

Admin serveur Web

NGINX





Introduction : Admin Serveur Web

Introduction : Module Admin serveur Web

1 Environment Linux

2 Le shell scripting

3 Serveur Web Nginx





Introduction : Environnement Linux

Introduction : Environnement Linux

1

Environment Linux

2

Le système de fichier

3

La gestion des utilisateurs

4

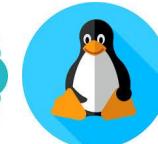
La gestion des paquets

5

Le service SSH

6

La gestion des processus



Introduction : Docker

7

Le réseaux sous Linux



Introduction : Environnement Linux



Introduction : Environnement Linux

1

Introduction

2

Présentation des familles Linux

3

Présentation du système Linux

4

Présentation du Terminal

5

Différence : Linux Serveur et OS



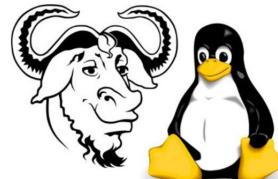
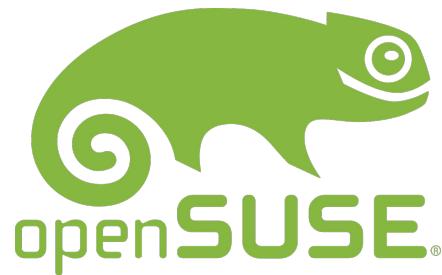
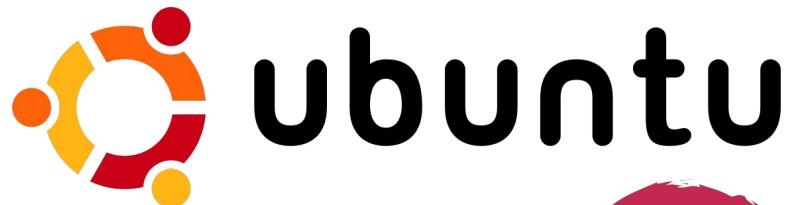
Introduction aux environnement sous Linux



Familles sous Linux



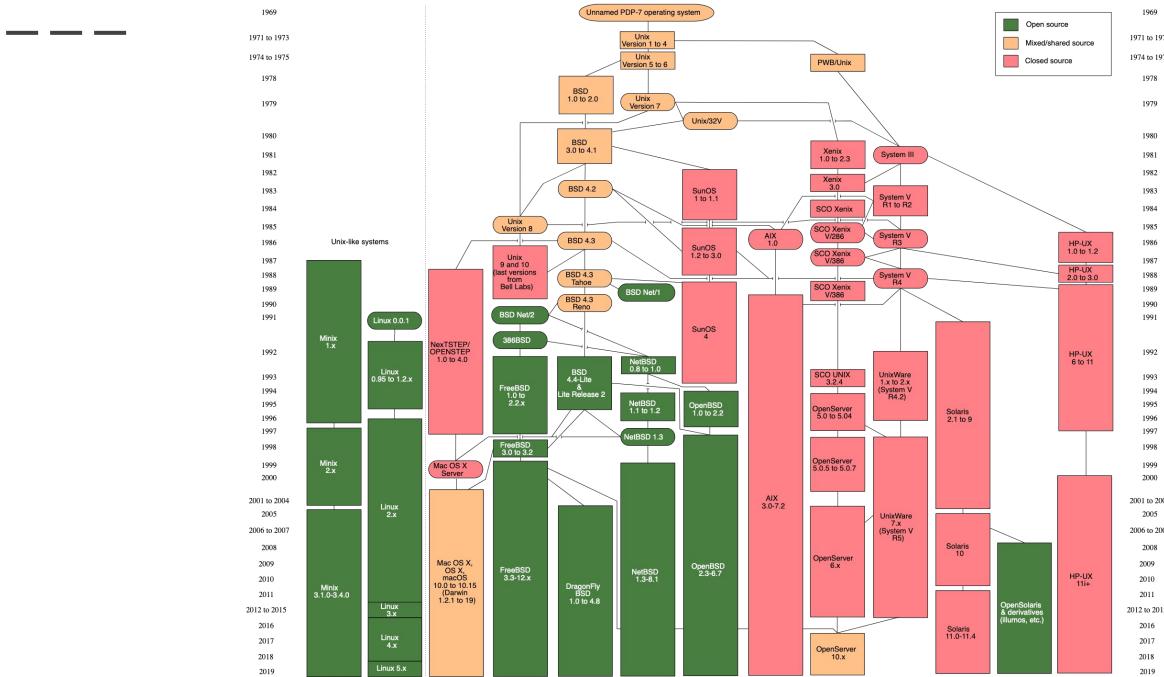
Introduction : Les familles Linux



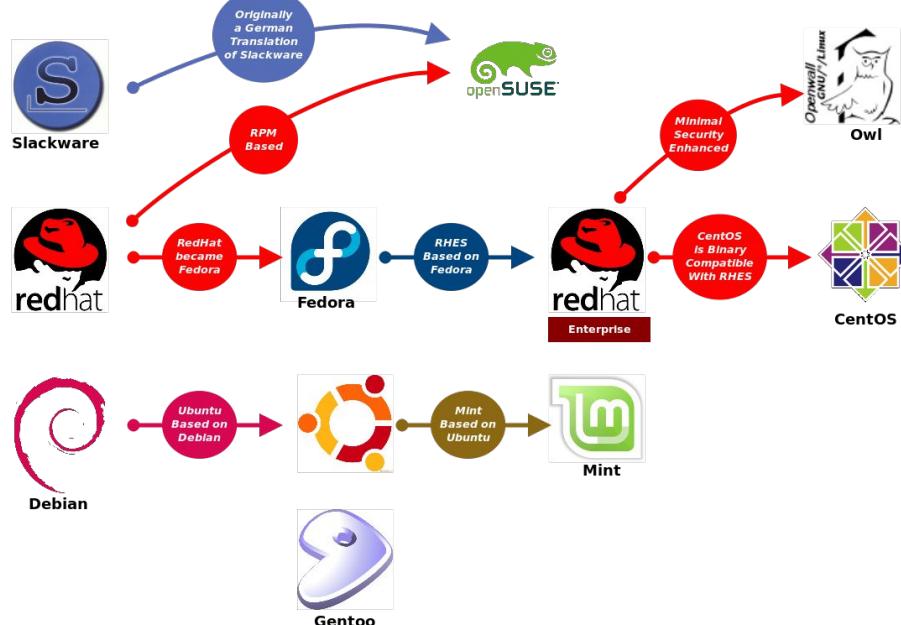
debian



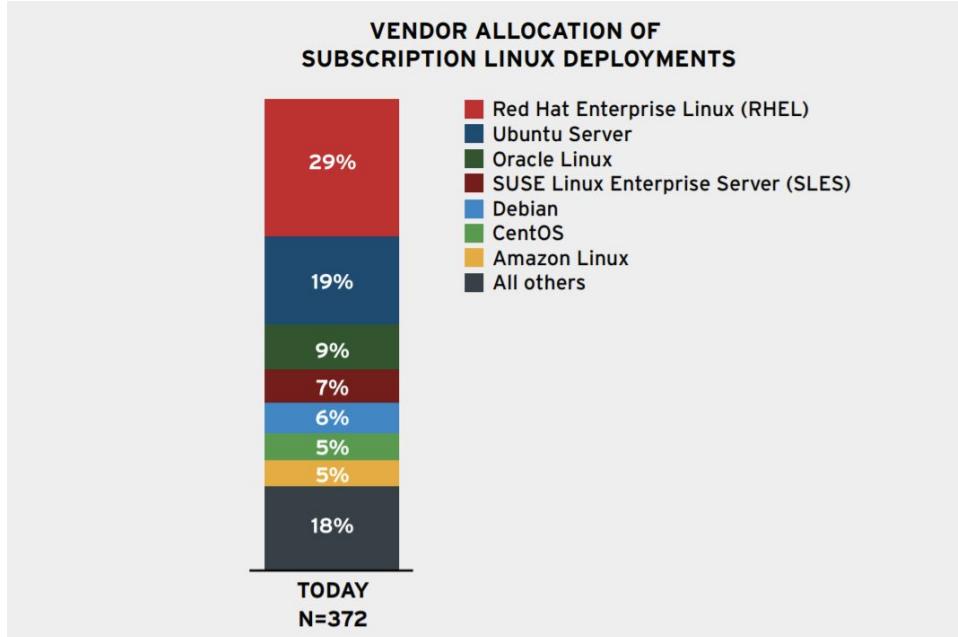
Introduction : Les familles Linux



Introduction : Les familles Linux



Introduction : Les familles Linux



Introduction : Les familles Linux

> OS vs Server Linux

> Il faut bien faire la différence entre :

Une distribution **server** et **OS sous** Linux

Server : Sans la partie graphique



Système sous Linux

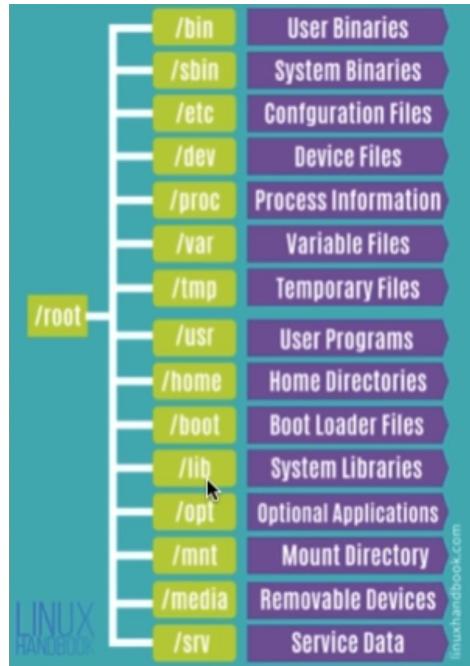


Introduction : Système Linux

- › **Hardware:** **Materiel physique**
- › **Linux Kernel:** **Coeur de l'OS**
- › **Shell:** **Coquille**
- › **Linux OS:** **Système d'opération**
- › **Applications - Utilities:** **Software**



Introduction : Système Linux



Présentation du cours : Linux



Introduction : Les familles Linux

- > Les commandes pour naviguer dans les dossiers
- > Créer des fichiers, afficher du contenu et des stats
- > Copier, renommer, supprimer des fichiers
- > Les commandes basiques pour gérer les process
- > Manipuler et parser des fichiers



Présentation du Terminal



Présentation du Terminal

- > Inscription sur **webminal** pour simuler le comportement d'un terminal sous Linux.



Présentation du Terminal

> **Man** : Manuel pour les commandes shell sous Linux



Différence OS / Serveur



Rappels : Différence OS / Serveur

> Exemple : Ubuntu Server vs Ubuntu Desktop

Ubuntu desktop inclut l'interface graphique utilisateur.



Commandes shell du Terminal



Basic commands to navigate into a directory



Basic commands

- > **Man** : Manuel
- > **PWD** : Current directory
- > **Ls** : List directory
- > **cd** : Change directory
- > **mkdir** : Make directory



Create files, display content and stats



Basic commands : Files

- > **touch** : create a file
- > **dir** : Display current directories
- > **clear** : Clean the screen
- > **Echo** : Display a line text



Basic commands : Files

- > **cat** : Display content file
- > **head** : Display specific line from the file beginning
- > **tail** : Display specific line from the file ending
- > **stat** : Stats related to a file



Copy, rename, Delete Files



Basic commands : Files

- > **du** : disk usage
- > **cp** : Copy
- > **md5sum** : Signature file
- > **Mv** : Move (rename) file



Basic commands : Files

- > **ln** : Symlink
- > **rm** : Remove file
- > **rmdir** : Remove directory



Basic process commands



Basic commands : Process

- > **ps** : Display processus
- > **kill, killall** : kill processus
- > **sleep** : Suspend an execution
- > **top** : Display and update processus



Basic commands : Process

- > **pidof** : search processus by id
- > **Nice, renice** : Execute a utility with a scheduler
- > **pstree** : Display a tree of processus
- > **time** : time execution for a command



Manipulate or parse file contents



Basic commands : File parsing and manip

- > **grep** : Search in file and folder
- > **wc** : Count (chars, lines, bytes) in file
- > **cut** : Remove section



Basic commands : File parsing and manip

- > **paste** : Merge line of files
- > **sort** : Sort or merge records of text
- > **diff** : Compare files line by line



Shell Script : Scripting



Introduction : Shell Script

1

Exécution d'un script

2

Les variables

3

Mise en place de tests

4

Les conditions

5

Les boucles for

6

Les variables de positionnement

Introduction : Shell Script

7

Saisir des données

8

La boucle while



Shell Script : Exécution d'un script



Shell script : Rappels

> Qu'est-ce qu'un Script ?

- > Les composants principaux d'un script
- > Comment utiliser des variables
- > Comment donner des arguments à votre script



Shell script : Rappels

> Qu'est-ce qu'un Script ?

- > Comment permettre à l'utilisateur de rentrer des données qui vont pouvoir être utilisées par le script
- > Utilisez l'IDE de votre choix



Shell script : Un script

- Contient une série de commandes
- Ces commandes sont exécutées par un interpréteur (le shell pour les scripts que nous allons voir) les unes après les autres
- Tout ce que vous pouvez taper en ligne de commande peut être inclus dans un script
- Le scripting est la méthode idéale pour l'automatisation de tâche (DevOPS)



Shell script : Rendre un script executable

- Il est important de mettre les droits sur le fichier contenant le script de manière à ce que celui-ci devienne exécutable.

```
chmod a+x script.sh
```

- Le chmod a+x permet de rajouter l'exécution sur tous les utilisateurs pour le fichier.sh



Shell script : Le shebang !

- Au début de chaque script bash, il est important de faire figurer le shebang :

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

- Il commence par un <#> (ce qui correspond la plupart du temps à des commentaires) puis d'un <!> et enfin le chemin menant au programme bash shell.
- En effet, nous y indiquons le chemin de l'interpréteur qui va lire le script.



```
#!/bin/sh  
#!/bin/csh  
#!/bin/zsh  
...
```

Shell script : Le shebang ! (2)

- > Si vous décidez de ne pas faire figurer de Shebang au début de votre script, alors les commandes à l'intérieur du script seront exécutées en utilisant votre propre shell.
- > Attention car certaines syntaxes varient lorsque l'on utilise le shell bash par rapport au shell zsh. C'est pourquoi il est fortement recommandé d'être propre et de spécifier l'interpréteur que votre script devra utiliser.



Shell script : Le shebang ! (3)

- Comme vous l'avez compris, le shebang fait référence à l'interpréteur qui sera utilisé.
- De ce fait, on peut lui spécifier n'importe quel interpréteur comme par exemple l'interpréteur Python, et écrire du code Python à l'intérieur du script :

```
#!/usr/bin/python
print('Bonjour je m'appelle Jordan')
```



Shell script : Les variables

- Ce sont des espaces de stockage qui possèdent un nom.
- Imaginez que vous associez à un nom, un contenu qui peut varier au fur et à mesure de votre script. C'est cela que l'on appelle variable !

NOM_DE_LA_VARIABLE="valeur"
- Les variables sont sensibles à la casse et par convention, on les met toujours en majuscules. Attention à ne pas mettre d'espace entre la variable, le signe = et les ".



Shell script : Les commentaires

- En bash, les commentaires sont des éléments figurants dans le script qui ne vont pas être interprétés par l'interpréteur.
- En effet, lorsque le script va détecter le caractère #, il ne va pas interpréter les éléments figurant après sur la même ligne excepté pour le shebang.

```
#!/bin/bash
echo "Bonjour je m'appelle Jordan"

#Ceci est un commentaire
#Ceci ne sera pas interprété
```



Shell Script : Les variables



Shell script : L'utilisation des variables

- Pour utiliser les variables et afficher le contenu associé, il faut faire précéder le nom de la variable par un \$.

```
#!/bin/bash
PRENOM="Jordan"
NOM="ASSOULINE"
echo "Bonjour $PRENOM $NOM et bienvenu"
```

```
./script.sh
```

```
Bonjour Jordan ASSOULINE et bienvenu
```



Shell script : L'utilisation des variables (2)

- Lorsque vous souhaitez inclure une variable dans un mot par exemple, vous pouvez utiliser les { }

```
#!/bin/bash
PRENOM="Jordan"
NOM="ASSOULINE"
AGE="16"
echo "Bonjour $PRENOM ${NOM}, vous avez
${AGE}ans"
```

```
./script.sh
```

```
Bonjour Jordan ASSOULINE, vous avez
16ans
```



Assigner les sorties de commandes à une variable

- Il est tout à fait possible d'assigner la sortie standard d'une commande à une variable en la mettant entre parenthèses.

```
#!/bin/bash
PRENOM="Jordan"
NOM="ASSOULINE"
MACHINE=$(hostname)
echo "Bonjour $PRENOM $NOM et bienvenu
sur la machine ${MACHINE}."
```

```
./script.sh
```

```
Bonjour Jordan ASSOULINE et bienvenu
sur la machine debian01
```



Assigner les sorties de commandes à une variable (2)

- Une autre syntaxe est également possible dans d'autres scripts qui utilisent le ` (alt + 7) à la place du \$().

```
#!/bin/bash
PRENOM="Jordan"
NOM="ASSOULINE"
MACHINE=`hostname`
echo "Bonjour $PRENOM $NOM et bienvenu
sur la machine ${MACHINE} ."
```

```
./script.sh
```

```
Bonjour Jordan ASSOULINE et bienvenu
sur la machine debian01
```



Attention au nom des variables !

- Attention à la syntaxe que l'on utilise pour les noms des variables.
- Quelques règles à respecter :
 - Les variables ne peuvent pas commencer par un chiffre mais peuvent en contenir.
 - Exemple : **1BONNEVARIABLE** n'est pas un bon nom
 - Exemple : **UNEBOONNE7VARIABLE** est un nom correct
 - Les variables ne peuvent pas contenir de tiret (-)
 - Exemple : **UNE-BONNE-VARIABLE** n'est pas un bon nom
 - Exemple : **UNE_BONNE_VARIABLE** est un nom correct
 - Les variables ne peuvent contenir que des underscores, majuscules, minuscules et chiffres :
 - Exemple : **UNE_BONNE@VARIABLE** n'est pas un bon nom
 - Exemple : **Une_bonne_7_variable** est un nom correct



Shell Script : Les tests



Shell script : Les tests

- > Lorsque vous tapez une commande, vous pouvez prendre le temps d'analyser la réponse du système et prendre une décision en fonction de cette réponse.
- > Il est tout à fait possible d'effectuer les mêmes étapes avec le scripting Shell grâce aux tests dont la syntaxe est la suivante :
[voici-la-condition-du-test-a-verifier]
- > Il est important de respecter les espaces après le [mais également avant le].



Shell script : Exemple de tests

- Vérifie si le fichier /home/jordan/bonjour existe :

[voici-la-condition-du-test-a-verifier]
- La commande nous retourne la valeur 0 (True) si le fichier existe
- La commande nous retourne la valeur 1 (False) si le fichier n'existe pas



Shell script : Exemple de tests

- Parmi les opérateurs principaux nous avons :
 - -e : 0 (True) si le fichier existe
 - -d : 0 (True) s'il s'agit d'un dossier
 - -r : 0 (True) si le fichier est disponible en lecture pour l'utilisateur
 - -s : 0 (True) si le fichier existe et n'est pas vide
 - -w : 0 (True) si le fichier est disponible en écriture pour l'utilisateur
 - -x : 0 (True) si le fichier est disponible en exécution pour l'utilisateur



Shell script : Les tests possibles

- Vous pouvez utiliser la commande 'help test' dans le shell bash pour obtenir les différents types de test qui existent.

```
orcl-ORCL10g:~$ help test
test: test [expr]
      Evaluate conditional expression.

      Exits with a status of 0 (true) or 1 (false) depending on
      the evaluation of EXPR. Expressions may be unary or binary. Unary
      expressions are often used to examine the status of a file. There
      are string operators and numeric comparison operators as well.

      The behavior of test depends on the number of arguments. Read the
      bash manual page for the complete specification.

File operators:
-a FILE      True if file exists.
-b FILE      True if file is block special.
-c FILE      True if file is character special.
-d FILE      True if file is a directory.
-e FILE      True if file exists.
-f FILE      True if file exists and is a regular file.
-g FILE      True if file is set-group-id.
-h FILE      True if file is a symbolic link.
-L FILE      True if file is a symbolic link.
```



Shell script : Les tests sur les chaînes de caractères

- Il est également possible de faire des tests sur des chaînes de caractères.

```
#!/bin/bash  
PRENOM='Jordan'  
[ -z $PRENOM ]  
echo $?
```

```
./script.sh
```

```
1
```

- Le `-z` nous renvoie 0 si la chaîne de caractère est vide et 1 si elle ne l'est pas.
- Le `$?` permet de demander d'afficher le retour de la dernière commande lancée



Shell script : Les tests sur les chaînes de caractères (2)

- Un autre test possible :

```
#!/bin/bash
PRENOM='Jordan'
[ -n $PRENOM ]
echo $?
```

```
./script.sh
1
```

- Le `-n` nous renvoie 1 si la chaîne de caractère est vide et 0 si elle ne l'est pas.



Shell script : Les tests sur les chaînes de caractères (3)

- Comparer deux chaînes entre elles :

```
#!/bin/bash  
PRENOM='Jordan'  
NOM='ASSOULINE'  
[ $PRENOM = $NOM ]  
echo $?
```

./script.sh

1

- En effet, on peut comparer les chaînes en utilisant le signe =.
- Le script nous renvoie 0 si les deux chaînes sont identiques et 1 si elles ne le sont pas.



Shell script : Les tests sur les chaînes de caractères (4)

- Comparer deux chaînes entre elles :

```
#!/bin/bash  
PRENOM='Jordan'  
NOM='ASSOULINE'  
[ $PRENOM != $NOM ]  
echo $?
```

```
./script.sh
```

```
1
```

- On peut vérifier si deux chaînes sont différentes grâce au " != "
- Le script nous renvoie 1 si les deux chaînes sont identiques et 0 si elles sont différentes.



Shell script : Les tests sur les chiffres

- De la même manière que les chaînes de caractères, il est tout à fait possible de comparer deux nombres entre eux.
- chiffre1 **-eq** chiffre2 : 0 si chiffre1 est égal à chiffre2
- chiffre1 **-ne** chiffre2 : 0 si chiffre1 est différent de chiffre2
- chiffre1 **-lt** chiffre2 : 0 si chiffre1 est plus petit que chiffre2
- chiffre1 **-le** chiffre2 : 0 si chiffre1 est plus petit ou égal que chiffre2
- chiffre1 **-gt** chiffre2 : 0 si chiffre1 est plus grand que chiffre2
- chiffre1 **-ge** chiffre2 : 0 si chiffre1 est plus grand ou égal que chiffre2



Shell Script : Les conditions



Shell script : L'utilisation du if

```
if [ condition-est-vraie ]
then
    command
    command2
fi
```

- > Lorsque l'on veut exécuter un certain nombre de commande si la condition est vraie, on utilise le if.
- > Ainsi si la condition est vraie, alors le script va exécuter les commandes situées après le "then".
- > Le "fi" marque la fin de la condition.



Shell script : Exemples d'utilisation

```
---  
#!/bin/bash  
  
touch /home/jordan/bonjour.sh  
  
if [ -e /home/jordan/bonjour.sh ]  
then  
    echo "Le fichier bonjour.sh a bien été créé"  
fi
```

```
./script.sh  
  
Le fichier bonjour.sh a bien été créé
```



Shell script : L'utilisation du if et du else

```
if [ condition-est-vraie ]
then
    command
else
    command
fi
```

- Si l'on veut agir sur la possibilité que la condition soit fausse, on peut utiliser le else.
- En effet, si la condition est vraie, alors on exécutera les commandes situées après le then.
- Si la condition est fausse, ce seront les commandes situées après le else qui seront exécutées.



Shell script : Exemples d'utilisation

```
---  
#!/bin/bash  
  
if [ -e /home/jordan/bonjour.sh ]  
then  
    echo "Le fichier bonjour.sh a bien été créé"  
else  
    echo "Le fichier bonjour.sh n'a pas été créé"  
fi
```

```
./script.sh
```

```
Le fichier bonjour.sh n'a pas été créé
```



Shell script : L'utilisation du if, du elif et du else

```
if [ condition-est-vraie ]
then
    command
elif [ condition-est-vraie ]
then
    command
else
    command
fi
```

- Le mot `elif` correspond à la contraction de `else` et `if`. En effet, il est possible d'indiquer au script d'exécuter des commandes en fonction de la validité d'une ou de l'autre condition.



Shell script : Exemples d'utilisation

```
#!/bin/bash
CHIFFRE1='16'
CHIFFRE2='17'
if [ $CHIFFRE1 -lt $CHIFFRE2 ]
then
    echo "$CHIFFRE1 est plus petit que $CHIFFRE2"
elif [ $CHIFFRE1 -gt $CHIFFRE2 ]
then
    echo "$CHIFFRE1 est plus grand que $CHIFFRE2"
else
    echo "$CHIFFRE1 est égal à $CHIFFRE2"
fi
```

```
./script.sh
```

```
16 est plus petit que 17
```



Shell Script : Les boucles for



Shell script : Les boucles for

- Lorsque vous voulez effectuer un certain nombre d'actions sur une liste d'objets, vous pouvez utiliser la boucle for.
- On fait débuter la boucle par le mot clé "for" puis on indique notre variable, suivi de "in" et de la liste des objets.
- Ensuite le mot clé "do" est l'équivalent du "then" dans les conditions.
- Enfin on ferme la boucle par le mot clé "done"



Shell script : Les boucles for

```
for VARIABLE in OBJET1 OBJET2 OBJET3 OBJETn  
do  
    command  
    command2  
done
```



Shell script : Exemple d'utilisation

```
#!/bin/bash

for CHIFFRE in 10 11 12 13
do
    echo "Chiffre : $CHIFFRE"
done
```

```
./script.sh
```

```
Chiffre : 10
Chiffre : 11
Chiffre : 12
Chiffre : 13
```



Shell script : Exemple d'utilisation

```
#!/bin/bash
CHIFFRES="10 11 12 13"
for CHIFFRE in $CHIFFRES
do
    echo "Chiffre : $CHIFFRE"
done
```

```
./script.sh
```

```
Chiffre : 10
Chiffre : 11
Chiffre : 12
Chiffre : 13
```



Shell Script : Les boucles while



Shell script : Les boucles while

- > Lorsque vous voulez effectuer une boucle, tant qu'un test est vérifié, vous pouvez utiliser la commande while.
- > On fait débuter la boucle par le mot clé "while" puis on indique notre test pour lequel nous allons boucler.
- > Ensuite le mot clé "do" est l'équivalent du "then" dans les conditions.
- > Enfin on ferme la boucle par le mot clé "done"



Shell script : Les boucles while

```
while [ la-condition-est-vraie ]
do
    command
    command2
done
```



Shell script : Exemple d'utilisation

```
#!/bin/bash

while [ -z $PRENOM ]
do
    read -p "Quel est votre prenom ?" PRENOM
Done
Echo "Votre prenom est $PRENOM"
```

```
./script.sh

Quel est votre prenom ?
Quel est votre prenom ?
Quel est votre prenom ? Jordan
Votre prenom est Jordan
```



Shell Script : Variables de position



Shell script : Variables de position

- > Les variables de position stockent le contenu des différents éléments de la ligne de commande utilisée pour lancer le script.
- > Il en existe 10 : \$0 jusqu'à \$9
- > Le script lui-même est stocké dans la variable \$0
- > Le premier paramètre est stocké dans la variable \$1
- > Le second paramètre est stocké dans la variable \$2



Shell script : Exemple

- En lançant le script :
- ./script.sh argument1 argument2

> Dans l'exemple ci-dessus où on lance le script en lui passant des arguments on va avoir les valeurs suivantes :

- > \$0 : script.sh
- > \$1 : argument1
- > \$2 : argument2



Shell script : Afficher les variables de position

- `#!/bin/bash`
 - `echo "Voici les paramètres utilisés : $@"`
-
- `./script.sh parametre1 parametre2`
 - Voici les paramètres utilisés : parametre1
parametre2



Shell script : Variables de position

- > \$# : récupère le nombre de paramètres (à partir du \$1)
- > \$* : récupère la liste des paramètres



Shell Script : Entrées utilisateurs : read



Shell script : La commande read

- La commande read permet d'accepter les données du STDIN (entrée standard), c'est-à-dire va permettre à l'utilisateur d'entrer des données.

```
•#!/bin/bash  
•echo "Quel est votre prenom ?"  
•read PRENOM
```

- Cette commande va nous permettre de stocker dans la variable "PRENOM" le contenu entré par l'utilisateur.



Shell Script : Code retour



Shell script : Code retour

- A chaque fois qu'une commande est exécutée, elle renvoie un code de sortie (exit code).
- Entier compris entre 0 et 255 (car codé sur 8 bits)
- Dans la plupart des langages de développement, une commande qui s'est exécutée correctement renvoie un code retour égal à 0.
- Si le code retour est différent de 0, alors c'est qu'une erreur s'est passée au moment de l'exécution du code.



Shell script : Obtenir le code retour

- Pour obtenir le code retour de la dernière commande exécutée, comme montré dans les précédentes vidéos, vous pouvez utiliser le code suivant :

```
echo $?
```



Shell script : Utilisation du code retour pour le ping

```
#!/bin/bash

HOTE=$1
NOMBRE_DE_PAQUETS=$2
ping -c $NOMBRE_DE_PAQUETS $HOTE

if [ "$?" -ne "0" ]
then
    echo "L'hôte $HOTE n'est pas joignable"

else
    echo "L'hôte $HOTE est joignable"
fi
```



Shell script : Code retour personnel

- Il est tout à fait possible de dire à votre script que dans certaines conditions, il quitte avec un code erreur différent de 0.
- En effet, s'il s'est correctement exécuté, il sortira avec un code égal à 0 (ou égal à celui de la dernière commande exécutée), mais vous pouvez lui spécifier un code différent avec la commande :

```
exit 1  
exit 2
```



Shell Script : && / AND et || / OU



Shell script : Le and

- Maintenant que vous avez compris à quoi font référence les codes erreurs, il est ais  de comprendre comment fonctionne le &&.
- && = AND
- Il permet d'ex cuter une deuxi me commande uniquement lorsque la premi re a renvoy  un code erreur gal  0 (signifiant qu'elle s'est bien ex cut e).



Shell script : Le ou

- Il = OU
- Il permet d'exécuter une deuxième commande uniquement lorsque la première a renvoyé un code erreur différent de 0 (signifiant qu'elle ne s'est pas exécutée de la bonne manière).



Shell Script : Les fonctions



Shell script : Les fonctions

- Permet d'écrire un block de code une fois, et de le réutiliser par la suite dans le script.
- Eviter d'avoir à retaper du code identique, pour effectuer plusieurs tâches similaires dans son script.
- Permet d'aérer et d'améliorer l'ergonomie du script et du code.



Shell script : Les fonctions

- Beaucoup plus facile de faire évoluer son script par la suite, en ajoutant d'autres fonctions.



Shell script : Syntaxe

```
#!/bin/bash

#Déclarer une fonction en utilisant le mot "function"
function je-suis-une-fonction() {
    command1
    command2
    commandn
}

#Déclarer une fonction sans spécifier le mot "function"
Je-suis-une-fonction() {
    command1
    command2
    commandn
}
```



Shell script : Appeler une fonction dans un script

```
-----  
#!/bin/bash  
  
function internet() {  
ping -c 1 8.8.8.8  
  
if [ $? -eq 0 ]  
then  
    echo "La connectivité vers internet est  
établie"  
else  
    echo "Pas de connectivité vers internet"  
fi  
}  
  
internet
```



Shell script : Les paramètres

- > Tout comme les scripts eux-mêmes, les fonctions peuvent accepter des paramètres.
- > De la même manière, le premier paramètre est stocké dans le \$1, le second dans le \$2, etc...
- > Attention, le \$0 fait référence au nom du script lui-même et non pas au nom de la fonction.



Shell script : Utilisation des paramètres dans une fonction

```
-----  
#!/bin/bash  
  
function internet() {  
ping -c $1 $2  
  
if [ $? -eq 0 ]  
then  
    echo "La connectivité vers internet est  
établie"  
else  
    echo "Pas de connectivité vers internet"  
fi  
}  
  
Internet "1" "8.8.8.8"
```



Shell script : Les variables

- Les variables globales peuvent tout à fait être utilisées dans des fonctions à condition qu'elles aient été déclarées avant la fonction.

```
#!/bin/bash

VARIABLE="Jordan"

function demonstration() {
    echo "La variable $VARIABLE est utilisable"
}
```



Shell script : Les variables (2)

- Cependant les variables déclarées dans une fonction ne peuvent être utilisées qu'une fois que la fonction a été exécutée

```
#!/bin/bash

VARIABLE="Jordan"

function demonstration() {
    echo "La variable $VARIABLE est utilisable"
    VARIABLE_IN_FUNCTION="1"
}
#VARIABLE_IN_FUNCTION n'est pas utilisable

demonstration
#VARIABLE_IN_FUNCTION est désormais utilisable
```



Shell script : Les variables locales

- Les variables locales ne peuvent qu'être utilisées au sein d'une fonction.

```
#!/bin/bash

VARIABLE="Jordan"

function demonstration() {
    echo "La variable $VARIABLE est utilisable"
    local VARIABLE_IN_FUNCTION="1"
}

demonstration
#VARIABLE_IN_FUNCTION n'est pas utilisable
```



Shell script : Le code retour d'une fonction

- Les fonctions possèdent un code de retour qui peut être explicitement indiqué grâce à la commande return.

```
#!/bin/bash

PRENOM="Jordan"

function demonstration() {
    echo "Bonjour je m'appelle $PRENOM"
    return 0
}
```

- Si ce code retour n'a pas été défini, alors c'est le code retour de la dernière variable exécutée dans la fonction qui sera celui de la fonction par défaut.



Shell script : Le code retour d'une fonction (2)

- Le code retour d'une fonction est compris entre 0 et 255.

```
#!/bin/bash

function internet_connectivity() {
    ping -c 1 8.8.8.8 && return 0
}

internet_connectivity

if [ $? -eq 0 ]
then
echo "Connectivité vers internet"
fi
```

- On peut y accéder grâce à la commande \$?



Shell Script : Le case



Shell script : Une alternative au if

- Lorsque l'on souhaite faire des conditions complexes, à savoir plusieurs elif comme par exemple ci-dessous :

```
#!/bin/bash

read -p "Quel est votre age ?" AGE

if [ $AGE -lt 18 ]
then
echo "Vous êtes mineur"
elif [ $AGE -gt 18 ]
then
echo "Vous êtes majeur"
elif [ $AGE -eq 18 ]
then
echo "vous êtes majeur et vous avez 18 ans"
fi
```



Shell script : Syntaxe

```
---  
>      case "$VARIABLE" in  
           premier_cas)  
                   commande1  
                   commande2  
                   commanden  
                   ;;  
           deuxieme_cas)  
                   commande1  
                   commande2  
                   commanden  
                   ;;  
           troisieme_cas)  
                   commande1  
                   commande2  
                   commanden  
                   ;;  
esac
```



Shell script : Exemple

```
case "$1" in
    start)
        /etc/init.d/apache2 start
        ;;
    stop)
        kill $(cat /var/run/apache2/apache2.pid)
        ;;
    *)
        echo "Merci d'indiquer start ou stop"
        exit 1
        ;;
esac
```



Shell script : Exemple (2)

```
case "$1" in
    start|START)
        /etc/init.d/apache2 start
        ;;
    stop|STOP)
        kill $(cat /var/run/apache2/apache2.pid)
        ;;
    *)
        echo "Merci d'indiquer start ou stop"
        exit 1
        ;;
esac
```

