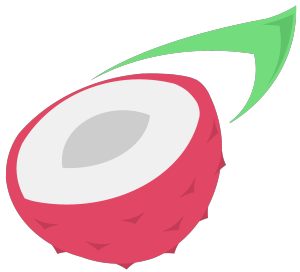

Litchi Pi

Scripting shell

Powershell

27 Novembre 2025

Introduction

Powershell, créé par Microsoft en 2006 (17 ans après le bash)

Open-source depuis 2016, basé sur le framework **.NET**

Windows CLI `cmd.exe` était trop limité, ne permettait pas d'automatiser tout le système

Les tentatives pour améliorer cela étaient des *désastres* de sécurité

Powershell a pour but de wrapper des API et interfaces existantes dans des **cmdlets** simples

Elles ont toutes un nom explicite, de la forme `Verbe-Objet <argument>` (ex: `Get-Help`)

Conçu pour du **script uniquement**, avec une pseudo-compatibilité *POSIX*

Timothée Cercueil - CC BY-NC license - Litchi Pi 2025

Script **orienté objet**, fonctionne comme un language de programmation

Chaque commande retourne un objet, pouvant être transmis par `|` ou affiché

Les objets ont des attributs associés, permet d'*éviter le parsing*

```
Get-Process | Where-Object { $_.VM > 50000 }
```

Les objets plus complexes (listes, tableaux) sont formattés selon un standard de Powershell

```
Get-Process | Sort-Object -Property CPU -Descending
NPM(K)   PM(K)   WS(K)    VM(M) CPU(s)     Id ProcessName StartTime
------   -----   -----    ----- ------     -- ----------- ---------
   143 239540 259384 2366162 22.73  12720 pwsh        12/5/2022 3:21:51 PM
   114  61776 104588 2366127 11.45  18336 pwsh        12/5/2022 7:30:53 AM
   156  77924  82060 2366185 10.47  18812 pwsh        12/5/2022 7:30:52 AM
    85  48216 115192 2366074 1.14   24428 pwsh        12/8/2022 9:14:15 AM
```

Timothée Cercueil - CC BY-NC license - Litchi Pi 2025

Syntaxe

```
$MY_VAR_1 = "toto"        # Stocke un texte
$MY_VAR_2 = Get-Process   # Stocke un objet
```

```
if (<condition>) {
  # ...
}
elseif (<condition>) {
  # ...
}
else {
  # ...
}
```

```
$letterArray = 'a','b','c','d'
foreach ($letter in $letterArray) {
  Write-Host $letter
}
```

Séparés par “;” tout comme *bash*

`$MY_VAR="titi"; $MY_OTHER_VAR="toto"` est une commande valide

Timothée Cercueil - CC BY-NC license - Litchi Pi 2025

Lever une erreur

Avec *throw*

```
throw "Il y a une erreur"
```

Récupérer des arguments nommés

Avec *param*

```
param ($servername, $envname)
```

Timothée Cercueil - CC BY-NC license - Litchi Pi 2025

L'environment

L’environment de la console est encore disponible dans powershell:

- **PATH** pour les binaires exécutables
- **HOME** pour le dossier personnel

etc ...

On peut donc utiliser les outils `cd`, `ls`, `rm`, etc ... dans Powershell

Cependant, les variables d’environment ne sont **pas des variables Powershell**

On peut y accéder avec `$Env:<variable name>`

Timothée Cercueil - CC BY-NC license - Litchi Pi 2025

Des variables spéciales sont définies dans Powershell:

- `$_` ou `$PSITEM` permet d'accéder au dernier objet du *pipeline*
- `$?` retourne *TRUE* si la dernière commande a réussi, sinon *FALSE*
- `$PSCOMMANDPATH` retourne le chemin vers le script exécuté actuellement

et beaucoup d'autres visibles *ici*

Timothée Cercueil - CC BY-NC license - Litchi Pi 2025

Cmdlets

Les cmdlets sont des commandes disponibles dans powershell qui ne sont **pas liés à un logiciel**

Ce sont des **abstractions** vers des systèmes existants:

- Filesystem
- Windows Registry
- Environment variables
- Liste de certificats
- Powershell runtime
- D'autres API de Windows

Les cmdlets, et leurs arguments sont **très verbeux**

Fait exprès pour avec un script *intelligible*, plus simple à lire.

Vous pourrez trouver *ici* de l'aide sur les cmdlets
(la barre latérale gauche contient la liste des cmdlets)

Timothée Cercueil - CC BY-NC license - Litchi Pi 2025

Gestion des erreurs

Par défaut, aucune commande ne se stoppe en cas d'erreur

```
# Stoppe en cas d'erreur
Get-Content -Path "./my_file" -ErrorAction Stop
```

Powershell permet d'utiliser le `try catch finally`

```
try {
  # Commande
}
catch {
  # Gestion de l'erreur
}
finally {
  # Commande à exécuter en cas d'erreur, ou non
}
```

Timothée Cercueil - CC BY-NC license - Litchi Pi 2025

Pokédex des cmdlets utiles

```
Get-Process | Where-Object {
  $MEMORY_KB = $_.VM / 1024
  $MEMORY_KB > 50000
}
```

```
Get-Process | ForEach-Object {
  $NAME = $_.Split(" ")[2]
  $DATE = $_.Split(" ")[3]
  "$NAME and the date is $DATE"
}
```

```
Get-Process | Sort-Object -Property CPU -Descending
```

```
Get-Content -Path "./my_file" | Out-File -FilePath "./my_file_copied"
```

```
Get-Content -Path "./my_file"
  | Select-String "^[0-9]{4}"
  | ForEach-Object {
    $YEAR = $_.Matches.Groups[1].Value
    "The year is $YEAR"
  }
```

Timothée Cercueil - CC BY-NC license - Litchi Pi 2025

TP

Analyse de logs