PowerShell

Un `bash` pour Windows !

- *Tab-completion*
- Ctrl-c
- `cmd1 | cmd2`
- `cmd1 > fichier`
- `cmd1 ; cmd2`

The big difference

- Les *Cmdlet* PowerShell traitent des objets
  - Les commandes UNIX traitent des chaines de caractères

```
# Un ls -l Perlimpinpin.txt …
Get-Item .\Perlimpinpin.txt

# … mais qui retourne un objet avec ses propriétés
$o = Get-Item .\Perlimpinpin.txt

# Affichage formaté (des propriétés essentielles)
$o

# Liste complète des propriétés de l'objet
Get-Item | Get-Member

# Afficher la valeur d'une propriété
$o.Length
```

Autre différence : elles ne sont pas sensibles à la casse

Cmdlet usuelles

UNIX	Rôle	CmdLet	Alias
echo		Write-Host	*Aucun*
ls		Get-ChildItem	gci
man		Get-Help	
cut	*Sélectionner*	Select-Object	select
grep	*Filtrer*	Where-Object	where
cat	*Afficher*	Get-Content	
wc	*Compter*	Measure-Object	

Exemples

```
# Affiche seulement le nom et la taille du fichier
Get-Item Perlimpinpin.txt | Select-Object Name,Length

# Affiche les fichiers de plus de 100 octets
Get-ChildItem | Where-Object Length -gt 100

# Affiche la taille du plus gros fichier
Get-ChildItem | Measure-Object -Maximum Length
```


Cmdlet usuelles

UNIX	Rôle	CmdLet	Alias
head		Select-Objet -First *n*	
tail		Select-Objet -Last *n*	
tr		$*str*.replace('match', 'replace')	

Cmdlet usuelles : réseau

UNIX	Rôle	CmdLet	Alias
ip a		Get-NetIPInterface	
ip a add		New-NetIPAddress	
ip route		Get-NetRoute	
ping		Test-Connection	
curl		Invoke-WebRequest	
host		Resolve-DnsName	

Cmdlet usuelles : fichiers

UNIX	Rôle	CmdLet	Alias
ls		Get-ChildItem	
		Get-Item	
touch		New-Item	
rm		Remove-Item	

Cmdlet usuelles : temps

UNIX	Rôle	CmdLet	Alias
`sleep`		`Start-Sleep`	
`exit`		`exit`	
`date`		`Get-Date`	
`time`	*Mesurer le temps d'exécution d'une commande*	`Measure-Command -Expression {}`	

Le man

```
# Aide succinte
Get-Help Get-ChildItem

# Aide complète
Get-Help Get-ChildItem -Full
```


Notions de programmation PowerShell

1. Créer un script
2. Initialiser une variable
3. Utiliser une variable
4. Faire une opération arithmétique

5. Lire la valeur de retour d'une commande
6. Récupérer la sortie d'une commande
7. Passer un paramètre

8. Agir suivant la valeur d'une variable
9. Faire une boucle
10. Boucler sur tous les éléments d'une liste
11. Boucler sur les éléments d'un fichier
12. Boucler sur les valeurs dans un intervalle

13. Gérer une exception

Créer un script 1

Pas de shebang

```
# Ceci est un commentaire
# Le script commence son
# exécution ici ...
# et se termine quand il
# atteint la dernière ligne
```

```
# Exécuter le script :
./script.ps1
```

Initialiser une variable 2

```
# Un entier
$i = 0

# Une chaîne de caractères
$msg = "salut"

# Une collection (liste) d'objets
$liste = "Ethernet0","Ethernet1","Ethernet2"

# Un objet (ou une collection d'objets)
$result = Get-ChildItem
```


Utiliser une variable 3

```
# Entier ou chaîne de caractères
$msg = "salut"
Write-Host $msg
```

```
# (Collection d')objet(s)
$result = Get-ChildItem
Write-Host $result
$result
```

*Affichage **minimal***

*Affichage des propriétés **principales** de chaque objet*

Faire une opération arithmétique 4

```
$a = 1
$b = 1

$somme = $a + $b

Write-Host Somme de $a et $b : $somme
```


Lire la valeur de retour d'une commande 5

```
New-Item fichier

Remove-Item fichier
Write-Host $?

Remove-Item fichier
Write-Host $?
```

Le Cmdlet se termine et retourne :

- `True` si tout s'est bien déroulé
- `False` sinon

Récupérer la sortie d'une commande 6

Paramètre `-Proxy` pour indiquer l'URL de ce %$#&@! de proxy …

```
$output = Invoke-WebRequest http://api.ipify.org
Write-Host $output
```


Passer un paramètre 7

On lance un script (nommé `ifconf`) avec un paramètre (ici `203.0.113.42`)

```
./ifconf 203.0.113.42
```

Le paramètre se retrouve dans la variable spéciale `$Args`

```
Write-Host "Conf de l'adresse : $Args[0]"
New-NetIPAddress -InterfaceIndex 12 -IPAddress $Args[0]
```

```
./ifconf.sh 203.0.113.42
Conf de l'adresse : 203.0.113.42
```

*Possibilité de passer un paramètre **par nom** …*

Agir suivant la valeur d'une variable 8

Peut être simplifié en :
`if (! $i)`

```
if ($i -eq 0)
{
   # actions à réaliser
   # si $i est égal à 0
}
else
{
   # actions à réaliser
   # si $i n'est pas égal à 0
}
```


Faire une boucle 9

```
while (1)
{
   # actions à répéter en boucle
}
```

Boucler sur tous les éléments d'une liste 10

```powershell
# Renvoie une collection (liste) d'objets
$liste = Get-ChildItem

# Pour chaque élément de la liste
foreach ($elem in $liste)
{
   Write-Host "$elem"
}
```


Lire un fichier ligne par ligne 11

- Il suffit de combiner deux des notions précédentes :

```powershell
# (6) Récupérer la sortie d'une commande
$liste = Get-Content fichier

# (10) Boucler sur les éléments d'une liste
foreach ($e in $liste)
{
   Write-Host  "La valeur de e est : $e"
}
```


Boucler sur les valeurs dans un intervalle 12

```powershell
foreach ($i in 10..20)
{
   Write-Host "Valeur de i : $i"
}
```


Capturer une exception 13

```powershell
$ErrorActionPreference = "Stop"

try
{
   Remove-Item poudre-de-perlimpinpin.txt
}
catch
{
   Write-Host "Poudre de perlimpinpin toujours là !"
}
```