

Infrastructure technologique et virtualisation

Module 6 : Utilisation des scripts

Bash

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

PowerShell

- Est à la fois un interpréteur de commandes et un puissant langage de script.
- Il tire sa puissance en grande partie grâce au Framework .NET sur lequel il s'appuie.

le Framework .NET est une immense bibliothèque de classes à partir desquelles nous ferons naître des objets ; objets qui nous permettront d'agir sur l'ensemble du système d'exploitation

- Un jeu de commandes d'administration et de gestion système extrêmement riche qui ne cesse de croître à chaque nouvelle version.
- Destiné à automatiser les tâches et permet de développer des outils.

- Windows PowerShell v1 en 2006
- Windows PowerShell v5.1/PowerShell Core 2016
- Depuis 2016, avec la version PowerShell Core, il est multiplateforme et Open Source.
 - La version Core ne s'appuie plus sur Framework .NET mais sur .NET Core
 - Voir le tableau à la page suivante.

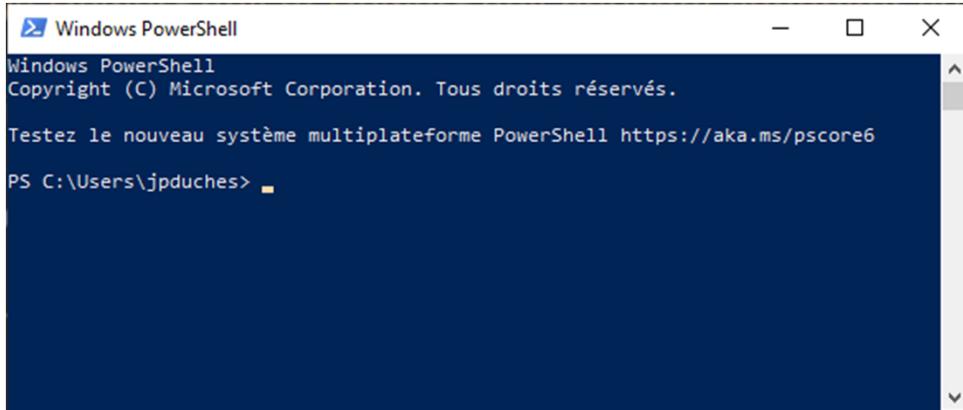
Lien vers le
code source



Historique des mises en production

Le tableau suivant contient une chronologie des versions majeures de PowerShell. Ce tableau est fourni à des fins de référence historique. Il n'est pas destiné à déterminer le cycle de vie de la prise en charge.

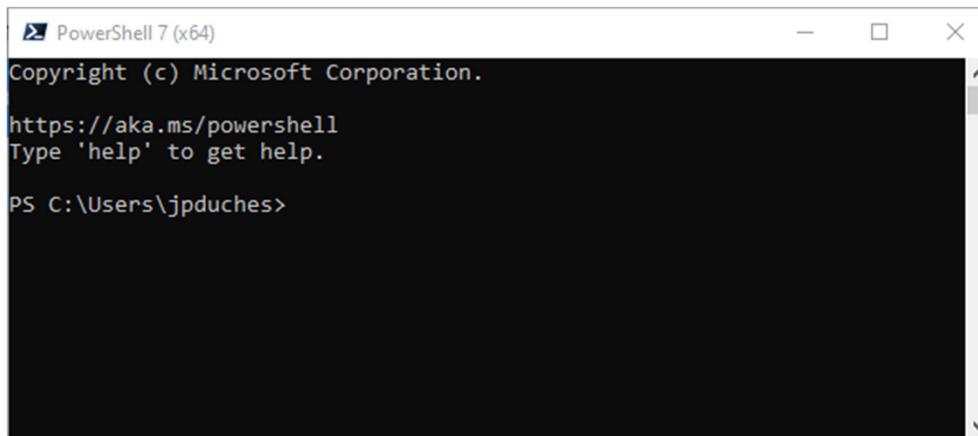
| Version | Date de sortie | Remarque |
|----------------------------|----------------|---|
| PowerShell 7.1 (actuel) | Nov-2020 | Basée sur .NET Core 5.0 (actuel). |
| PowerShell 7.0 (LTS) | Mars 2020 | Basée sur .NET Core 3.1 (LTS). |
| PowerShell 6.2 | Mars 2019 | |
| PowerShell 6.1 | Septembre 2018 | Basée sur .NET Core 2.1. |
| PowerShell 6.0 | Janvier 2018 | Première version, basée sur .NET Core 2.0. Installable sur Windows, Linux et macOS. |
| PowerShell 5.1 | Août 2016 | Publiée dans Mise à jour anniversaire Windows 10 et Windows Server 2016. |



```
Windows PowerShell
Copyright (C) Microsoft Corporation. Tous droits réservés.

Testez le nouveau système multiplateforme PowerShell https://aka.ms/pscore6

PS C:\Users\jpduches>
```



```
PowerShell 7 (x64)
Copyright (c) Microsoft Corporation.

https://aka.ms/powershell
Type 'help' to get help.

PS C:\Users\jpduches>
```

Histoire de PowerShell

- En 2020, PowerShell Core devient PowerShell
- Windows PowerShell et PowerShell peuvent cohabiter sur la même machine Windows
 - Windows PowerShell se lance avec l'exécutable powershell.exe
 - PowerShell se lance avec l'exécutable pwsh.exe
- Dans un futur proche : PowerShell sera intégré à Windows.
- Il est aussi multi plateforme (prochaine diapo)

Source : Cloud Horizon, 2021, **PowerShell - Maitrisez les bases fondamentales, cours Udemy.**

PowerShell sur Ubuntu

```
j pduches@Bilbo: ~
j pduches@Bilbo:~$ pwd
/home/j pduches
j pduches@Bilbo:~$ ls
n21-w12-se-4392-jpd p1 packages-microsoft-prod.deb packages-microsoft-prod.deb.1 scripts
j pduches@Bilbo:~$ pwsh
PowerShell 7.1.3
Copyright (c) Microsoft Corporation.

https://aka.ms/powershell
Type 'help' to get help.

PS /home/j pduches> get-command
```

| CommandType | Name | Version | Source |
|-------------|---------------------------------|---------|------------------------------|
| Function | cd.. | ----- | ----- |
| Function | cd\ | | |
| Function | Clear-Host | | |
| Function | Compress-Archive | 1.2.5 | Microsoft.PowerShell.Archive |
| Function | Configuration | 2.0.5 | PSDesiredStateConfiguration |
| Function | Expand-Archive | 1.2.5 | Microsoft.PowerShell.Archive |
| Function | Find-Command | 2.2.5 | PowerShellGet |
| Function | Find-DSCResource | 2.2.5 | PowerShellGet |
| Function | Find-Module | 2.2.5 | PowerShellGet |
| Function | Find-RoleCapability | 2.2.5 | PowerShellGet |
| Function | Find-Script | 2.2.5 | PowerShellGet |
| Function | Get-CredsFromCredentialProvider | 2.2.5 | PowerShellGet |
| Function | Get-DscResource | 2.0.5 | PSDesiredStateConfiguration |
| Function | Get-InstalledModule | 2.2.5 | PowerShellGet |
| Function | Get-InstalledScript | 2.2.5 | PowerShellGet |
| Function | Get-PSScriptAnalyzer | 2.2.5 | PowerShellGet |
| Function | help | | |
| Function | Install-Module | 2.2.5 | PowerShellGet |
| Function | Install-Script | 2.2.5 | PowerShellGet |
| Function | Invoke-DscResource | 2.0.5 | PSDesiredStateConfiguration |
| Function | New-DscChecksum | 2.0.5 | PSDesiredStateConfiguration |

Python

Python est un excellent langage de programmation qui vous permet d'être productif dans une grande variété de domaines.

Python, c'est aussi un logiciel appelé interpréteur. L'interpréteur est le programme dont vous aurez besoin pour exécuter le code et les scripts Python. Techniquelement, l'interpréteur est une couche de logiciel qui s'intercale entre votre programme et votre matériel informatique pour faire fonctionner votre code.

Selon l'implémentation de Python que vous utilisez, l'interpréteur peut être :

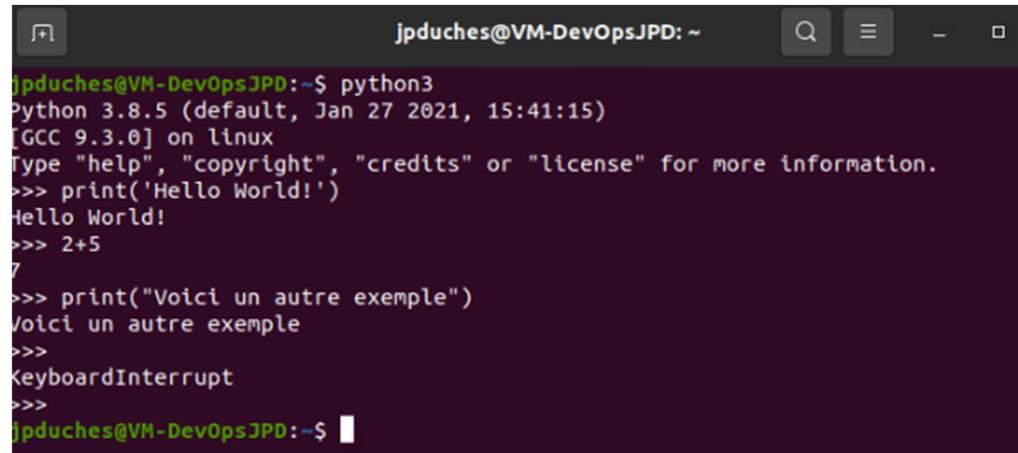
- **Un programme écrit en C, comme CPython, qui est l'implémentation de base du langage.**
- Un programme écrit en Java, comme Jython
- un programme écrit dans Python lui-même, comme PyPy
- un programme implémenté en .NET, comme IronPython.

Python

La première condition pour pouvoir exécuter des scripts Python est d'avoir l'interpréteur correctement installé sur votre système.

L'interpréteur est capable d'exécuter du code Python de deux manières différentes :

- En tant que script ou module
- Comme un morceau de code tapé dans une session interactive

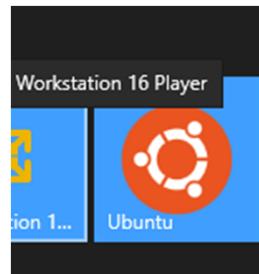


A screenshot of a terminal window titled "jpduches@VM-DevOpsJPD:~". The window shows a Python 3.8.5 interactive session. The user types commands and sees their results:

```
j pduches@VM-DevOpsJPD:~$ python3
Python 3.8.5 (default, Jan 27 2021, 15:41:15)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print('Hello World!')
Hello World!
>>> 2+5
7
>>> print("Voici un autre exemple")
Voici un autre exemple
>>>
KeyboardInterrupt
>>>
j pduches@VM-DevOpsJPD:~$
```

Bash

- Bash (acronyme de *Bourne-Again shell*) 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

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

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

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**

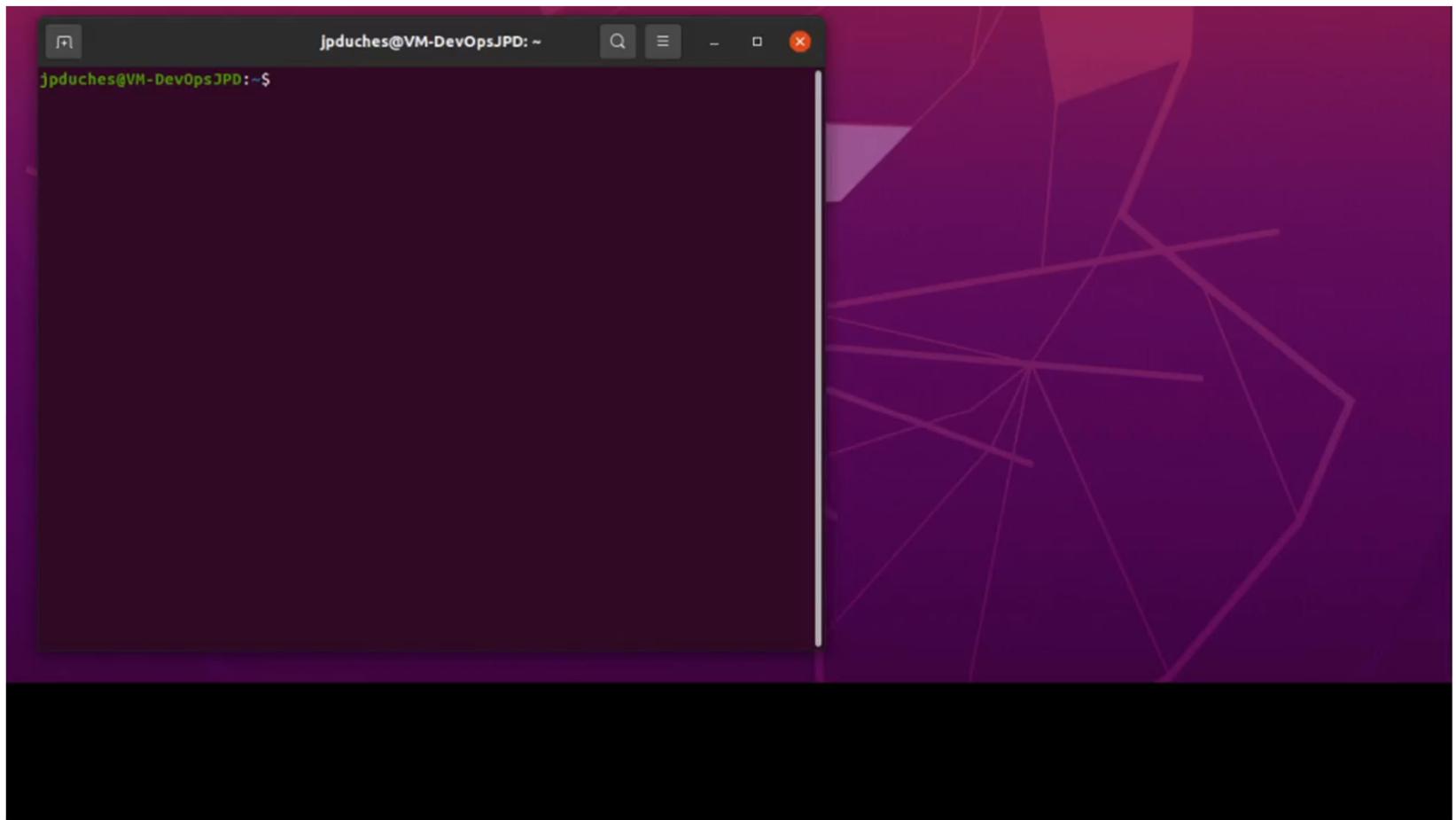
➤ D'autres shebang

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

Trouver l' interpréteur :

```
jpduches@VM-DevOpsJPD:~$ ls -al /usr/bin/ |grep bash
-rwxr-xr-x 1 root root 1183448 jun 18 2020 bash
-rwxr-xr-x 1 root root 6794 jun 18 2020 bashbug
-rwxr-xr-x 1 root root 2446 jan 25 2020 dh_bash-completion
lrwxrwxrwx 1 root root 4 jun 18 2020 rbash -> bash
jpduches@VM-DevOpsJPD:~$
```

Exemple avec un autre interpréteur :



Les variables et commentaires

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

NOM_DE_LA_VARIABLE="valeur"

The diagram illustrates the execution of a Bash script. On the left, a screenshot of a code editor shows the file `*script.sh` containing the following code:

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

An orange arrow points from the code editor to a terminal window on the right. The terminal window shows the command `j pduches@VM-DevOpsJPD:~$./script.sh` being run, followed by the output `Jean-Pierre Duchesnau`, and ends with the prompt `j pduches@VM-DevOpsJPD:~$`.

Les tests

Exemple de tests :

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

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

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

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

Variable de positionnement

- Stockent le contenu
- 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
- Si plus de 9 arguments on utilise la commande shift (voir page suivante)

```
#!/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 $#"

echo "Les arguments ont pour valeurs $*"
```

```
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
```

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

While [la condition-est-vraie]

```
Do  
  ◦ Command  
done
```

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

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

Opération

Trois méthodes permettent d'effectuer des calculs :

La première utilise la syntaxe spéciale `$((operation))`

```
$a=1  
$a=$(($a + 1))  
$echo $a  
2
```

La seconde utilise la commande `let`

```
$a=1  
$let "a=$a + 1"  
$echo $a  
2
```

La troisième utilise la commande `bc`,
qui accepte aussi les nombres décimaux.

```
$a=1  
$echo $(echo " $a + 1" |bc )  
$echo $a  
2
```