Quelques commandes Powershell/cmd

Polytech Marseille

Simon Vilmin
simon.vilmin@univ-amu.fr

2024 - 2025

Contexte

cmd (*invite de commande*) est un *interpréteur de commandes* DOS.

Powershell est un programme qui propose un *interpréteur de commandes* et *son langage de script* associé

- permet d'automatiser des tâches pour le système
- permet l'exploration du système de fichier

Remarque :

- Powershell repose sur de l'*orienté-objet*
- Powershell est le successeur de cmd
- on peut faire en Powershell tout ce qu'on fait en cmd

Commandslets et alias

Syntaxe : tous les noms de commandes powershell ont la forme

> `prefixe-objet`

ces commandes sont dites *commandlets* ou *cmdlets*

- exemples de préfixes : `get`, `set`, `add`, `clear`, `import`, `export`, `new`, `write`
- exemples d'« objet » : `command`, `item`, `content`, ...

Important : beaucoup de ces commandes ont un *alias* qui permettent d'utiliser des noms plus classiques : `cd`, `ls`, `help`, `mkdir`, ...

Astuce : la `doc` est des myriades de mini-guides sur le web.

Exemple : l'aide !

Important : on peut quasi-tout comprendre avec l'*aide* des commandes

Syntaxe : aide sur une commande

```
get-help <commande>

alias : help, man
```

Syntaxe : liste des commandes disponibles

```
get-command

alias : gcm
```

Syntaxe : correspondance entre commandes et alias

```
get-alias -name <alias> # commande depuis un alias
get-alias -definition <commande> # alias d'une commande
```

Comprendre l'aide

Syntaxe :

```
commande [[-param1] <valeur>] <valeur> [-param2 <valeur>]
  [-param3 {<valeur1 | valeur2 | ... | valeurn>}]
```

les `[ ]` indiquent le *facultatif* :

- `[[-param1] <valeur>]` : pas obligatoire, et si on le précise, `-param` pas obligatoire non plus (*positionnel*)
- `<valeur>` : obligatoire
- `[-param3 <valeur>]` : pas obligatoire, mais si on le met, on doit écrire `-param3`
- `[-param <valeur1 | valeur2 | ... | valeurn>]` : l'accolade signifie que la valeur du paramètre est un sous-ensemble de `valeur1`, ..., `valeurn`

Se déplacer dans l'arborescence

Syntaxe : changer de répertoire

```
set-location <chemin>
set-location .. # remonter au parent

alias : cd, chdir, sl
```

Syntaxe : lister les éléments d'un répertoire

```
get-childitem

alias : dir, ls, gci
```

Remarque : pour `get-childitem`

- `-force` permet de lister les éléments cachés
- `-recurse` permet de lister récursivement tous les sous-dossiers

Création de dossiers et fichiers

Syntaxe : créer un dossier ou un fichier

```
new-item -path <chemin> -itemtype directory # dossier
new-item -path <chemin+nomfichier> -itemtype file # fichier


alias : ni
```

Exemple

```
ni -path "C:\Users\Documents\pouet" -itemtype directory
ni -path "C:\Users\Documents\pouet\pouet.txt" -itemtype file
```

Syntaxe : renommer un dossier ou un fichier

```
rename-item <anciennom> <nouveaunom>


alias : ren, rni
```

Affichage d'un fichier

Syntaxe : afficher le contenu d'un fichier

```
get-content <fichier>

alias : cat, gc, type
```

Copie et déplacement

Syntaxe : copier un fichier ou un dossier

```
copy-item <nomelement> <nouvellelocation> [-recurse] [-force]

alias : cp, cpi, copy
```

Syntaxe : déplacer un fichier ou un dossier

```
move-item <nomelement> <nouvellelocation> [-recurse] [-force]

alias : mv, mi, move
```

Attention : pour copier/déplacer tout le contenu d'un *dossier* (récursivement) il faut utiliser `-recurse` !

Suppression

Syntaxe : supprimer un dossier ou un fichier

```
remove-item <element> [-recurse] [-force]

alias : rd, ri, rm, rmdir, erase
```

Remarque :

- même plus trop de surprise !
- toutes ces commandes ont bien sur d'*autres options*

Redirection pour modifier un fichier

On peut réutiliser le *résultat* d'une commande avec des *opérateurs de redirection*

- `commande1 | commande2` : le résultat de `commande1` sera l'entrée de `commande2`
- `commande > <fichier>` : résultat de `commande2` écrit dans `fichier` à la place de ce qu'il y avait avant (écrasement)
- `commande >> <fichier>` : résultat de `commande2` écrit dans `fichier` à la suite des données précédentes (*append*)

Syntaxe : pour écrire le résultat d'une commande dans un fichier

```
commande | out-file <fichier> # permet du formattage detaille
commande > <fichier> # redirige tout sans format special
```

Idée de filtrage

Idée :

- Powershell orienté objet $\longrightarrow$ les commandes renvoient une *liste d'objets*
- le | *transfère le résultat* d'une commande

on peut faire du *filtrage* !

Idée de filtrage

Syntaxe : filtrer les objets

```
commande | where-object -options <de filtrage>

alias : ?, where
```

Syntaxe : grouper les objets

```
commande | group-object -options <de groupement>

alias : group
```

Exemple :

```
dir  # liste les elements
  | where -property length -GE 10000 # >= 10000 carac
  | group -property extension # groupe par extension
```

Processus

Syntaxe : avoir la liste des processus actifs

```
get-process

alias : ps, gps
```

Syntaxe : arrêter un processus

```
stop-process <PID> [-force] [-confirm] # via l'identifiant (PID)
stop-process -name <nom> [-force] [-confirm] # via le nom
```

Remarque : `-confirm` demande confirmation de l'arrêt du processus

On est des hackèrses

Dans powershell :

- `systeminfo` : donne les informations du système (hardware inclus)
- `cmd` : lance une instance de `cmd` dans Powershell

Dans cmd :

- `choice` : propose un choix
- `color` : change les couleurs
- `prompt` : change le prompt