Windows Operating System Environment

Computadors – Grau en Ciència i Enginyeria de Dades – 2019-2020 Q2

Facultat d'Informàtica de Barcelona – Departament d'Arquitectura de Computadors

In this laboratory session, we will learn to access and use the Windows Operating System, using a desktop computer. We will browse over the different tools provided by the system, to obtain information from the OS, execute commands, and build small scripts to do simple tasks. We will be using Windows 10, and please take into account that some of the features we will show may be partially available only on older Windows releases.

Booting your system with Windows10

Start your desktop machine, and wait for the boot loader to show you the boot options. You will see:

Suse Linux   Windows10   ... among other options.

Please, select "Windows10", and observe how the machine initiates a full "reboot", and it brings you the login page of Windows10.

Enter your username and password. Use your credentials as you do in the UPC Intranet, or in the RACÓ.

During the log-in process, the system will issue a message regarding where is the proper place to save your files. Usually, the disk drive labeled as F: will contain your personal files. This is the same area that you see from your Linux account, as "dades".

Browsing around the environment

Desktop view

On the initial desktop of W10 you will see the menu bar at the bottom of the screen, and the Recycle Bin at the top-left corner, among other icons representing application (Chrome browser...)

In the menu bar at the bottom, click on the left-most icon, the Windows Menu. This menu can also be opened at any time by using the Windows key.

PowerShell

In the menu that appears, use the search box to search for "Powershell", and open it (should be "Windows Powershell", from the menu after searching for "Powershell"). You will see a command line shell. This is the current shell environment that Windows offers. It is also available on older Windows versions, although there it may offer a reduce set of the current functionality.

PowerShell is also available in Linux and MacOS environments, see https://github.com/PowerShell/PowerShell.

The shell displays a welcome message, similar to:

```
Windows PowerShell

Copyright (C) Microsoft Corporation. All rights reserved.
```

And it shows a command prompt, waiting for commands, usually something like:

```
PS C:\Users\xavim>
```

From now on, we will use the simplified command prompt (>) to indicate that you should provide some input command, to see or report its output.

Basic command examples

As as example, try the following commands and compare the output you get with the one proposed here:

```
> whoami

vpcxavim7\xavim

> dir

            Directory C:\Users\xavim

Mode         LastWriteTime       Length   Name

----         -------------       ------   ----

d----     2/14/2018  10:10 AM             Contacts

d-r--     2/13/2018   1:38 PM             Desktop

   ...

-a---     2/11/2018   9:43 AM       170   env.bat

-a---     2/11/2018  10:40 AM       185   env.ps1

-a---     1/31/2018   4:16 PM      1050   Makefile

-a---     2/01/2018   3:55 PM      3290   program.c

-a---     2/01/2018   3:57 PM     73216   program.exe

-a---     2/01/2018   3:56 PM      1808   program.obj
```

Commands and command options and parameters

```
> dir -r  # recurses inside the directories

...

> dir -Recurse    # equivalent, better for self-documentation

...

> dir -Path \     # lists the root directory on the current unit (C:)

      Directory: C:\

...

d----     7/14/2009    5:20 AM     PerfLogs

d-r--     2/13/2018   10:06 AM     Program Files

...

d----     2/13/2018    1:30 PM     Windows
```

For the "dir" comand, directories are indicated by the "d" attribute in Mode. Files are indicated with a "-" in the same position of Mode. The definition of these attributes is the following:

```
d----    directory / file
-a---    not archived / archived
--r--    read-only / read-write
---h-    hidden / visible
----s    system / user
```

When a file has the "a" attribute, it means "this file has not been archived". Hidden files are not usually listed by the dir command. They can be listed with the -Force command option. Windows files have more attributes that will be explained across the course, if needed.

Try:

```
> dir -Force
...     # you may see hidden files here
```

and

```
> dir -Force \
...     # you may see the swap and hibernation files here
```

In Windows, both filenames and command line options are taken as case insensitive.

Some more example commands:

```
> # this is a comment for Powershell, useful in scripts
      ( --- no output here --- )


> "These are four equivalent commands"
These are four equivalent commands
```

(You can use the "up" arrow key to go to the previous commands you have typed)

```
> 'These are four equivalent commands'
These are four equivalent commands


> echo "These are four equivalent commands"
These are four equivalent commands
```

```
> echo 'These are four equivalent commands'

These are four equivalent commands
```

Use the ";" character to separate commands on the same line (Linux shells also allow this command separation on the same line). Probably the most simple combined command is this:

```
> "This one"; "shows two lines of text"

This one

shows two lines of text
```

Getting help

```
> help echo    # shows a manual page similar to the Linux ones

NAME

  Write-Output


SYNOPSYS

  Sends the specified objects to the next command in the pipeline(*). If

  the command is the last command in the pipeline, the objects are

  displayed in the console.


SYNTAX

  Write-Output [-InputObject] <PSObject[]> [<CommonParameters>]


DESCRIPTION

    ...
```

(*) "pipeline" is explained later.

Why do you think we are getting the "Write-Output" command help, instead of the "echo" help?

Answer: echo is an alias(**) for Write-Output

(**) aliases are explained later

```
> Write-Output "These are //five// equivalent commands"

These are //five// equivalent commands
```

A specific command can print its own help. The current syntax for requesting the help is "-?". Arguments starting with a minus sign (or dash) are options for the commands. Formerly, and for older commands, they can accept the string "/?", instead of "-?". To determine if a command accepts the current syntax, or the old syntax, please, just try both. For example:

```
> Write-Output -?
```

Observe that using this way, the help is displayed without stopping on a page by page basis. What is the reason for the help not being displayed on a page by page basis?

Answer: help Write-Output uses the "more" command to manage pages, while the commands themselves do not do that.

```
> Write-Output -? | more  # Congratulations!!!
                           # this is your first command pipeline
```

Pipelines

The text written by Write-Output while showing the help, is read by the more command, and this one interacts with the user to display it on a page by page basis.

When the more command displays " -- More -- " at the bottom of the page, you can use a few keys to manage the display of the text:

```
<space>    jumps to the next page
<return>   steps to the next line
q          quits
h          shows a small help
```

Aliases

Windows 10 supports aliases in the form of a translation for a single name, to a another PowerShell command.

```
> Set-Alias grep Select-String
```

Aliases can be removed with the command “Remove-Item”

```
> Remove-Item alias:grep
```

Exercise 1

(Edit a new “answers.txt” file in order to answer the next exercises)

Using “alias”, list the translation of the following commands

ls:

dir:

cd:

copy:

history:

start:

(write your answers in the “answers.txt” file)

Using PowerShell CmdLets

The internal commands of PowerShell are named as CmdLets. The aliases we have seen before are used to simplify the names of the CmdLets.

Exercise 2

Try the following CmdLets to get familiar with them, and explain briefly what they do:

```
> Get-Item  alias:ls

> Get-Item   "."

> Get-ChildItem   "."

> Get-ChildItem   "."   -Recurse   -include    "answers.txt"

> Select-String   history   answers.txt                          # behaves similar to Linux grep
```

(write your answers in the "answers.txt" file)

Environment variables

Try:

$env:Path

```
;c:\sc55\bin;c:\utils;C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;C:\WI
NDOWS\System32\WindowsPowerShell\v1.0\;C:\Program Files\Intel\WiFi\bin\;C:\Program
Files\Common Files\Intel\WirelessCommon\;C:\Users\xavie\AppData\Local\Microsoft\
WindowsApps;
```

This shows the list of search paths that the shell follows to find your commands. Components are separated by semicolons (;). Remind: In Linux PATH components are separated by the colon character (':').

Exercise 3

Prepend the path that you obtain from the string "$Home\bin" to your env:Path variable, so that you can execute commands from your own "bin" directory.

(hint: > $env:Path   =   ...)

(write the command line you use to the "answers.txt" file)

The Desktop environment

Get the description of the machine using the File Browser. Go to "Computer -> Properties" (it may be also "This PC -> Properties"), using the right mouse button. Be aware that in the following exercises you may find some issues related to access permissions.

Exercise 4

Get the following properties of your environment:

Windows edition:

Processor:

Installed memory (RAM):

System type:

Computer Name:

Workgroup: **(write the information you find in the "answers.txt file)**

Exercise 5

From the same system view window, check all environment variables, using "Advanced Systems Settings -> Advanced options tab -> Environment Variables". List the contents of these variables:

(from the user variables) TEMP

(from the system variables) ComSpec

(from the system variables) NUMBER_OF_PROCESSORS

(from the system variables) OS

(write the information you find in the "answers.txt file)

Exercise 6

Try the following two commands:

> cmd

(and then issue "exit" to finish the execution of the "cmd")

> start cmd

Explain the differences you observe.

(write your answer in the "answers.txt file)

Probably, you are already familiar with the command line of "cmd", just check that the view that you get with it (dir) in your home directory is equivalent to the view you had with PowerShell.

In PowerShell, execute:

> start -nonewwindow cmd

And try to give two or three consecutive commands to see the behaviour of PowerShell and Cmd sharing the same window.

Is there any command line option to be given to "start" so that the behaviour gets corrected?

(write your answer in the "answers.txt file)

PowerShell scripts

Math

PowerShell allows to compute expressions with several data types:

```
> ((46+63)/2)

54.5

> [math]::sqrt([float]63.0)

7.93725...

> [math]::Pow(3,2)

9
```

```
> ([math]::Pow(3,2)).GetType()

IsPublic IsSerial Name                                     BaseType

-------- -------- ----                                     --------

True     True     Double                                   System.ValueType

> ([float]::MaxValue)

3.402823E+38                  # Check these values with the ones

> ([float]::Epsilon)          # shown in slide 21 of the slide-set

1.401298E-45                  # 2. Representació de Dades
```

Computing on data with scripts

Get the file “FilesS2.zip” attached in Racó, and uncompress it. You can use the File Explorer browser in order to do it.

Inside “FilesS2.zip” you will find a data file (temp.csv) with minute temperatures recorded around the Barcelona area in the period Jan 17th, 2020 to Feb 15th, 2020, in Fahrenheit units:

temp.csv

```
month,day,second,temp
01,17,064542,46          #seconds represent HH:MM:SS,
01,17,065042,46          # without the : sign
01,17,065543,46
01,17,070042,46
....
```

This file follows the Comma-Separated Values (CSV) format. The first row contains the header, or the names of the different columns. The rest of the lines contain the values. In this case, they indicate a time stamp, consisting of month number, day of the month, hour-minute-second, and the temperature registered (in Fahrenheit). The CSV format is very commonly used to hold data.

A second file (`get-statistics.ps1`) is a PowerShell script that is able to process files with the same format, in order to obtain the Average temperature on a day by day basis:

Try the following command:

```
> .\get-statistics.ps1  temp.csv  “01,17”  # get average for Jan 17th

Average
-------
55.5288...     # average in Fahrenheit
13.0715...     # average in Celsius
```

If you observe that your operating system indicates that it cannot execute the PowerShell script due to a limitation on the execution policy, this is a security feature of Windows. You can temporarily enable the execution of scripts with the command:

```
> Set-ExecutionPolicy -ExecutionPolicy bypass -Scope Process
```

Exercise 7

Look at the script `get-statistics.ps1` and try to understand it.
(write your findings in the "answers.txt file)

Exercise 8

Now, change the `get-statistics.ps1` script in order to also display the **minimum** and **maximum** statistics, for the requested month/day.
(explain how do you find the way to calculate the minimum and maximum values of the temperatures, and the changes you do in the get-statistics.ps1 script, into the "answers.txt file)

Upload the Deliverable

Compress your "answers.txt" file and the new version of the `get-statistics.ps1` script into a "session2.zip" file. You can use the File Explorer, which provides the zipping functionality. Then, go to the RACÓ, and upload your "session2.zip" file, to the corresponding session slot.