Introduction to PowerShell for Unix People

Introduction to PowerShell for Unix People

PowerShell.org

This project can be followed at:
https://www.penflip.com/powershellorg/a-unix-persons-guide-to-powershell

1. About

Principal author: Matt Penny

This e-book is intended as a 'Quick Start' guide to PowerShell for people who already know Bash or one of the other Unix shells.

The book has 3 elements:

- an introductory chapter which covers some PowerShell concepts
- a summary list of PowerShell equivalents of Unix commands in one e-book chapter
- a detailed discussion of Powershell equivalents of Unix commands, organised in the alphabetical order of the unix command

---

Visit *www.penflip.com/powershellorg* to check for newer editions of this e-book.

PowerShell.org eBooks are works-in-progress, and many are curated by members of the community. We encourage you to check back for new editions at least twice a year, by visiting *www.penflip.com/powershellorg*.

You can download this book in a number of different formats (including EPUB, PDF, Microsoft Word and Plain Text) by clicking **Download** on the right side of the page.

PDF Users: Penflip's PDF export often doesn't include the entire ebook content. We've reported this problem to them; in the meantime, please consider using a different format, such as EPUB, when you're downloading the book.

You may register to make corrections, contributions, and other changes to the text - we welcome your contributions! However, we recommend you check out our contributor tips and notes before jumping in.

You may also subscribe to our monthly e-mail TechLetter for notifications of updated e-book editions. Visit PowerShell.org for more information on the newsletter.

2. Introduction to PowerShell for Unix people

The point of this section is to outline a few areas which I think *nix people should pay particular attention to when learning Powershell.

Resources for learning PowerShell

A *full* introduction to PowerShell is beyond the scope of this e-book. My recommendations for an end-to-end view of PowerShell are:

- Learn Windows PowerShell in a Month of Lunches - Written by powershell.org's Don Jones and Jeffery Hicks, I would guess that this is the book that most people have used to learn Powershell. It's 'the Llama book' of Powershell.

- Microsoft Virtual Academy's 'Getting Started with PowerShell' and 'Advanced Tools & Scripting with PowerShell' Jump Start courses - these are recordings of day long webcasts, and are both free.

unix-like aliases

PowerShell is a friendly environment for Unix people to work in. Many of the concepts are similar, and the PowerShell team have built in a number of Powershell aliases that look like unix commands. So, you can, for example type:

```
ls
```

....and get this:

```
    Directory: C:\temp
Mode                LastWriteTime     Length Name
----                -------------     ------ ----
-a---        22/02/2015     16:51      25773 all_the_details.md
-a---        20/02/2015     07:31       3390 commands-summary.md
```

These can be quite useful when you're switching between shells, although I found that it can be irritating when the 'muscle-memory' kicks in and you find yourself typing `ls -ltr` in PowerShell and get an error. The 'ls' is just an alias for the PowerShell `get-childitem` and the Powershell command doesn't understand `-ltr`[1].

the pipeline

The PowerShell pipeline is much the same as the Bash shell pipeline. The output of one command is piped to another one with the '|' symbol.

The big difference between piping in the two shells is that in the unix shells you are piping *text*, whereas in PowerShell you are piping *objects*.

This sounds like it's going to be a big deal, but it's not really.

In practice, if you wanted to get a list of process names, in bash you might do this:

```
ps -ef | cut -c 49-70
```

…whereas In PowerShell you would do this:

```
get-process | select ProcessName
```

In Bash you are working with characters, or tab-delimited fields. In PowerShell you work with field names, which are known as 'properties'.

get-help, get-command, get-member

get-member

When you run a PowerShell command, such as `get-history` only a subset of the `get-history` output is returned to the screen.

In the case of `get-history`, by default two properties are shown - 'Id' and 'Commandline'…

```
$ get-history

  Id CommandLine
  -- -----------
   1 dir -recurse c:\temp
```

…but get-history has 4 other properties which you might or might not be interested in:

```
$ get-history | select *

Id                 : 1
CommandLine        : dir -recurse c:\temp
ExecutionStatus    : Completed
StartExecutionTime : 06/05/2015 13:46:56
EndExecutionTime   : 06/05/2015 13:47:07
```

The disparity between what is shown and what is available is even greater for more complex entities like 'process'. By default `get-process` shows 8 columns, but there are actually over 50 properties (as well as 20 or so methods) available.

The full range of what you can return from a PowerShell command is given by the `get-member` command[2].

To run `get-member`, you pipe the output of the command you're interested in to it, for example:

```
get-process | get-member
```

….or, more typically:

```
get-process | gm
```

`get-member` is one of the 'trinity' of 'help'-ful commands:

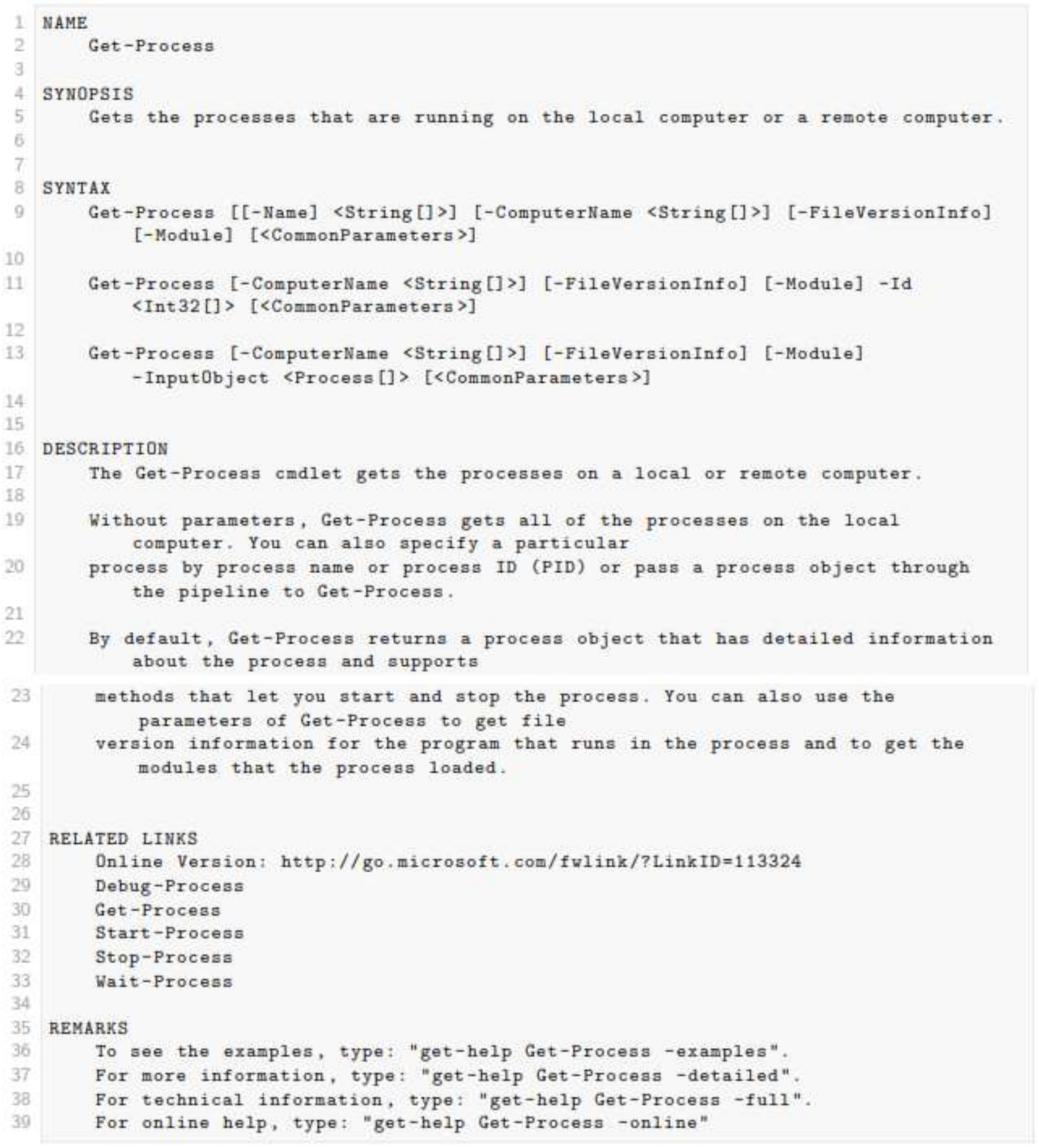
- get-member

- get-help
- get-command

get-help

`get-help` is similar to the Unix `man`[3].

So if you type `get-help get-process`, you'll get this:



```
NAME
    Get-Process

SYNOPSIS
    Gets the processes that are running on the local computer or a remote computer.


SYNTAX
    Get-Process [[-Name] <String[]>] [-ComputerName <String[]>] [-FileVersionInfo]
        [-Module] [<CommonParameters>]

    Get-Process [-ComputerName <String[]>] [-FileVersionInfo] [-Module] -Id
        <Int32[]> [<CommonParameters>]

    Get-Process [-ComputerName <String[]>] [-FileVersionInfo] [-Module]
        -InputObject <Process[]> [<CommonParameters>]


DESCRIPTION
    The Get-Process cmdlet gets the processes on a local or remote computer.

    Without parameters, Get-Process gets all of the processes on the local
        computer. You can also specify a particular
    process by process name or process ID (PID) or pass a process object through
        the pipeline to Get-Process.

    By default, Get-Process returns a process object that has detailed information
        about the process and supports
    methods that let you start and stop the process. You can also use the
        parameters of Get-Process to get file
    version information for the program that runs in the process and to get the
        modules that the process loaded.


RELATED LINKS
    Online Version: http://go.microsoft.com/fwlink/?LinkID=113324
    Debug-Process
    Get-Process
    Start-Process
    Stop-Process
    Wait-Process

REMARKS
    To see the examples, type: "get-help Get-Process -examples".
    For more information, type: "get-help Get-Process -detailed".
    For technical information, type: "get-help Get-Process -full".
    For online help, type: "get-help Get-Process -online"
```

There are a couple of wrinkles which actually make the PowerShell 'help' even more *help*-ful.

- you get basic help by typing `get-help`, more help by typing `get-help -full` and…probably the best bit as far as I'm concerned…you can cut to the chase by typing `get-help -examples`
- there are lots of '`about_`' pages. These cover concepts, new features (in for example `about_Windows_Powershell_5.0`) and subjects which dont just relate to one particular command. You can see a full list of the 'about' topics by typing `get-help about`
- get-help works like `man -k` or `apropos`. If you're not sure of the command you want to see help on, just type `help process` and you'll see a list of all the help topics that talk about processes. If there was only one it would just show you that topic
- *Comment-based help*. When you write your own commands you can (and should!) use the comment-based help functionality. You follow a loose template for writing a comment header block, and then this becomes part of the get-help subsystem. It's good.

get-command

If you don't want to go through the help system, and you're not sure what command you need, you can use `get-command`.

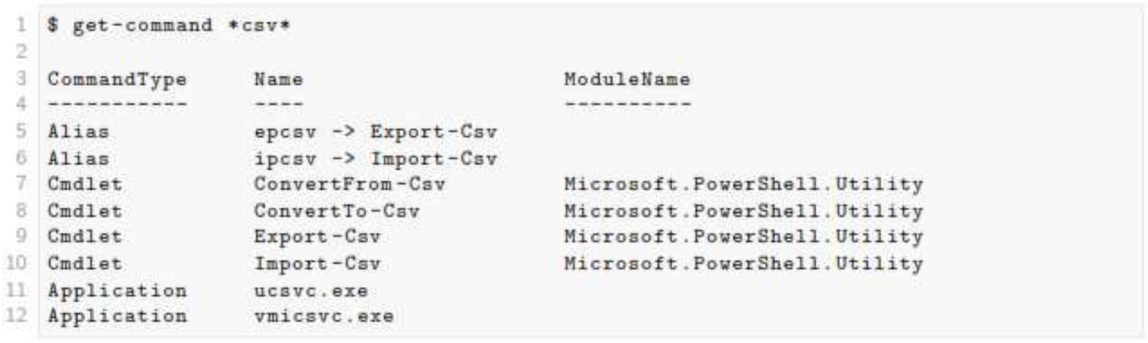
I use this most often with wild-cards either to explore what's available or to check on spelling.

For example, I tend to need to look up the spelling of `ConvertTo-Csv` on a fairly regular basis. PowerShell commands have a very good, very intuitive naming convention of a verb followed by a noun (for example, `get-process`, `invoke-webrequest`), but I'm never quite sure where 'to' and 'from' go for the conversion commands.

To quickly look it up I can type:

```
get-command *csv*
```

… which returns:



```
$ get-command *csv*

CommandType     Name                     ModuleName
-----------     ----                     ----------
Alias           epcsv -> Export-Csv
Alias           ipcsv -> Import-Csv
Cmdlet          ConvertFrom-Csv          Microsoft.PowerShell.Utility
Cmdlet          ConvertTo-Csv            Microsoft.PowerShell.Utility
Cmdlet          Export-Csv               Microsoft.PowerShell.Utility
Cmdlet          Import-Csv               Microsoft.PowerShell.Utility
Application     ucsvc.exe
Application     vmicsvc.exe
```

Functions

Typically PowerShell coding is done in the form of *functions*[4]. What you do to code and write a function is this:

Create a function in a plain text .ps1 file[5]

```
gvim say-HelloWorld.ps1
```

say-helloworld.png

…then source the function when they need it

```
$ . .\say-HelloWorld.ps1
```

…then run it

```
$ say-helloworld
Hello, World
```

Often people autoload their functions in their `$profile` or other startup script, as follows:

```
write-verbose "About to load functions"
foreach ($FUNC in $(dir $FUNCTION_DIR\*.ps1))
{
  write-verbose "Loading $FUNC.... "
  . $FUNC.FullName
}
```

## Footnotes

[1] If you wanted the equivalent of `ls -ltr` you would use `gci | sort lastwritetime`. 'gci' is an alias for 'get-childitem', and I think, 'sort' is an alias for 'sort-object'.

[2] Another way of returning all of the properties of an object is to use 'select *'…so in this case you could type `get-process | select *`

[3] There is actually a built-in alias `man` which tranlates to `get-help`, so you can just type `man` if you're pining for Unix.

[4] See the following for more detail on writing functions rather than scripts: http://blogs.technet.com/b/heyscriptingguy/archive/2011/06/26/don-t-write-scripts-write-powershell-functions.aspx

[5] I'm using 'gvim' here, but notepad would work just as well. PowerShell has a free 'scripting environment' called *PowerShell ISE*, but you don't have to use it if you dont want to.

3. commands summary

alias (set aliases)

```
set-alias
```

More

alias (show aliases)

```
get-alias
```

More

apropos

```
get-help
```

More

basename

```
dir | select name
```

More

cal

No equivalent, but see the script at http://www.vistax64.com/powershell/17834-unix-cal-command.html</a>

cd

```
cd
```

More

clear

```
clear-host
```

More

date

```
get-date
```

More

date -s

```
set-date
```

More

df -k

```
Get-WMIObject Win32_LogicalDisk | ft -a
```

More

dirname

```
dir | select directory
```

More

du

No equivalent, but see the link

echo

```
write-output
```

More

echo -n

```
write-host -nonewline
```

More

| egrep -i sql

```
 | where {[Regex]::Ismatch($\_.name.tolower(), "sql") }
```

More

egrep -i

```
select-string
```

More

egrep

```
select-string  -casesensitive
```

More

egrep -v

```
select-string -notmatch
```

More

env

```
Get-ChildItem Env: | fl
```

or

get-variable

More

errpt

```
get-eventlog
```

More

export PS1=”$ “

```
function prompt {"$ " }
```

More

find

```
dir  *whatever* -recurse
```

More

for (start, stop, step)

```
for ($i = 1; $i -le 5; $i++) {whatever}
```

More

head

```
gc file.txt | select-object -first 10
```

More

history

```
get-history
```

More

history | egrep -i ls

```
history | select commandline | where commandline -like '*ls*' | fl
```

More

hostname

```
hostname
```

More

if-then-else

```
if  ( condition ) { do-this } elseif { do-that } else {do-theother}
```

More

if [-f “$FileName”]

```
if (test-path $FileName)
```

More

kill

```
stop-process
```

More

less

```
more
```

More

locate

```
no equivalent but see link
```

More

ls

```
get-childitem OR gci OR dir OR ls
```

More

ls -a

```
ls -force
```

More

ls -ltr

```
dir c:\ | sort-object -property lastwritetime
```

More

lsusb

```
gwmi Win32_USBControllerDevice
```

More

mailx

```
send-mailmessage
```

More

man

```
get-help
```

More

more

```
more
```

More

mv

```
rename-item
```

More

pg

```
more
```

More

ps -ef

```
get-process
```

More

ps -ef | grep oracle

```
get-process oracle
```

More

pwd

```
get-location
```

More

read

```
read-host
```

More

rm

```
remove-item
```

More

script

```
start-transcript
```

More

sleep

```
start-sleep
```

More

sort

```
sort-object
```

More

sort -uniq

```
get-unique
```

More

tail

```
gc file.txt | select-object -last 10
```

More

tail -f

```
gc -tail 10 -wait file.txt
```

More

time

```
measure-command
```

More

touch - create an empty file

```
set-content -Path ./file.txt -Value $null
```

More

touch - update the modified date

```
set-itemproperty -path ./file.txt -name LastWriteTime -value $(get-date)
```

More

wc -l

```
gc ./file.txt | measure-object | select count
```

More

whoami

```
[Security.Principal.WindowsIdentity]::GetCurrent() | select name
```

More

whence or type

```
No direct equivalent, but see link
```

More

unalias

```
remove-item -path alias:aliasname
```

More

uname -m

```
Get-WmiObject -Class Win32_ComputerSystem | select manufacturer, model
```

More

uptime

```
get-wmiobject -class win32_operatingsystem | select LastBootUpTime`
```

More

\ (line continuation)

```
` (a backtick)
```

More

4. commands detail - a

alias (list all the aliases)

The Powershell equivalent of typing `alias` at the bash prompt is:

```
get-alias
```

alias (set an alias)

At it's simplest, the powershell equivalent of the unix 'alias' when it's usedto set an alias is 'set-alias'

```
set-alias ss select-string
```

However, there's a slight wrinkle....

In unix, you can do this

```
alias bdump="cd /u01/app/oracle/admin/$ORACLE_SID/bdump/"
```

If you try doing this in Powershell, it doesn't work so well. If you do this:

```
set-alias cdtemp "cd c:\temp"
cdtemp
```

...then you get this error:

```
cdtemp : The term 'cd c:\temp' is not recognized as the name of a cmdlet, function,
    script file, or operable program. Check the spelling of the name, or if a path
    was included, verify that the path is correct and try again.
At line:1 char:1
+ cdtemp
+ ------
    + CategoryInfo          : ObjectNotFound: (cd c:\temp:String) [],
      CommandNotFoundException
    + FullyQualifiedErrorId : CommandNotFoundException
```

A way around this is to create a function instead:

```
remove-item -path alias:cdtemp
function cdtemp {cd c:\temp}
```

You can then create an alias for the function:

```
set-alias cdt cdtemp
```

apropos

apropos is one of my favourite bash commands, not so much for what it does...but because I like the word 'apropos'.