

Systemes d'exploitation 420-W12-SF

Utilisation des scripts Bash

Jean-Pierre Duchesneau, Hiver 2022

Les cours

1. Intro aux infrastructures informatiques et les composantes internes du PC
2. Les composantes internes du PC et réseau
3. Système d'Exploitation
4. Virtualisation de clients et de serveurs
5. Disque dur, partition et système de fichier
6. La ligne de commandes (Shell) et les scripts
 1. CMD
 2. Linux
 3. Bash
 4. **Les scripts**
7. Git, le contrôle de version
8. Installation d'application et WAMP

Un script

Contient une série de commandes.

Ces commandes sont exécutés par un interpréteur 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 tâches, notamment dans la mouvance DevOps

Langage de scripts les plus utilisés

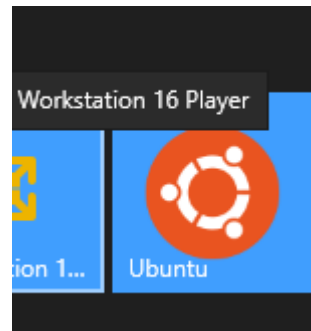
Bash

Power Shell

Python

Bash

- **Bash** (acronyme de ***B**ourne-**A**gain **s**hell*) est un interpréteur en ligne de commande de type script.
- C'est le shell Unix du projet GNU.
- Bash est un logiciel libre publié sous licence publique générale GNU.



Ubuntu sous Windows

```
MINGW64:/c
jpduches@Bilbo MINGW64 /c
$ pwd
/c

jpduches@Bilbo MINGW64 /c
$ ls
'$Recycle.Bin'/' Recovery/
'$WINRE_BACKUP_PARTITION.MARKER' SQL2019/
'$WinREAgent'/' Source/
BOOTNXT 'System Volume Information'/'
Config.Msi/' Users/
'Documents and Settings'@ VMActives/
DumpStack.log Windows/
DumpStack.log.tmp 'avast! sandbox'/'
Garmin/ bootmgr
HPLJP1000_P1500_Series.log h21-w12-se-4392-jpd/
OneDriveTemp/ hiberfil.sys
PerfLogs/ pagefile.sys
'Program Files'/' process.txt
'Program Files (x86)'/' swapfile.sys
ProgramData/ xampp/

jpduches@Bilbo MINGW64 /c
$
```

Bash sous Windows

L'aide sur Bash

La documentation de Bash est disponible en ligne, comme celle de la plupart des logiciels GNU.

<https://www.gnu.org/software/bash/>

Vous pouvez également trouver des informations sur Bash en exécutant

- **info bash** ou
- **man bash**, ou en consultant
- **/usr/share/doc/bash/**, **/usr/local/share/doc/bash/**,
- ou des répertoires similaires sur votre système.
- Un résumé est disponible en exécutant **bash --help**.

Rendre le script exécutable

➤ Modifier les droits : **chmod a+x script.sh**

➤ Le chemin d'exécution :

Chemin absolu :
/home/jpduches/script.sh

Chemin relatif (si je suis dans le même répertoire que le script) :
./script.sh

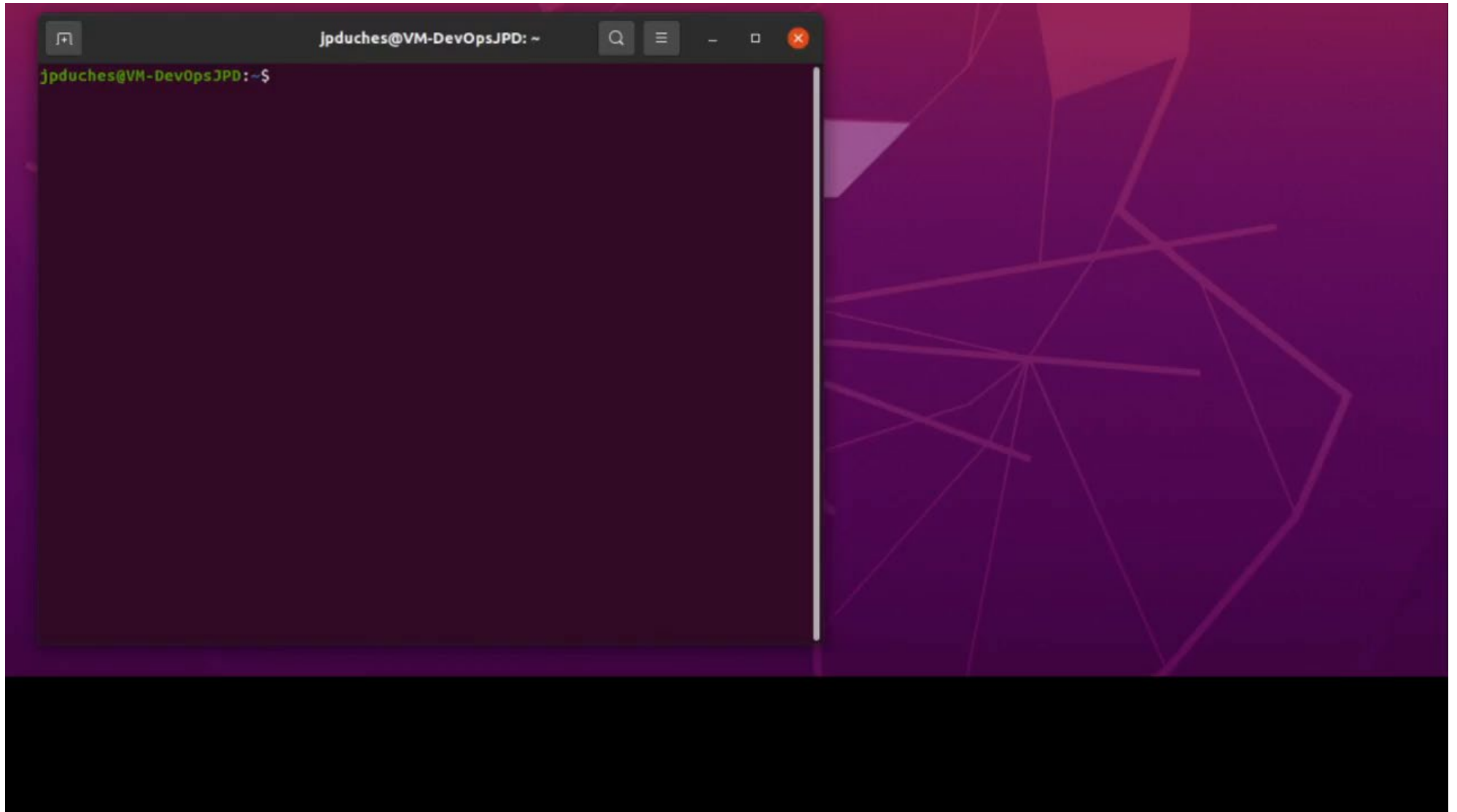
➤ Le shebang au début du script bash :

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

➤ D'autres shebang

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

Exemple avec un autre interpréteur :



Les variables et commentaires

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

```
NOM_DE_LA_VARIABLE="valeur"
```

```
Ouvrir ▼ [+]
```

```
*script.sh
```

```
1 #!/bin/bash
2 NOM="Jean-Pierre Duchesneau"
3 #Ici j'ai un commentaire
4 echo $NOM
5
```



```
jpduches@VM-DevOpsJPD: ~
```

```
jpduches@VM-DevOpsJPD:~$ ./script.sh
Jean-Pierre Duchesneau
jpduches@VM-DevOpsJPD:~$
```

Les quotes

Il est possible d'utiliser des **quotes** pour délimiter un paramètre contenant des espaces. Il existe trois types de quotes :

- les apostrophes ' ' (simples quotes) ;
- les guillemets " " (doubles quotes) ;
- les accents graves ` ` (back quotes).

Selon le type de quotes que vous utilisez, la réaction de bash ne sera pas la même.

Les simples quotes ' '

```
jpduches@Bilbo:~/scripts$ message='Bonjour tous le monde'
jpduches@Bilbo:~/scripts$ echo $message
Bonjour tous le monde
jpduches@Bilbo:~/scripts$ echo 'Le message est : $message'
Le message est : $message
jpduches@Bilbo:~/scripts$
```

Avec de simples quotes, la variable n'est pas analysée et le \$ est affiché tel quel.

Les doubles quotes " "

```
jpduches@Bilbo:~/scripts$ echo "Le message est : $message"
Le message est : Bonjour tous le monde
jpduches@Bilbo:~/scripts$
```

ça fonctionne ! Cette fois, la variable est analysée et son contenu affiché.

Les back quotes

Un peu particulières, les back quotes demandent à bash **d'exécuter** ce qui se trouve à l'intérieur.

Un exemple valant mieux qu'un long discours, regardez la première ligne :

```
jpduches@Bilbo:~/scripts$ message=`pwd`  
jpduches@Bilbo:~/scripts$ echo "Vous êtes dans le dossier $message"  
Vous êtes dans le dossier /home/jpduches/scripts  
jpduches@Bilbo:~/scripts$ _
```

La commande **pwd** a été exécutée et son contenu inséré dans la variable **message** !
Nous avons ensuite affiché le contenu de la variable.
Cela peut paraître un peu tordu, mais c'est réellement utile.

Utilisation des variables

- Pour utiliser les variables et afficher le contenu associé, il faut faire précéder le nom de la variable par un \$.
- Lorsque vous souhaitez inclure une variable dans un mot, vous pouvez utiliser {} :

```
print( "Bonjour je m'appelle Jean Pierre" )
jpduches@Bilbo:~/scripts$ cat script3
#!/bin/bash
#Définition des variables
NOM="Duchesneau"
PRENOM="Jean-Pierre"
EMAIL="jpduchesneau@csfoyc.ca"
AGE=58
DATENAISSANCE=1963-04-29
MACHINE=`hostname`
#Programme

echo "Bonjour $PRENOM ${NOM}, vous avez ${AGE}ans."
echo "Bienvenu sur la machine ${MACHINE}."

jpduches@Bilbo:~/scripts$ ./script3
Bonjour Jean-Pierre Duchesneau, vous avez 58ans.
Bienvenu sur la machine Bilbo.
jpduches@Bilbo:~/scripts$
```

Attention la back cote.

```
jpduches@Bilbo:~/scripts$ MACHINE=${hostname}
jpduches@Bilbo:~/scripts$ echo $MACHINE

jpduches@Bilbo:~/scripts$ MACHINE=`hostname`
jpduches@Bilbo:~/scripts$ echo $MACHINE
Bilbo
jpduches@Bilbo:~/scripts$
```

Les tests

- Lorsque vous taper une commande, vous pouvez prendre le temps d'analyser la réponse du système et prendre une décision en fonction de cette réponse.
- Avec le scripting bash il est possible de faire des tests en utilisant la syntaxe suivante :

[voici-la-condition-du-test-a-vérifier]

- Il est important de respecter les espaces après le [et aussi avant le].

Les tests

Exemple de tests :

Vérifier si le fichier /home/jpduches/bonjour existe. Renvoi 0 (true) 1 (false)

```
[ -e /home/jpduches/bonjour ]  
$echo $?
```

```
jpduches@VM-DevOpsJPD:~$ [ -e /home/jpduches/bonjour.py ]  
jpduches@VM-DevOpsJPD:~$ echo $?  
0  
jpduches@VM-DevOpsJPD:~$ [ -e /home/jpduches/bonjour ]  
jpduches@VM-DevOpsJPD:~$ echo $?  
1  
jpduches@VM-DevOpsJPD:~$
```

Opérateur principaux :

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

```
$help test
```

Liste des tests possibles :

```
jpduches@Bilbo:~/scripts$ 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.
    -k FILE      True if file has its 'sticky' bit set.
    -p FILE      True if file is a named pipe.
    -r FILE      True if file is readable by you.
    -s FILE      True if file exists and is not empty.
    -S FILE      True if file is a socket.
    -t FD        True if FD is opened on a terminal.
    -u FILE      True if the file is set-user-id.
    -w FILE      True if the file is writable by you.
    -x FILE      True if the file is executable by you.
    -O FILE      True if the file is effectively owned by you.
    -G FILE      True if the file is effectively owned by your group.
    -N FILE      True if the file has been modified since it was last read.
```

Variable de positionnement

- Les variable 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

```
#!/bin/bash

echo "Le premier argument a pour valeur $1"
echo "Le second argument a pour valeur $2"

echo "les arguments ont pour valeur : $@"
echo "Le nombre d'argument est de $#"
```

```
jpduches@VM-DevOpsJPD:~$ ./position.sh Jean-Pierre Duchesneau
Le premier argument a pour valeur Jean-Pierre
Le second argument a pour valeur Duchesneau
les arguments ont pour valeur : Jean-Pierre Duchesneau
Le nombre d'argument est de 2
Les arguments ont pour valeurs Jean-Pierre Duchesneau
jpduches@VM-DevOpsJPD:~$
```

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

Les conditions : if, elif, else

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

Si la condition est vraie, les commandes situées après le then sont exécutées.

Si la condition est fausse, les commandes situées après le else sont exécutées.

```
jpduches@VM-DevOpsJPD:~$ ls
bonjour.py  Documents  Modèles  Public  Téléchargements
Bureau      Images     Musique  script.sh  Vidéos
jpduches@VM-DevOpsJPD:~$ if [ -e ./script.sh ]
> then echo "le fichier existe"
> else echo "le fichier n'existe pas"
> fi
le fichier existe
jpduches@VM-DevOpsJPD:~$
```

```
#!/bin/bash

CHIFFRE1='16'
CHIFFRE2='15'

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 égale à $CHIFFRE2"
fi
```

```
jpduches@VM-DevOpsJPD:~$ ./script.sh
16 est plus grand que 15
jpduches@VM-DevOpsJPD:~$
```

Boucles :

for
do
done

```
#!/bin/bash  
  
CHIFFRES="10 11 12 13 14"  
  
for CHIFFRE in $CHIFFRES  
do  
echo "CHIFFRE : $CHIFFRE"  
done
```

```
jpduches@VM-DevOpsJPD:~$ ./boucle.sh  
CHIFFRE : 10  
CHIFFRE : 11  
CHIFFRE : 12  
CHIFFRE : 13  
CHIFFRE : 14  
jpduches@VM-DevOpsJPD:~$
```

While [la condition-est-vraie]

Do
◦ Command
done

```
#!/bin/bash  
  
while [ -z $PRENOM ]  
do  
read -p "Quel est votre prnénom ? " PRENOM  
done  
  
echo "Votre prénom est $PRENOM"
```

```
jpduches@VM-DevOpsJPD:~$ ./while.sh  
Quel est votre prnénom ?  
Quel est votre prnénom ?  
Quel est votre prnénom ? Jean-Pierre  
Votre prénom est Jean-Pierre  
jpduches@VM-DevOpsJPD:~$
```

Fonctions

```
#!/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
```

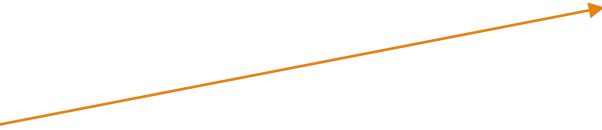
```
jpduches@VM-DevOpsJPD:~$ ./fonction.sh
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 octets de 8.8.8.8 : icmp_seq=1 ttl=116 temps=5.56 ms

--- statistiques ping 8.8.8.8 ---
1 paquets transmis, 1 reçus, 0 % paquets perdus, temps 0 ms
rtt min/avg/max/mdev = 5.556/5.556/5.556/0.000 ms
La connectivité vers internet est établie
jpduches@VM-DevOpsJPD:~$
```

Le code retour d'une commande

- 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.
- Le man nous informe des code :

man ls :



```
change the settings. Use the dircolors command to set it.  
  
Exit status:  
  0      if OK,  
  
  1      if minor problems (e.g., cannot access subdirectory),  
  
  2      if serious trouble (e.g., cannot access command-line argument).  
  
AUTHOR  
       Written by Richard M. Stallman and David MacKenzie.  
  
REPORTING BUGS
```

Obtenir le code de retour

- Pour obtenir le code retour de la dernière commande exécutée, il faut utiliser le code :

echo \$?

```
jpduches@Bilbo: ~  
jpduches@Bilbo:~$ pwd  
/home/jpduches  
jpduches@Bilbo:~$ echo $?  
0  
jpduches@Bilbo:~$ ls -l bonjour  
ls: cannot access 'bonjour': No such file or directory  
jpduches@Bilbo:~$ echo $?  
2  
jpduches@Bilbo:~$
```

Utilisation du code retour pour le ping

On peut utiliser le code retour du ping pour vérifier la connectivité vers un hôte de notre équipement :

```
#!/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 "
    exit 1
else
    echo "L'hôte $HOTE est joignable "
    exit 0
fi
```

Allez plus loin :

- Bash scripting cheatsheet (devhints.io)
- [Bash Notes For Professionals](#)