esiea

Introduction à Linux

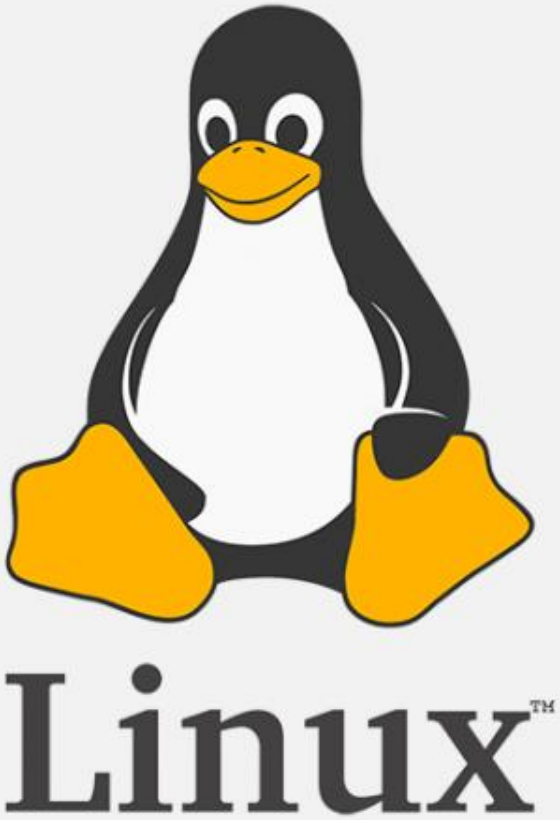
V2 - 2023

SEMESTRE 1

Sommaire

esiea

Qu’est-ce-que Linux ??



- Créé en **1991** par *Linus Torvalds*

Linux est un système d'exploitation réunissant :

- **le noyau Linux** (The Linux Kernel)
- **le système d'exploitation GNU**

- Un système **Libre**, basé sur **UNIX (free and open source)**

- Massivement utilisé en **embarqué**, sur les **super-calculateurs** ou les **serveurs,**

- Propose **de nombreuses déclinaisons** sous forme de **"distributions"**

Qu'est-ce que GNU ?

GNU est un système d'exploitation constitué de **logiciels libres (free software)**, c'est-à-dire dont l'utilisation, l'étude, la modification et la duplication par autrui en vue de sa diffusion sont permises, techniquement et juridiquement

Le système d'exploitation GNU comprend des logiciels GNU (programmes publiés par le projet GNU) ainsi que des logiciels libres publiés par des tiers.

Le développement de GNU a rendu possible l'utilisation d'un ordinateur sans logiciel susceptible de bafouer votre liberté.

***G**NU's **N**ot **U**NIX, littéralement : GNU n'est pas UNIX*

L'OS GNU deviendra le projet GNU

L'utilisation de Linux dans le monde

Serveur public sur Internet:

- 80% de Unix-link (Linux, etc)
- 20% Windows

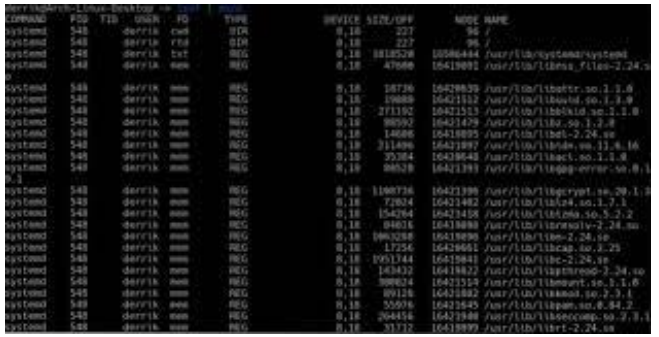
96.3% des meilleurs du monde 1 million de serveurs tournent sous Linux.

90% de toute l'infrastructure cloud fonctionne sur Linux

Pratiquement tous les meilleurs hébergeurs cloud l'utilisent.

Source: https://frameboxxindore.com/other/what-percentage-of-servers-are-linux.html - 2019

- **Interface graphique**
- \+ Simple
- \- Change selon les distributions, pas toujours disponible



Ligne de Commande

\+ Puissante, rapide, permet de comprendre, toujours disponible, change peu

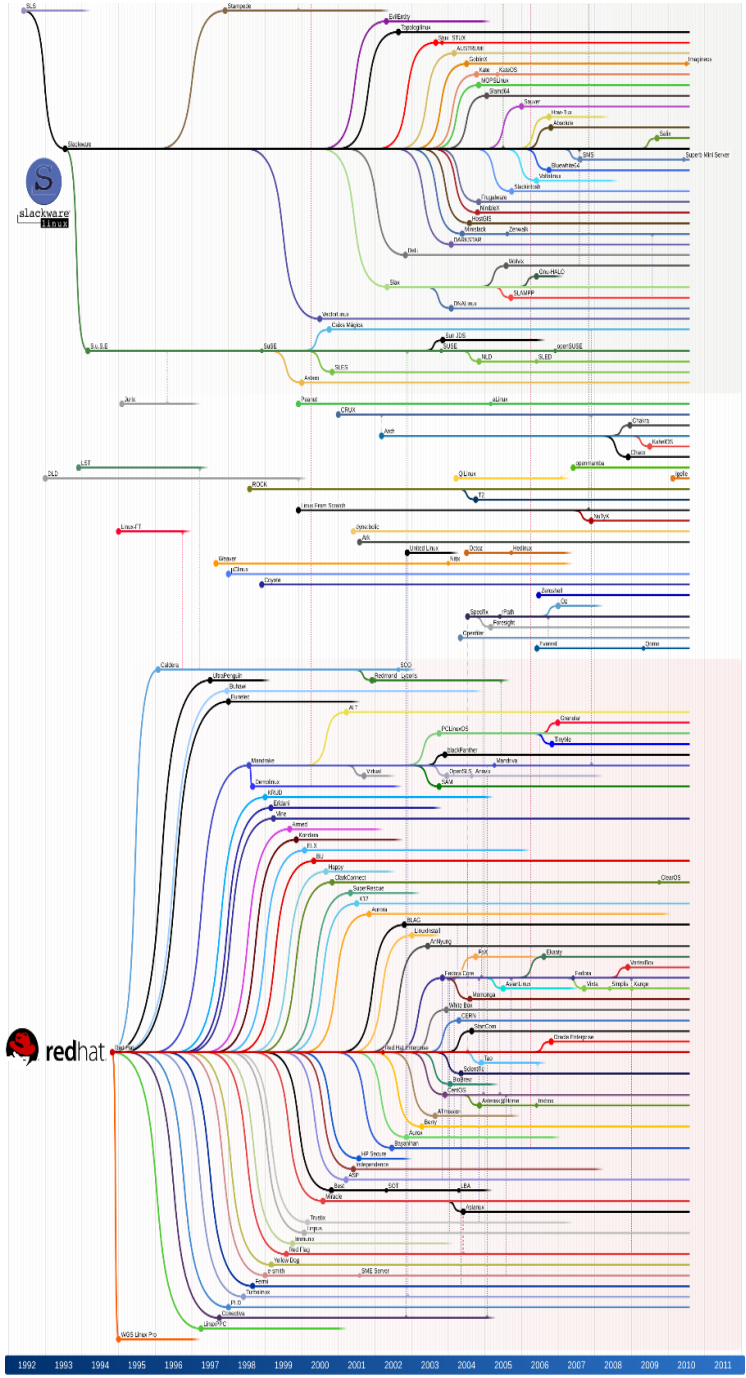
\- Plus complexe au début

Les éléments les différenciant principalement sont :

- ✓ la convivialité (facilité de mise en œuvre),
- ✓ l'intégration (taille du parc de logiciels validés distribués),
- ✓ la notoriété (communauté informative pour résoudre les problèmes),
- ✓ leur fréquence de mise à jour, leur gestion des paquets
- ✓ le mainteneur de la distribution (généralement une entreprise ou une communauté).

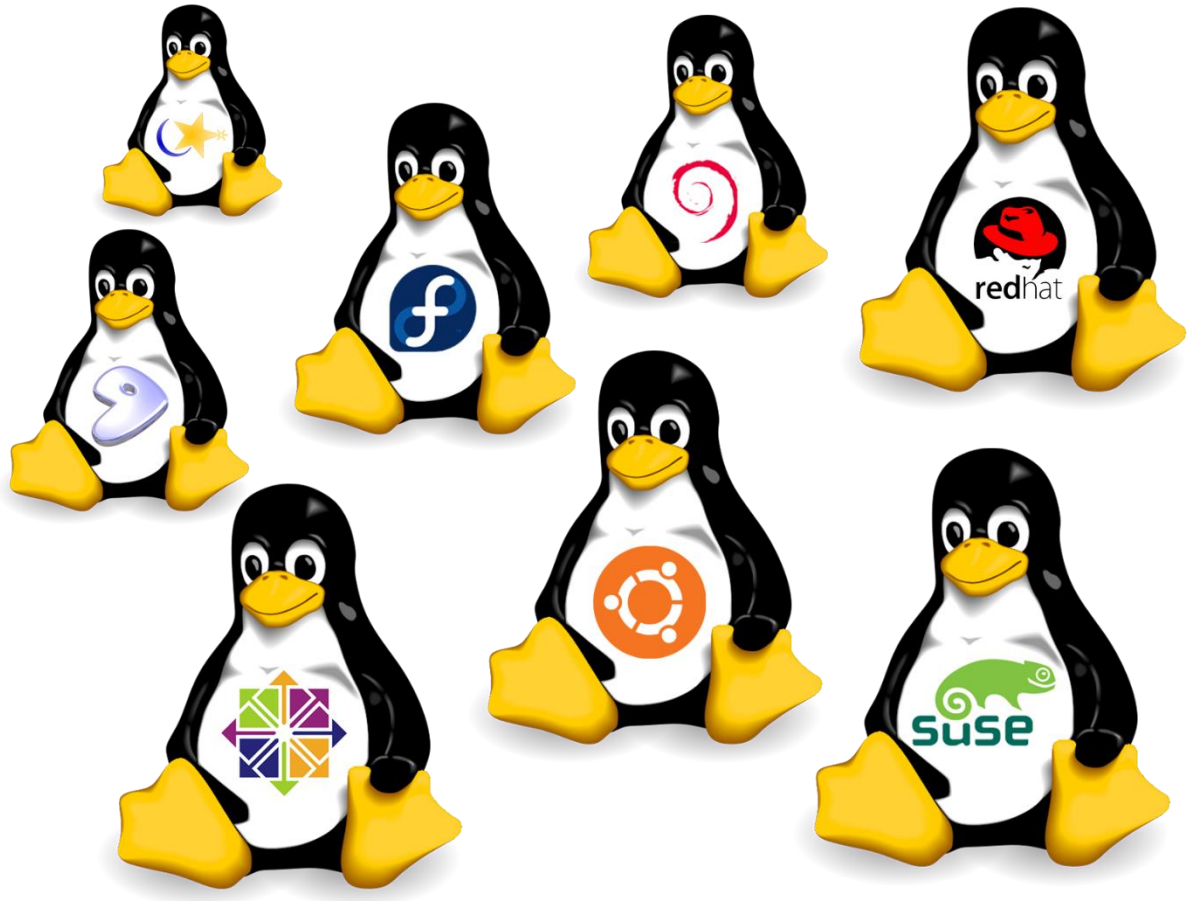
Leur point commun est le noyau Linux, et un certain nombre de commandes Unix.

Les parties GNU et Linux d'un système d'exploitation sont indépendantes, on trouve aussi bien des systèmes avec Linux et sans GNU — comme Android — ou des systèmes GNU sans Linux — comme GNU/Hurd.



Laquelle je choisis ???

Difficile … il vous faudra tester !
Il y en a pour tous les goûts, pour toutes les utilisations, professionnels, particuliers, gouvernement, sécurité, enfants …



Si nous devions en retenir 3 ..

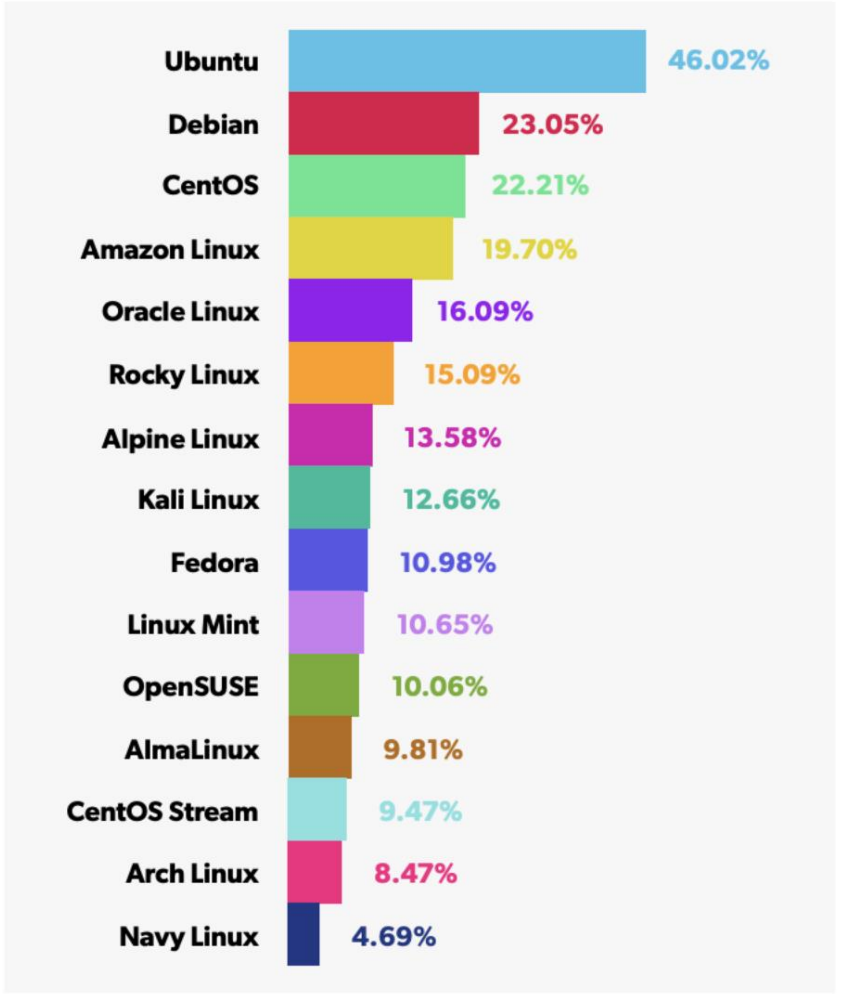
- Debian
- Ubuntu (base Debian)
- Mint (base Ubuntu… basée Debian)

Mais…

Le monde des distributions Linux est en perpétuel mouvement, ainsi voici les distributions populaires de 2024 :

- MX Linux
- Mint
- EndeavourOS
- Debian
- Manjaro
- Ubuntu
- Fedora
- Zorin
- Pop!_OS
- openSUSE

Which Open Source Linux Distributions Does Your Organization Use Today?



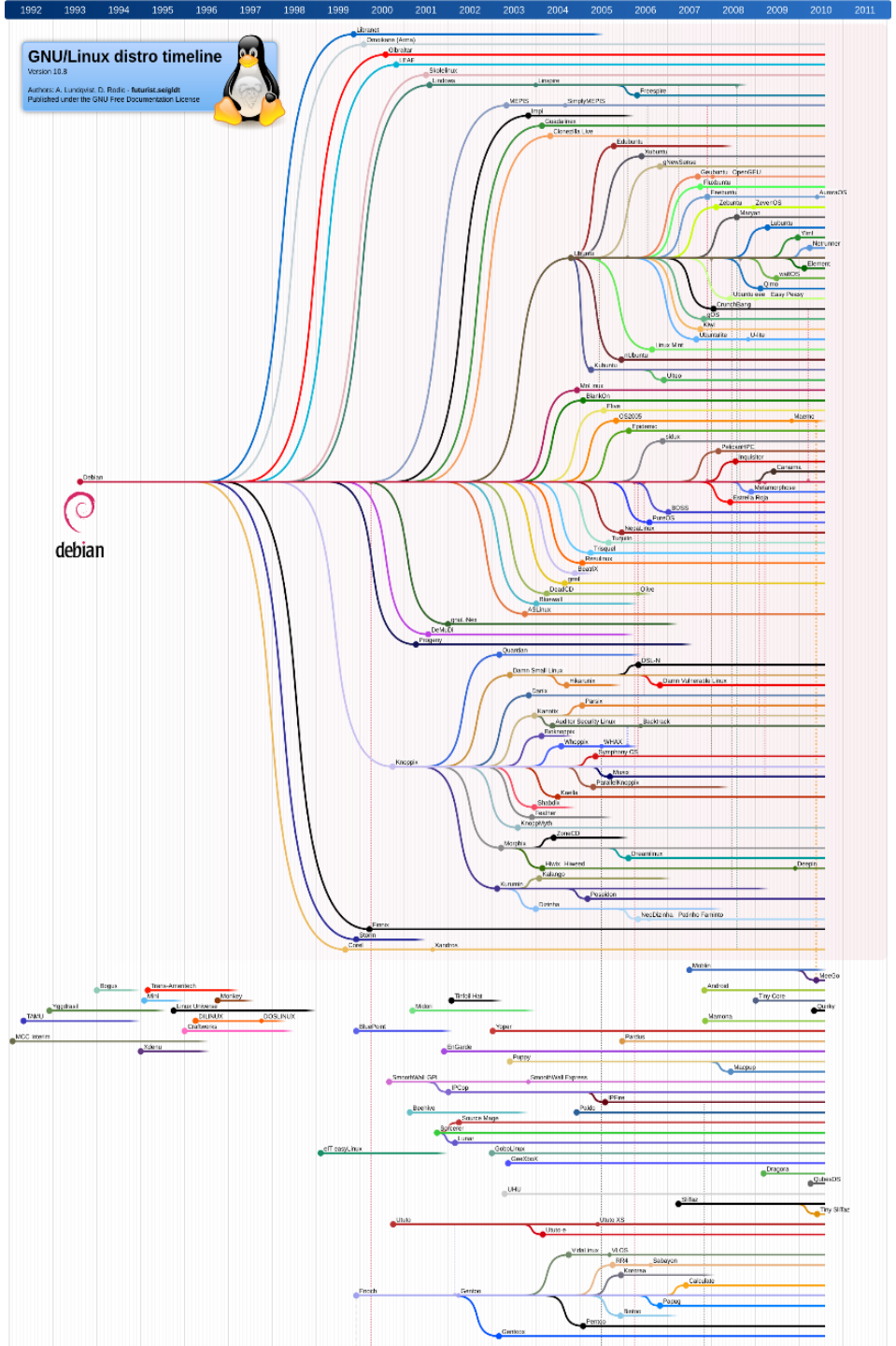
Les distributions de Linux
"Linux Distro"

Une distribution Linux, appelée aussi distribution GNU/Linux pour faire référence aux logiciels du projet GNU, **est un ensemble cohérent de logiciels**, la plupart étant **des logiciels libres**, assemblés autour du **noyau Linux**.

Le terme « distribution » est calqué sur l'anglais *software distribution* qui signifie « collection de logiciels » en français.

Il existe une **très grande variété de distributions Linux**, chacune ayant des **objectifs et une philosophie particulière**.

Certaines distribution Linux ne font pas partie du projet GNU



[Toutes les distributions Linux]

WSL
Windows Subsystem for Linux

https://learn.microsoft.com/en-us/windows/wsl/install

WSL est une sorte de machine virtuelle qui permet de faire tourner des distributions Linux dans Windows.

L'installation est simplifiée par rapport a une VM traditionnelle et elle consomme moins de ressources et démarre plus vite.

Le reste de cours sera fait dans WSL avec la distribution Debian officielle (installé via le Store Windows).

https://apps.microsoft.com/detail/9msvkqc78pk6

esiea

L’arborescence Linux

Dans les systèmes Linux, tout est fichier, donc, contrairement aux systèmes Windows, un fichier Linux peut être

Un fichier

Un programme en cours d'exectution

Un périphérique

Un répertoire

Une partition

…

Pour structurer ces fichiers, Linux ne dispose pas d'unités :

C:

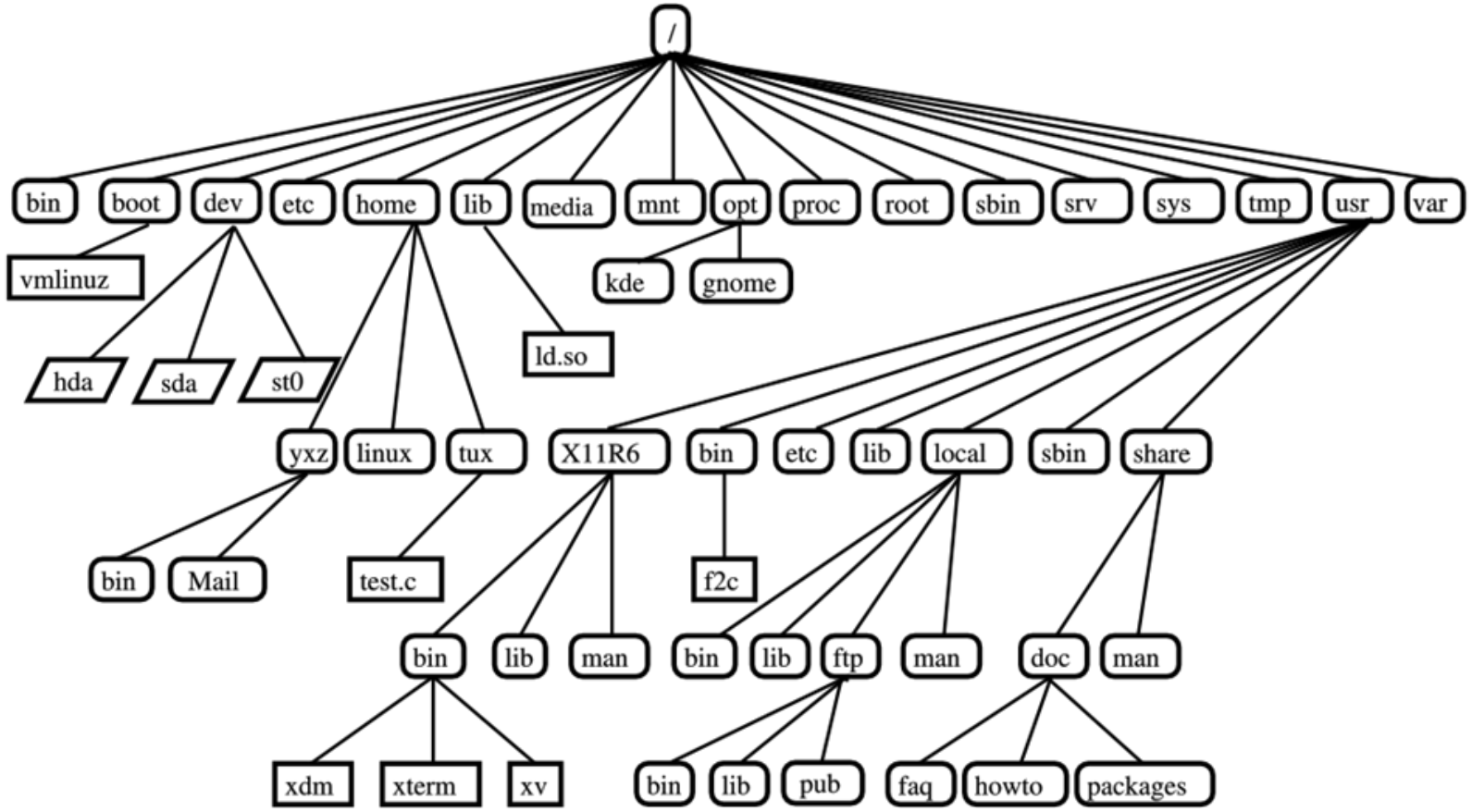
D:

E:

…

Linux utilise le standard FHS pour définir son arborescence.

Ce standard propose une structure de répertoire dont chacun possède un rôle spécifique dans FHS



Parmi ces répertoires, plusieurs sont importants, on peut en citer :

- /bin/ : Contient toutes les commandes de base nécessaires au démarrage et à l'utilisation d'un système minimaliste (par exemple : cat, ls, cp, sh).
- /sbin : Contient les commandes systèmes réservées aux administrateurs.
- /boot : Contient les fichiers nécessaires au démarrage du système d'exploitation.
- /dev : Contient des fichiers correspondants à un périphérique (disques , disquettes …).
- /etc : Contient la plupart des fichiers de configuration du système.
- /home/ : Utilisé pour stocker les répertoires utilisateurs (exemple : /home/user1).
- /opt : Utilisé comme emplacement d'installation d'un logiciel utilisé.
- /tmp/ : Utilisé pour stocker les fichiers temporaires tout comme /var/tmp et /run/tmp et généralement vidé à chaque démarrage.

esiea

La syntaxe Linux

La Ligne de Commande

Comme tout nouveau langage, il est compliqué à appréhender au début … le tout est de pratiquer !

Lorsque vous utilisez un système d'exploitation Linux, vous devez utiliser **un shell** – une interface qui vous donne accès aux services du système d'exploitation.

La plupart des distributions Linux utilisent une interface utilisateur graphique (GUI) comme shell, principalement pour faciliter l'utilisation par les utilisateurs.

Cela étant dit, **il est recommandé d'utiliser une interface en ligne de commande (CLI) car elle est plus puissante et plus efficace**.

Les tâches qui nécessitent un processus en plusieurs étapes via l'interface graphique peuvent être effectuées en quelques secondes en tapant des commandes dans l'interface en ligne de commande.

Chaque utilisateur connecté au système d'exploitation est capable de diriger la machine en exécutant une commande dans un terminal :

COMMANDE OPTIONS ARGUMENTS

La commande étant la commande elle-même

suivie d'une ou plusieurs options **facultatives** qui servent à modifier le comportement de la commande

tandis que les arguments ,**facultatifs** aussi, représentent les objets sur lesquels la commande agit.

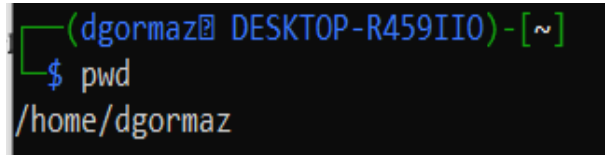
Exemple : ls –a /home

Traduction : Commande qui liste l'ensemble des fichiers, même cachés du répertoire /home

Voici une liste des commandes les plus populaires pour manipuler le système Linux :

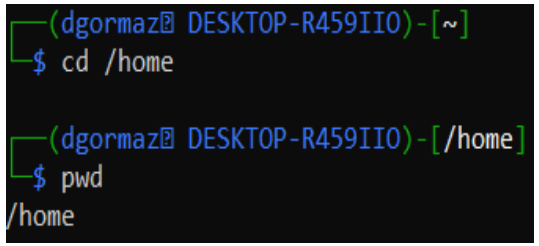
- La commande **pwd** :

Elle permet d'afficher l'emplacement où on se situe actuellement dans la hiérarchie FHS



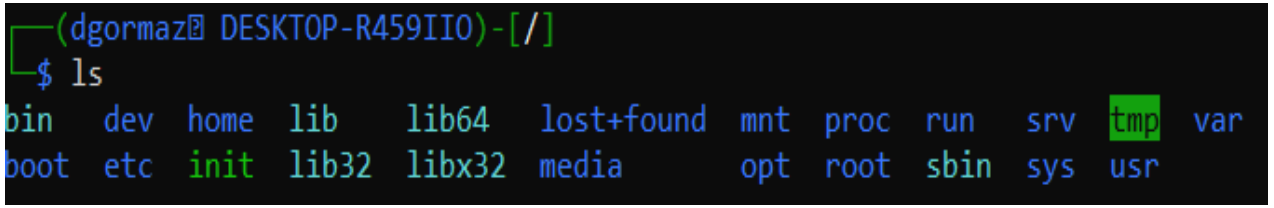
- La commande **cd** :

Elle permet de changer de répertoire courant et se situer sur un autre



- La commande **ls** :

Elle permet de lister les fichiers disponibles dans un répertoire. Si appelée sans arguments alors elle listera ceux du répertoire courant



Voici une liste des commandes les plus populaires pour manipuler le système Linux :

- La commande **mkdir / rmdir** :

Elle permet de créer/supprimer un répertoire

```
┌──(dgormaz DESKTOP-R459IIO)-[~]
└─$ mkdir repertoire

┌──(dgormaz DESKTOP-R459IIO)-[~]
└─$ ls
repertoire
```

- La commande **touch** :

Elle permet de créer un fichier vide ou modifier la date du dernier accès

```
┌──(dgormaz DESKTOP-R459IIO)-[~]
└─$ touch fichier

┌──(dgormaz DESKTOP-R459IIO)-[~]
└─$ ls -l
total 4
-rw-r--r-- 1 dgormaz dgormaz    0 Oct 11 11:20 fichier
```

- La commande **cp** :

Elle permet copier un fichier ou un répertoire

```
┌──(dgormaz DESKTOP-R459IIO)-[~]
└─$ cp fichier fichier2

┌──(dgormaz DESKTOP-R459IIO)-[~]
└─$ ls
fichier  fichier2  repertoire
```

```
┌──(dgormaz DESKTOP-R459IIO)-[~]
└─$ cp repertoire repertoire2 -r

┌──(dgormaz DESKTOP-R459IIO)-[~]
└─$ ls
fichier  fichier2  repertoire  repertoire2
```

- La commande **rm** :

Elle permet de supprimer un fichier ou un répertoire

```
┌──(dgormaz DESKTOP-R459IIO)-[~]
└─$ rm -f fichier fichier2

┌──(dgormaz DESKTOP-R459IIO)-[~]
└─$ ls
repertoire  repertoire2
```

Voici une liste des commandes les plus populaires pour manipuler le système Linux :

- La commande **mv**:

Elle sert à renommer ou déplacer un fichier ou un répertoire

- Les commandes d'édition **cat, more, head**, **tail, vi, emacs** :

Elle permet d'afficher le contenu d'un fichier

```
┌──(dgormaz DESKTOP-R459IIO)-[~]
└─$ cat /etc/hostname
DESKTOP-R459IIO
```

- La commande **echo** :

Elle permet d'afficher mais aussi écrire ou écraser (« > ») le contenu d'un fichier dans un autre fichier voire le suffixer (« >> »)

- La commande **curl** :

```
┌──(dgormaz DESKTOP-R459IIO)-[~]
└─$ echo hostname > /home/dgormaz/hostname
```

Elle permet d'envoyer ou télécharger un fichier accessible sur le réseau

Voici une liste des commandes les plus populaires pour manipuler le système Linux :

- La commande **alias** :

Elle permet de créer « un raccourci » d'une longue commande

```
┌──(dgormaz DESKTOP-R459IIO)-[~]
└─$ alias MX='dig mx groupe-aen.info'

┌──(dgormaz DESKTOP-R459IIO)-[~]
└─$ MX

; <<>> DiG 9.18.4-2-Debian <<>> mx groupe-aen.info
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 56517
;; flags: qr rd ad; QUERY: 1, ANSWER: 3, AUTHORITY: 0, ADDITIONAL: 0
;; WARNING: recursion requested but not available

;; QUESTION SECTION:
;groupe-aen.info.               IN      MX

;; ANSWER SECTION:
groupe-aen.info.        0       IN      MX      0 groupeaen-info01e.mail.protection.outlook.com.
groupeaen-info01e.mail.protection.outlook.com. 0 IN A 104.47.4.36
groupeaen-info01e.mail.protection.outlook.com. 0 IN A 104.47.11.202

;; Query time: 170 msec
;; SERVER: 172.23.32.1#53(172.23.32.1) (UDP)
;; WHEN: Tue Oct 11 11:49:55 CEST 2022
;; MSG SIZE  rcvd: 186
```

esiea

Voici une liste des commandes les plus populaires pour manipuler le système Linux :

- La commande de recherche **grep** :

Elle permet de rechercher les occurrences de mots à l'intérieur de fichiers. Des arguments permettent d'affiner la recherche comme –i (la casse), -c (compter),…etc.

- D'autres commandes :
  - cal
  - date
  - wc
  - spell
  - read
  - diff
  - …

esiea

L’autocomplétion !

Si vous ne connaissez pas le mot vous allez savoir l’apprécier !

Tapez le début d’une commande sur le prompt, puis sur la touche <TAB>, le système se chargera de trouver la fin de la commande.

GAIN DE TEMPS

MOINS D’ERREURS

esiea

- **Sensibilité à la casse !**

Dans du texte, la casse (minuscule ou majuscule) des lettres peut parfois changer la signification. Les mots en lettres capitales (haut de casse) n'ont pas toujours le même sens s'ils sont écrits en minuscules (bas-de-casse). Par exemple : **Rennes** est une ville mais **rennes** désigne des caribous

La **sensibilité à la casse** (traduction de l'anglais *case sensitivity*) est une notion informatique signifiant que dans un contexte donné, le sens d'une chaîne de caractères (par exemple un mot, un nom de variable, un nom de fichier, un code alphanumérique, etc.) dépend de la casse (capitale ou bas de casse) des lettres qu'elle contient.

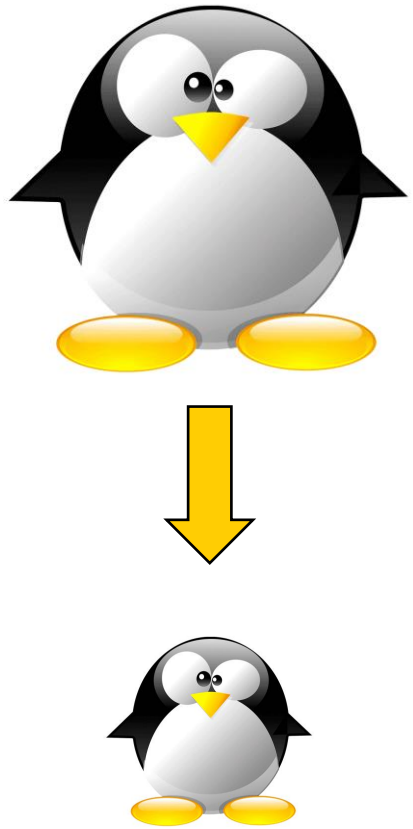
Ce terme tire son origine de la casse en typographie.
La sensibilité à la casse peut intervenir dans le tri par ordre alphabétique comme dans les comparaisons.

Dans ce sens ….

Lorsque vous créez votre utilisateur …

Laissez-le en minuscule…

pas de MAJUSCULE !!



Le Prompt

Le prompt est la première ligne affichée à l'ouverture du terminal / de l'invite de commande :

Pour exemple, je m'appelle Cedric (cedric) et le nom de mon ordinateur est ced-pc

Le prompt affichera : **cedric@ced-pc~$**

~ : Le tilde signifie que l'on se trouve sur mon répertoire personnel (home/cedric)

$: Le dollars indique que nous sommes connectés en simple utilisateur

Les Raccourcis

Correspondance :

~	Répertoire /home/username/
..	Répertoire parent
.	Répertoire courant

Ils permettent tous les trois de simplifier l'expression de **références absolues**…

Attention à ne pas confondre la racine de l'arborescence (racine = “root” en anglais) avec le superutilisateur “root”: ce sont deux choses différentes.

esiea

Qu’est-ce que le Scripting Shell ?

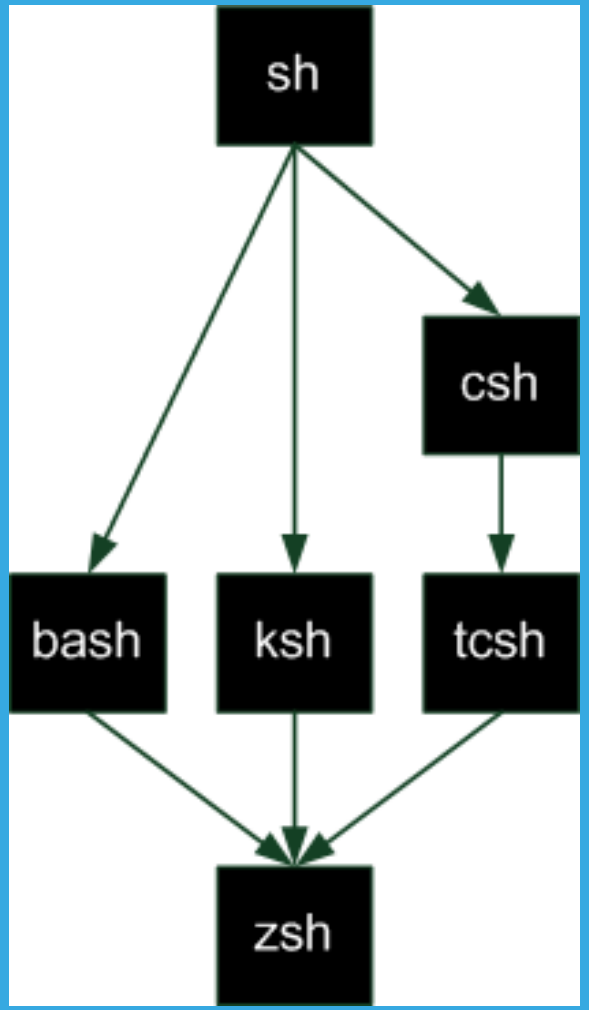
Qu'est qu'un *shell* ?

- Il s'agit basiquement d'un **interpréteur de commandes** (**CLI** – **C**ommand-**L**ine **I**nterpreter) …
- … qui permet d'**interagir** avec le **système d'exploitation**.
- On tape directement des **commandes** dans une **console** …
- … ou au travers d'un **script** sous forme de **fichier texte**.
- Exemples de shells : **bash**, **cmd**, **powershell**, …

esiea

Les *shells* sous Unix/Linux :

- Le ***Bourne Shell*** est l'ancêtre commun (date de 1977 !).
- Les principaux sont :
  - **bash** (***B****ourne-****A****gain* ***Sh****ell*) : amélioration du *Bourne Shell*.
  - **ksh** (***K****orn* ***Sh****ell*) : contient diverses fonctionnalités avancées.
  - **csh** (**C Sh**ell) : utilise une syntaxe proche du langage C.
  - **tcsh** (***Tenex C Shell***) : amélioration du *C Shell*.
  - **zsh** (***Z Shell***) : reprend des fonctionnalités de *bash*, *ksh* et *tcsh*



esiea

Utilisation du shell bash :

On entre la commande puis on tape sur entrée

```
$ echo "0xDEAD"
0xDEAD
$ echo "0xBEEF"
0xBEEF
```

Chaîner plusieurs commandes avec le caractère ;

```
$ echo "0xDEAD" ; echo "0xBEEF" ; echo "0xDEADBEEF"
0xDEAD
0xBEEF
0xDEADBEEF
```

esiea

Qu’est-ce qu’un script ?

- La première ligne est le ***shabang***.
- Commence par **#!**.
- Suivi du **chemin** vers l’**interpréteur** à utiliser.
- Les **lignes suivantes** seront interprétées, il s’agit du **contenu du script**.
- Cela fonctionne avec d’**autres interpréteurs** : PHP, NodeJS, Python, Perl, …

```
#!/bin/bash
echo "Hello World"
```

```
#!/usr/bin/php
<?php
echo "Hello World";
?>
```

```
#!/usr/bin/python3.5
print("Hello World")
```

esiea

Comment exécuter mon script ?

#1

```
$ cat script.sh
#!/bin/bash
echo "Hello World"
$ chmod +x script.sh
$ ./script.sh
Hello World
$
```

Le rendre executable

puis le lancer comme un programme classique

#2

```
$ cat script.sh
echo "Hello World"
$ /bin/bash script.sh
Hello World
$
```

Le donner en argument de mon interpreteur

N.B. : avec cette méthode, le shabang n'est pas obligatoire.

#3

```
# cat script.sh
#!/bin/bash
echo "Hello World"
# chmod +x script.sh
# cp script.sh /bin
# script.sh
Hello World
```

Le mettre dans un dossier de $PATH

pour l'utiliser comme une commande

esiea

Quizz time !

Comment chainer plusieurs commandes sur mon shell ?

- Avec le caractère « ; » Exemple : echo « a » ; echo « b »

Quelle est la première ligne de mon script et à quoi elle ressemble?

- Le Shabang
- De la forme #!/chemin/vers/interpréteur

Quels sont les trois méthodes pour exécuter un script ?

- Le rendre exécutable (chmod +x script.sh et ./script.sh)
- Le passer en argument de l'interpréteur (/bin/bash script.sh)
- Le mettre dans un chemin de $PATH (chmod +x script.sh et cp script.sh /bin), il devient ainsi une commande du shell

esiea

Les Variables

[1/3] Les bases sur les variables :

- Les variables sont toujours des **chaînes de caractères** …
- … mais suivant le **contexte** elles peuvent être traitées comme des **entiers**.
- Elles n'ont **pas besoin** d'être **déclarées**.
- On accède à la valeur en **préfixant** par le caractère **$**.
- La commande **echo** permet d'**afficher du contenu**.
- La commande **read** permet la **saisie interactive**.

esiea

[2/3] Les bases sur les variables :

Assigner une valeur à une variable et l'afficher :

```
$ MyVar='Elliot Alderson'
$ echo $MyVar
Elliot Alderson
```

Assigner une valeur via la saisie interactive et l'afficher :

```
$ read MyVar
Elliot Alderson
$ echo $MyVar
Elliot Alderson
```

esiea

[3/3] Les bases sur les variables :

Concaténer une variable dans une chaîne de caractère :

```
$ MyVar=‘fsociety’
$ echo "We are $MyVar"
We are fsociety
$ MyOtherVar="We are $MyVar"
$ echo $MyOtherVar
We are fsociety
```

Que se passe-t-il si nous utilisons le caractère ‘ ?

```
$ MyVar=‘fsociety’
$ echo ‘We are $MyVar’
We are $MyVar
```

esiea

[1/2] Les opérations arithmétiques :

Méthode #1 : via **let**

```
$ let a=666*42
$ echo $a
27972
$ let "b = $a / 42"
$ echo $b
666
```

esiea

[2/2] Les opérations arithmétiques :

Méthode #2 : via les **doubles parenthèses**

```
$ a=$((8484 / 2))
$ echo $a
4242
$ b=$(($a - 3576))
$ echo $b
666
```

esiea

Quizz time !

Quelles sont les commandes pour afficher et lire (interactif) des variables ?

- echo : afficher
- read : lire

Les variables en bash sont forcément des chaînes de caractères, est-ce vrai ?

- En théorie oui …
- … mais il est possible de faire des opérations arithmétiques !

Quels sont les deux moyens pour réaliser des opérations arithmétiques ?

- Via let
- Via les doubles parenthèses

esiea

Les Arguments

<Rappel sur les Arguments>

COMMANDE OPTIONS ARGUMENTS

La commande étant la commande elle-même

suivie d'une ou plusieurs options **facultatives** qui servent à modifier le comportement de la commande

tandis que les arguments ,**facultatifs** aussi, représentent les objets sur lesquels la commande agit.

Exemple : ls –a /home

Traduction : Commande qui liste l'ensemble des fichiers, même cachés du répertoire /home

esiea

#1/2 - Les arguments

Comment utiliser les arguments ?

- Nous pouvons **passer des arguments** à nos scripts.
- Ils sont **séparés par un espace** pour les délimiter.
- Pour y accéder, rien de plus simple :
  - $1 : argument numéro 1
  - $2 : argument numéro 2
  - …
  - $42 : argument numéro 42
  - …

```
$ cat demo.sh
#!/bin/bash
echo "bonjour $1"
echo "ton second argument est $2"
$ ./demo.sh Elliot EvilCorp
bonjour Elliot
ton second argument est EvilCorp
$ ./demo.sh UnSeulArgument
bonjour UnSeulArgument
ton second argument est
```

esiea

#2/2 - Les arguments

Les variables « spéciales » :

- **$0** : contient le **nom du script** tel qu'il est lancé.
- **$#** : contient le **nombre** d'arguments passés au script.
- **$*** : contient **l'ensemble** des arguments.
- Il existe aussi d'autres variables « spéciales » sans rapport avec les arguments.

```
$ cat demo.sh
#!/bin/bash
echo "nom du script : $0"
echo "nombre d'arguments : $#"
echo "liste des arguments : $*"
$ ./demo.sh toto titi tata
nom du script : ./demo.sh
nombre d'arguments : 3
liste des arguments : toto titi tata
```

esiea

Attention au caractère espace :

Si nous souhaitons passer un paramètre qui contient un espace il faudra l'entourer de "

```
$ cat demo.sh
#!/bin/bash
echo "bonjour $1"
$ ./demo.sh Elliot Alderson
bonjour Elliot
$ ./demo.sh "Elliot Alderson"
bonjour Elliot Alderson
```

esiea

Quizz time !

Comment j'accède au 4eme argument depuis mon script ?

- Via la variable $4

Comment vérifier si l'utilisateur de mon script a passé le bon nombre d'arguments ?

- Grâce à la variable $# qui contient le nombre d'arguments

J'appelle mon script de cette façon :

```
./script.sh "The" "devil is" at "his strongest"
```

Quel sera le contenu des variables ?

- $1 = The
- $2 = devil is
- $3 = at
- $4 = his strongest

esiea

Les conditions

esiea

#1/4 - Les conditions en Bash

- Utilisation classique de **if / else**.
- Il est possible d'imbriquer des tests avec **elif** (contraction de **else if**).

```
if [ condition ] ; then
        echo "Vrai"
else
        echo "Faux"
fi

if [ condition2 ] ; then
        echo "cond2"
elif [ condition3 ] ; then
        echo "cond3"
fi
```

esiea

#2/4 - Les conditions en Bash

- **Différents types de tests** peuvent être réalisés :
  - Valeurs numériques,
  - Chaînes de caractères,
  - Fichiers.
- Utilisation de « **&&** » et « **||** » pour **combiner** plusieurs tests.
- Le test peut être **inversé** avec le caractère « **!** ».

esiea

#3/4 - Les conditions en Bash

Exemple #1 : vérifier si un fichier existe

```
if [ -f "/root/fsociety.dat" ] ; then
        echo "File exists"
else
        echo "File doesn’t exist"
fi
```

Exemple #2 : comparer des chaînes de caractères

```
if [ "$a" = "$b" ] && [ -n "$a" ] ; then
        echo "A = B"
else
        echo "A != B or A is null"
fi
```

#4/4 - Les conditions en Bash

Exemple #3 : comparaisons numériques

```
if [ "$a" -gt 0 ] ; then
        echo "A > 0"
elif [ "$a" -lt 0 ] ; then
        echo "A < 0"
else
        echo "A = 0"
fi
```

esiea

Les boucles

esiea

#1/3 - Les boucles en Bash

- Utilisation classique de **for** et **while**.
- Pour la boucle while, les tests sont identiques à ceux pour les conditions.

```
for i in expr ; do
        # l'élément courant se trouve dans $i
done


while [ cond ] ; do
        # do something
done
```

esiea

#2/3 - Les boucles en Bash

- Exemple #1 : utilisation classique de la boucle for

```
for i in {1..10} ; do
        echo $i
done
```

```
$ ./script.sh
1
2
3
[…]
9
10
```

esiea

#3/3 - Les boucles en Bash

- Exemple #2 : utilisation classique de la boucle while

```
i=1
while [ "$i" -le 10 ] ; do
        echo $i
        i=$(($i + 1))
done
```

```
$ ./script.sh
1
2
3
[…]
9
10
```

esiea

Les conditions sur les chaînes de caractères :

Condition	Signification
["$a" = "$b"]	Vérifie si les deux chaines de caractères sont identiques
["$a" != "$b"]	Vérifie si les deux chaines sont différentes
[-z "$a"]	Vérifie si la chaine $a est vide
[-n "$a"]	Vérifie si la chaine $a est non vide

Les conditions sur les valeurs numériques :

Condition	Signification
["$a" -eq "$b"]	Vérifie si les deux variables contiennent la même valeur numérique (**eq***ual*)
["$a" -ne "$b"]	Vérifie si les nombres sont non égaux (**n***ot* **e***qual*)
["$a" -lt "$b"]	Vérifie si a est inférieur (<) à b (**l***ower* **t***han*)
["$a" -le "$b"]	Vérifie si a est inférieur ou égal (<=) à b (**l***ower or* **e***qual*)
["$a" -gt "$b"]	Vérifie si a est supérieur (>) à b (**g***reater* **t***han*)
["$a" -ge "$b"]	Vérifie si a est supérieur ou égal (>=) à b (**g***reater or* **e***qual*)

esiea

Les conditions sur les fichiers :

Condition	Signification
[-e "$nom"]	Vérifie si le fichier existe
[-d "$nom"]	Vérifie si le fichier est un répertoire
[-f "$nom"]	Vérifie si le fichier est un… fichier (un vrai, pas un clavier ou une souris)
[-L "$nom"]	Vérifie si le fichier est un lien symbolique
[-r "$nom"]	Vérifie si le fichier a les droits en lecture
[-w "$nom"]	Vérifie si le fichier a les droits en écriture
[-x "$nom"]	Vérifie si le fichier est exécutable
["$n" -nt "$m"]	Vérifie si n est plus récent que m (**n**ewer **t**han)
["$n" –ot "$m"]	Vérifie si n est plus vieux que m (**o**lder **t**han)

esiea

Quizz time !

Comment réaliser une condition en bash ?

```
if [ condition1 ] ; then
        echo "condition 1"
elif [ condition2 ] ; then
        echo "condition 2"
else
        echo "ni cond1 ni cond2"
fi
```

Comment réaliser une boucle for en bash ?

```
for i in expr ; do
        # l'élément courant se trouve dans $i
done
```

esiea

Quizz time !

Comment réaliser une boucle while en bash ?

```
while [ cond ] ; do
        # do something
done
```

Sur quels types d'éléments je peux faire des conditions?

- Sur des chaînes de caractères
- Sur des valeurs
- Sur des fichiers

esiea

Quelques astuces utiles

#1/4 - Tips

Récupérer le résultat d'une commande pour le mettre dans une variable :

```
MyVar=$(commande -argument)
```

Exemple #1 : récupérer le contenu d'un fichier

```
ContenuFichier=$(cat /tmp/mon_fichier)
```

Exemple #2 : récupérer la liste des fichiers d'un répertoire

```
ListeRepertoire=$(ls /etc/mon_repertoire)
```

#2/4 - Tips

Boucler sur les résultats d'une commande :

```
for resultat in $(commande –argument) ; do
        # l'élément courant est dans $resultat
done
```

Exemple : boucler sur les fichiers d'un répertoire :

```
for fichier in $(ls /etc/mon_repertoire) ; do
    echo $fichier
done
```

#3/4 - Tips

Lire un fichier ligne par ligne avec une boucle while :

```
while read LIGNE ; do
        echo "Ligne = $LIGNE"
done < /chemin/vers/le/fichier
```

```
$ ./script.sh
Ligne = The devil is at his strongest while we're looking the other way.
Ligne = Like a program running in the background silently.
Ligne = While we're busy doing other shit.
Ligne = Daemons, they call them.
```

#4/4 - Tips

Afficher les commandes qui sont exécutées :

```
$ cat script.sh
set -x
echo "Planting rootkit"
mv /tmp/fsociety.dat /root
systemctl start fsociety
exit 0
```

```
$ ./script.sh
+ echo ‘Planting rootkit’
Planting rootkit
+ mv /tmp/fsociety.dat /root
+ systemctl start fsociety
+ exit 0
```

esiea

Les commentaires

Un script sans commentaire = bullshit

- Pourquoi faire des commentaires ?

1. Pour votre collègue qui va prendre la suite, cela sera plus facile et plus rapide de le modifier.
2. Dans 3 ans quand vous allez changer de serveur, vous aurez oublié qu'elles étaient les variables que vous aviez créées spécifiquement pour ce cas.
3. Afin de mieux partager ces connaissances avec ses collègues qui en ont moins (il pourra mieux comprendre l'objet du script et progresser).
4. Pour savoir où l'on en est dans un script qui n'est pas terminé
5. Une note ou un rappel, a spécificité ou une clarification.
6. Et plein d'autres raisons….

NB: Un comment n'est pas de redécrire ce que fait le code en français/anglais mais plus pourquoi et/ou le contexte. Ou de résumer plusieurs lignes de code en une seule.