour utiliser au mieux les possibilités offertes par un UNIX. C'est aussi pourquoi il est important de vous intéresser à l'histoire des technologies pour comprendre leur design !

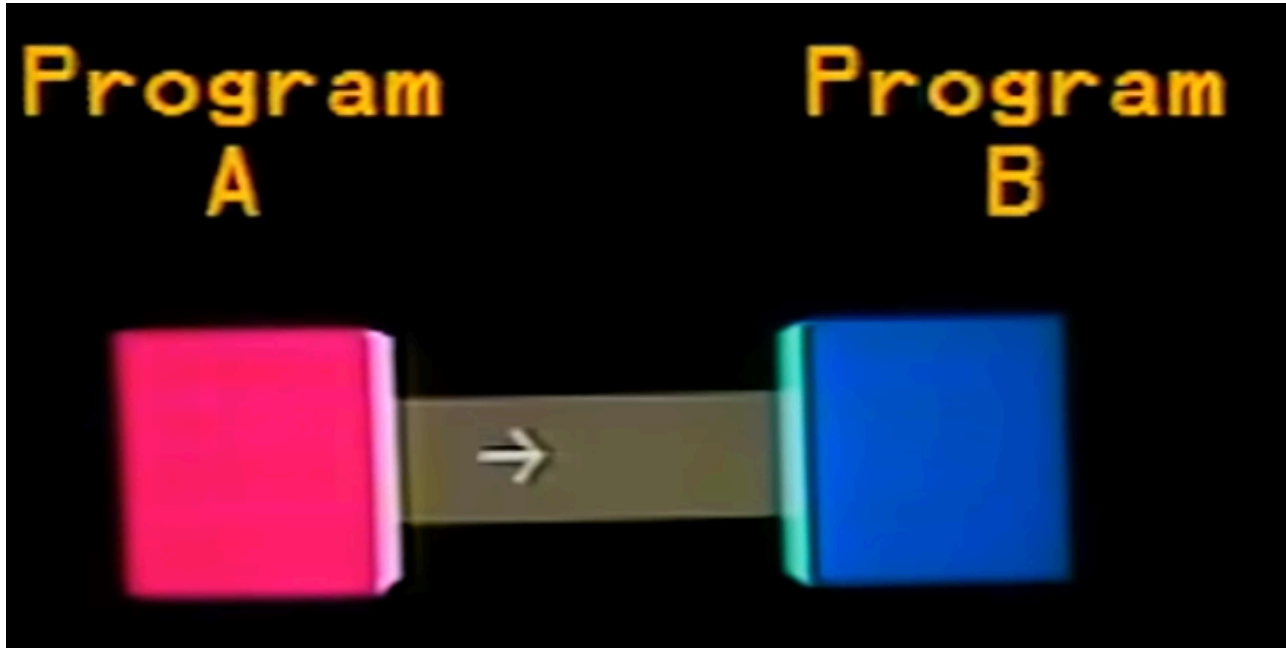
- Unix : **faire chaque chose d'une seule façon** (*simplicité* vs *complexité*)
- Dans la tradition d'Unix, **tout programme devrait présenter une interface *texte***, l'interface *universelle*. Cette interface permet à un programme de recevoir l'entrée d'un autre programme sous forme de texte, facilitant leur composition via des *pipes*. Ainsi, le programme peut être *composé* avec d'autres programmes pour en fabriquer de nouveaux.
- Chaque programme fait **une chose et le fait bien** ;
- Personne ne sait *à l'avance* ce dont l'utilisateur a besoin. Au lieu de fournir des applications complexes et rigides, Unix fournit **des briques élémentaires composables** pour **créer n'importe quel programme complexe** capable de **résoudre un problème donné**.

Redirections de flux et composabilité

{.marp-bg-img}

```
#Compter les fichiers sources JS d'un projet
ls | grep "*.js" | wc -l
```

Source de l'image : UNIX: Making Computers Easier To Use -- AT&T Archives film from 1982, Bell Laboratories



> Une interface texte universelle, cela vous rappelle quelque-chose ? Oui, **le protocole HTTP** (flux texte entre machines) ! Les web API RESTful utilisent le même principe ! La philosophie Unix de simplicité et de composabilité a inspiré la conception des protocoles modernes.

Philosophie d'UNIX : composition et composants

Pour *composer* deux programmes, il faut utiliser un pipe (ou tube), représenté par le caractère `|`. Par exemple :

```
command1 | command2 | command3
```

`command1` lit son entrée depuis la sortie de `command2`, et `command3` lit son entrée depuis la sortie de `programme1`. **On peut dire que `programme1 | programme2 | programme3` est un *nouveau* programme**, *composé* à partir de 3 autres programmes.

On pourrait écrire `alias command4 = 'command1 | command2 | command3'`.

> Commande ? Programme ? Peu importe ! C'est un **détail d'implémentation** grâce à l'interface texte ! Chaque brique ici peut être n'importe quel programme : une commande shell, un programme écrit en Bash, Python, PHP, Node.js, Perl, C#, etc. **Tant qu'il implémente l'interface texte** !

Code de retour d'une commande

- **Chaque commande renvoie un code de sortie** (*exit status*) accessible via `$?`
  - `0` : succès
  - `>0` : erreur ou code spécifique à la commande
- Code de sortie permet de :
  - **Vérifier** le succès d'une commande ;
  - Enchaîner des commandes **conditionnellement** ;
  - Debug/logging ;
  - Tests.

```
ls
echo $?
ls fichier_inexistant
echo $?
```

Opérateurs du shell (à connaître)

Opérateur	Description	Exemple
`&&`	Exécute la commande suivante **si la précédente réussit** (exit 0)	`mkdir test && cd test`
`||`	Exécute la commande suivante **si la précédente échoue** (exit != 0)	`grep motif fichier || echo "Non trouvé"`
`;`	Exécute les commandes **successivement, sans condition**	`echo "1"; echo "2"`
`&`	Exécute la commande **en arrière-plan**	`sleep 10 &`
`()`	Exécute les commandes **dans un sous-shell**	`(cd /tmp && ls)`
`{}`	Groupe de commandes **dans le shell courant**	`{ echo 1; echo 2; } > out.txt`

Sécuriser la connexion SSH

TP

En conclusion

Être à l'aise avec le shell, penser et agir selon les concepts d'UNIX.

- **Création** et **configuration d'une machine virtuelle** Debian comme **environnement de test**
- Gestion des paquets avec `apt` et gestion des dépôts
- Installation d'un serveur SSH sécurisé
- Les flux `STDIN`, `STDOUT`, `STDERR`
- **Redirections de flux**, *command* et *process substitution*
- Utilisation du **manuel** (`man`), création d'**alias** et compréhension des **quotations**, expressions régulières
- **Philosophie UNIX** :
  - tout est *fichier* ;
  - toute donnée est *texte* (**interface universelle**) ;
  - chaque programme réalise une tâche de manière complète (comme une fonction !);
  - programmes **composables** *via* des *pipes* pour en créer de nouveaux.

TP : Configuration des VM

1. **Transformer** la VM `server` en serveur FTP. Pour trouver un programme serveur ftp, rechercher dans les paquets disponibles:

```
apt search 'serveur ftp léger'
```

2. **Tester** en copiant des fichiers sur la VM via le protocole FTP avec un client FTP de votre choix.

3. **Créer** une autre machine virtuelle `db` qui fera office de serveur de bdd. Pour cela, cloner l'image `server`.

4. Y installer et configurer une base de données MySQL :

```
apt search 'mysql server' | grep -iA5 '^mysql'
```

5. **Requêter** la `db` depuis `server` avec le programme client `mysql`.