ELTE INFORMATIKA

Computer Systems

10. Powershell

Review

- Computers, information, number representation, code writing, architecture, file systems
- Base commands, processes, regular expressions
- Variables, command substitution, arithmetical, logical expressions
- Script control structures, sed, awk
- Batch, WSH
- PS overview, PS variables, operations
- Basic Powershell commands, control structures

What comes today?

- PowerShell functions
- PowerShell language elements
- …

Functions in PowerShell

- function name(par) { function block }
  - You can place it to anywhere, but it can be used only after the definition!
  - result: return instruction
  - $lastexitcode variable (contains the result of the last executed external program)
  - Call: name 5 or name(5)
  - Several parameters:
    - name $x $y

```
function nfactor($n)
{
 $f=1
 for($i=2;$i -le $n;$i++) {$f*=$i}
 return $f
}
echo "N factor"
nfactor(5) or nfactor 5
```

Classical or parameter block

Classical parameter usage:

```
function global:Where-BigProcess(
    [int]    $meret = 200MB,
    [String] $property = “VirtualMemorySize”,
    [String] $formatString = “Total {2} = {1}”
){ body }
```

PARAMeter block usage

```
function global:Where-BigProcess {
    PARAM(
    [int]    $meret = 200MB,
    [String] $property = “VirtualMemorySize”,
    [String] $formatString = “Total {2} = {1}”
)
    body
}
```

Function parameters, arguments

- Typically they are given with the classical form:
  - function apple($name,$size) { …} or param block
- Default values, arguments:
  - $name=”zoli”, like in e.g.. C#
  - Mandatory parameters:
    - function apple([parameter(Mandatory=$true)] $name) { …}
    - Previously: apple($name=$(throw ”Give me a name!!”)) { …}
- Variable numbers of parameters
  - Similar to .NET, $args array

Giving function arguments

- Function apple($name,$price,$color) {…}
- Fitting of arguments by position:
  - Call: apple „jonatan” 150 ”red”;
- Fitting of arguments by name:
  - Call: apple –name golden –color yellow
    - We did not give the price so it is empty!
    - Short form: apple –n golden –c yellow
- We can mix the two different usage of arguments!
  - E:g.: apple –c green green 250
  - Advice: do not use the mixed form!

Modifying function, variable level (dot sourcing)

- You may declare a function within a function. An inner function may not be called directly!
  - Execute it with a dot: . Funct
    - The result of it that the inner functions also may be seen directly.
- Function local variable may not be seen from outside.
  - Execute it with a dot: . Fv
    - The result of it that the local function variables also may be seen directly!
- Be careful with it!!

Direct usage of functions

- This is an example of using level modification with .!
  - „Dot-Sourcing” operator
  - The same in Shell script!

Windows PowerShell ISE (x86)

File Edit View Tools Debug Add-ons Help

square.ps1

```
1  param($number=$(throw "Give me a number"))
2  # square function
3  function square($x)
4  {
5  return $x*$x
6  }
7  $n=square($number)
8  "The square of the original argument is {0}" -f $n
```

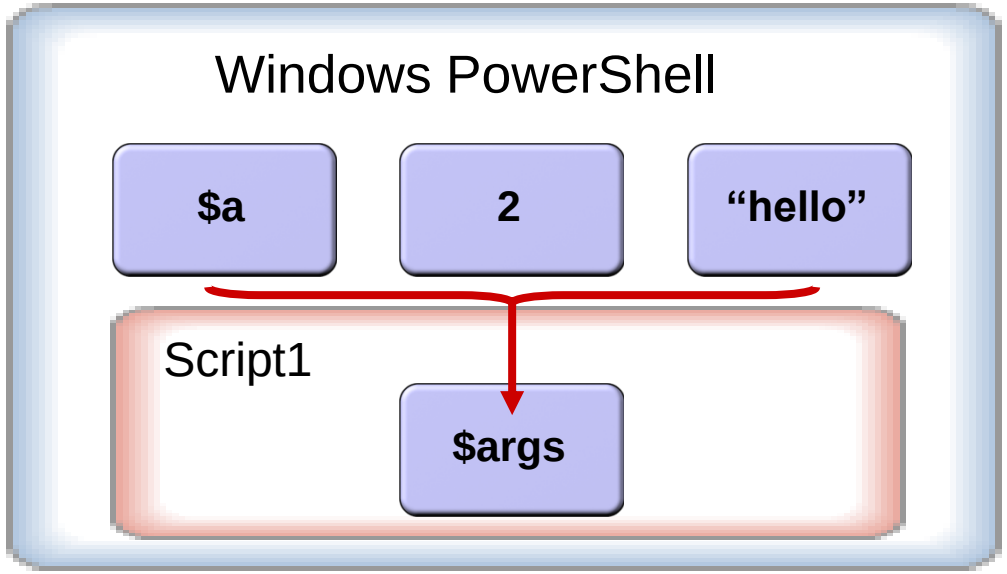
```
PS D:\home> . .\square.ps1 6
The square of the original argument is 36

PS D:\home> square 7
49

PS D:\home> |
```

Ln 23 Col 13 140%

Script arguments used as an array

```
# script file argtest.ps1
foreach( $i in $args ){“Parameters {0:D};” -f $i }
```

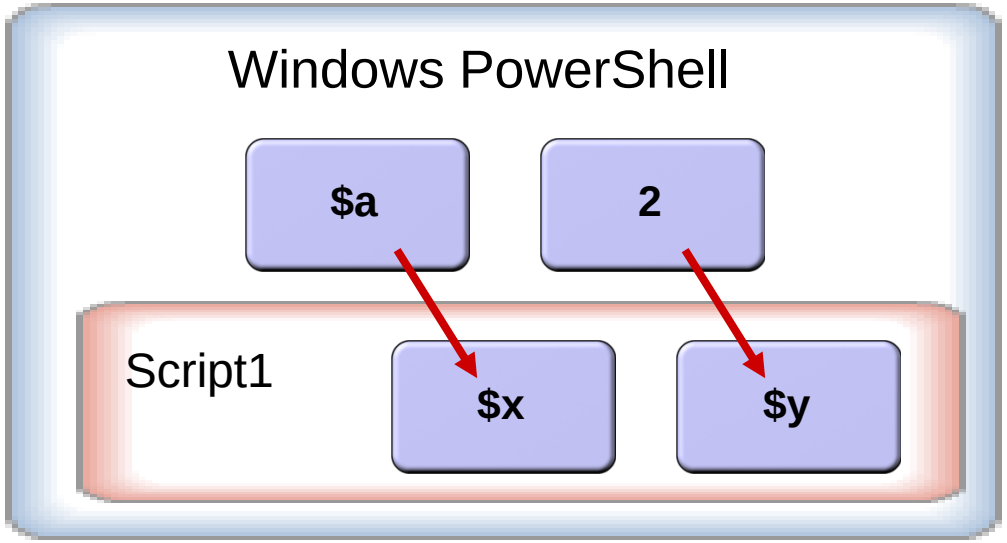
```
PS> $a = 10
PS> .\argtest.ps1 $a 2 “hello”
```



Named script parameters

```
# named parameters
param( $x, $y )
“The `$x={0}” -f $x
“The `$y={0}” -f $y
```

```
PS> $a = 10
PS C:\> .\paramtest.ps1 $a 2
```



Common usage of named and normal parameters

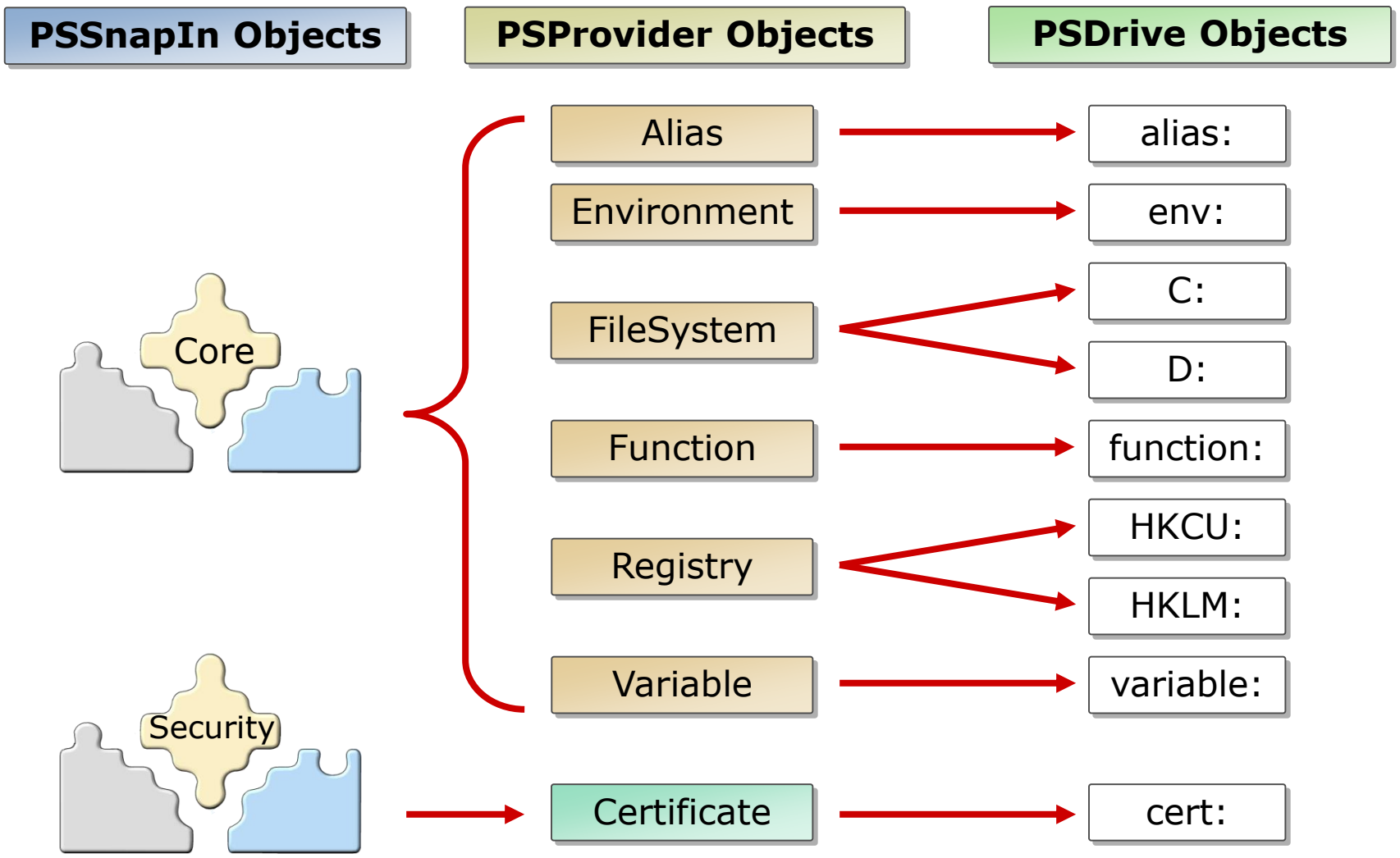
- You may mix the named and normal parameters.

```
#
param($x,$y)
write-output $args.length
# write-output $args.count
# the same as the previous
write-output $x
write-output $y
write-output „Starting with the  2. parameter:"
foreach( $i in $args )
  {“The script parameters one after the other {0:D};”
-f $i }
```

PowerShell (important) inner variables

- $_ - current pipe object, (foreach)
- $? – the status of the previous command result, logical
- $home – user’s home directory
- $$ - the last word of the previous command line
- $^ - the first word of the previous command line
- $host - current server (not a name!!)
- $myinvocation – current executional informations
- $pshome - PS install directory
- $profile – name of the user’s profile file

PS Source-Drive



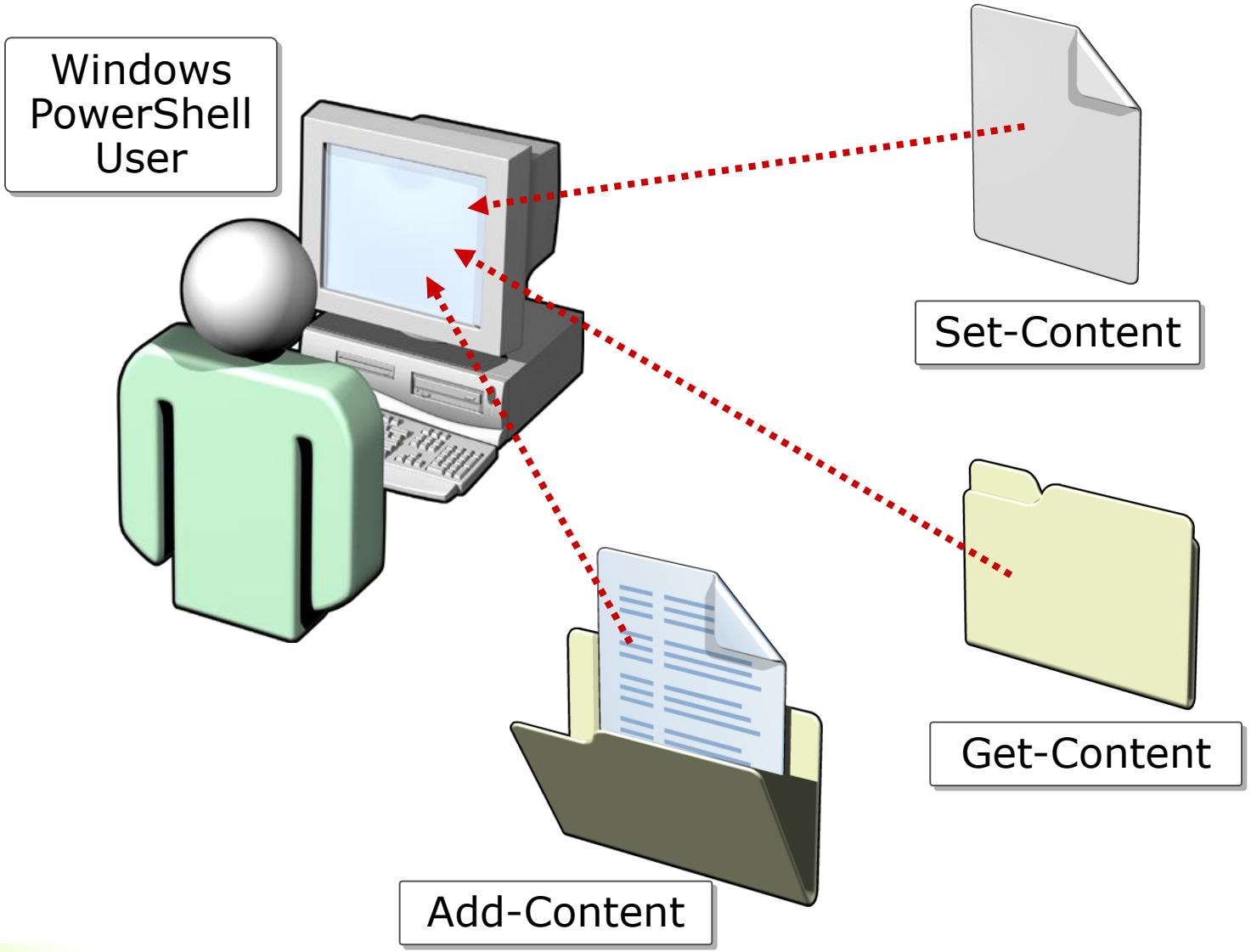
Data sources, Providers

- dir command is the same as get-childitem, it gives back the elements of the current data source.
- What kind of data sources are?
  - Get-PSDrive, Get-PSProvider
- How can we change between them?
  - Set-Location, e.g.: set-location alias:\
    - Cd hklm:
  - dir – Get-Childitem (for what does it refer?)
  - set-location d:\home

Output redirection (file creation)

- „Go Real Madrid!” > real.txt # overwrite, new file
  - „To refuse” >$null
  - Del real1.txt 2> $null # error output to somewhere else
    - 1> not exists, use simply >
- Get-Content real.txt # PS like usage
  - Cat real.txt # unix like usage
  - Type real.txt # dos like usage
- Append: >>
  - „Real Madrid - Reyo 10:2” >>real.txt # If there is no file yet then it is created!
- < or << redirections are missing!

Reaching files



Examples of file usage

- dir|set-content dir.txt # only the file-names will be added, why? (the elements of dir are objects)
- dir|out-string|set-content dir1.txt # the whole table is added
  - dir|out-file dir2.txt # the whole table-like writing
- dir|out-printer # prints to the default printer
- dir|export-csv dir.csv; import-csv dir.csv
- dir|export-clixml dir.xml; import-clixml dir.xml
- Out-null # similar to the /dev/null

Filtering data - Where command

- Data giving:
  - Pipeline
  - With parameters: -inputobject
- Where-Object { filterblock }
  - The filterblock lets the given object through the filter in the case of a logical true value. (logical operators: -gt, -lt, -eq, etc.)
  - E.g: dir|where-object { !$_.PSisContainer}
    - Listing of the objects which are not directories.
    - $_ The current value of the pipe (object).

Where-Object – foreach example

- Two versions of foreach with where-object
  - Where is similar to unix grep

```
$list = Get-ChildItem –Recurse|where-object {!$_.PSisContainer}
#we examine the sub-directories too
foreach ( $file in $list )
 {  $name = $file.name; $size = $file.length
    write-output “The size of $name file is: $size byte.”
 }
Get-ChildItem -Recurse | where-object {!$_.PSisContainer } |ForEach-
Object {
       $name = $_.name; $size = $_.length
       write-output “The size of $name file is: $size byte.” }
```

Regular expressions in PowerShell

Character	Meaning	Example
.	Optional character	.o.th
[xyz]	One from the listed ones	[CMRS]andy
[x-z]	One from the interval	[A-Z]eramy
^	The beginning of the text	^Subject:
$	The end of the text	meeting$
*	0 or more repeatition from the previous	W.*s
+	1 or more repeatition from the previous	[MZ]+any
?	0 or 1 previous	[MZ]?any
\	The following is a special character	Try\$

–like and –match operator example

```
PS> gci -r | where-object { $_.name -match “\.x[ml][ls]” }
... # reg. expr, the XML, XLS, XMS, XLL files.
```

```
PS> get-process | where-object { $_.name -match “ss$” }
... # all of the system service processes.
```

```
PS> $p = Get-Content planes.txt; `
    $p -match “a[ie]ro?plane”
... # lines containing words airplane, aeroplane.
```

```
PS> $p -like “*plane*”
... # there is the plane word in the lines.
```

Sorting – Sort in PowerShell

- Sort-Object [property] –parameters
  - If the property is given, it is sorted by it
    - E.g.: length
  - Parameters: -unique, -casesensitive, -descending, -culture name, E.g.: Get-Culture, Set-Culture commands
  - If there is no parameter or property then the whole object is sorted by name, ascending, not taking notice to casesensitivity (this is the default meaning).
  - E.g.: dir|sort

Select-Object - selection

- It selects object properties
  - E.g.: get-process|select-object processname,Id
  - Selects the name and identifier of processes.
- Important parameters: -first 4, -last 5, -unique
- Example: (you can write a userdefined hashtable)

```
$ get-process|sort-object processname|select-object -first 5
$
$ $p = get-process | select-object ProcessName,@{Name=„Starting time";
Expression = {$_.StartTime}}
$ $p
```

Operations on objects (Measure-Object)

- Measure-Object, averadge, sum etc.
  - Get-content dir.txt|measure-object -line –word –char
- It works using up one given property of the object.

```
PS C:\P> dir|measure-object -Property length -sum -Average -Maximum -
Minimum
Count    : 9
Average  : 3666,66666666667
Sum      : 33000
Maximum  : 11459
Minimum  : 120
Property : length
PS C:\P>
```

Process handling

- Get-Process
  - ps | Where-Object {$_.handles –gt 500}
- Finish a process
  - $p = Get-Process powershell
  - $p.kill gives the form of kill!
  - $p.kill() # it kills PowerShell…
  - ps|stop-process –whatif #what would happen if …
  - ps|where-object {$_.name –like „s*” } # filename
  - ps|where-object {$_.name –match „s*” } # fitting of regular expression

Execution in the background (PS 2.0)

- Start-Job –scriptblock {start-sleep 10}
- Get-job # we get the list of running programs
- Remove-job –id number #deletion
- Stop-job –id number # stops it
- Invoke-Command: Command execution in local or remote computer
  - Enable-PSRemoting –force
    - Firstly we have to give permission for handling PS sessions – if we want to use it!
  - Invoke command can be run in a PSSession or directly on a computer (-computer).

Service commands, starting, stopping…

```
PS C:\> Get-Command -Noun Service

CommandType Name              Definition
----------- ----              ----------
Cmdlet      Get-Service       Get-Service [[-Name] <String[]>] [-Co…
Cmdlet      New-Service       New-Service [-Name] <String> [-Binary…
Cmdlet      Restart-Service   Restart-Service [-Name] <String[]> [-…
Cmdlet      Resume-Service    Resume-Service [-Name] <String[]> [-P…
Cmdlet      Set-Service       Set-Service [-Name] <String> [-Displa…
Cmdlet      Start-Service     Start-Service [-Name] <String[]> [-Pa…
Cmdlet      Stop-Service      Stop-Service [-Name] <String[]> [-For…
Cmdlet      Suspend-Service   Suspend-Service [-Name] <String[]> [-…
```

Authentication object

- Get-Credential
  - $c= Get-Credential apple

```
#for username apple
$c=get-credential apple
write-host „Username: "+ $c.username
write-host „Password: "+ $c.password
```

- $c=Get-Credential
  - We use frequently authentication using Get-WmiObject command
  - E.g.: Get-WmiObject Win32_DiskDrive –computername server1 –credential $c

PowerShell addins (SNAPIN)

- PowerShell architecture is modular
  - gcm| Where-Object {$_.name -match "PSSnapin" } # Get, Add, Remove
- Get-PSSnapin # gives back the list of the current modules
- Add-PSSnapin Webadministration
- Remove-PSSnapin Webadministration
- Naturally previously you have to install: IIS webadmin snapin!

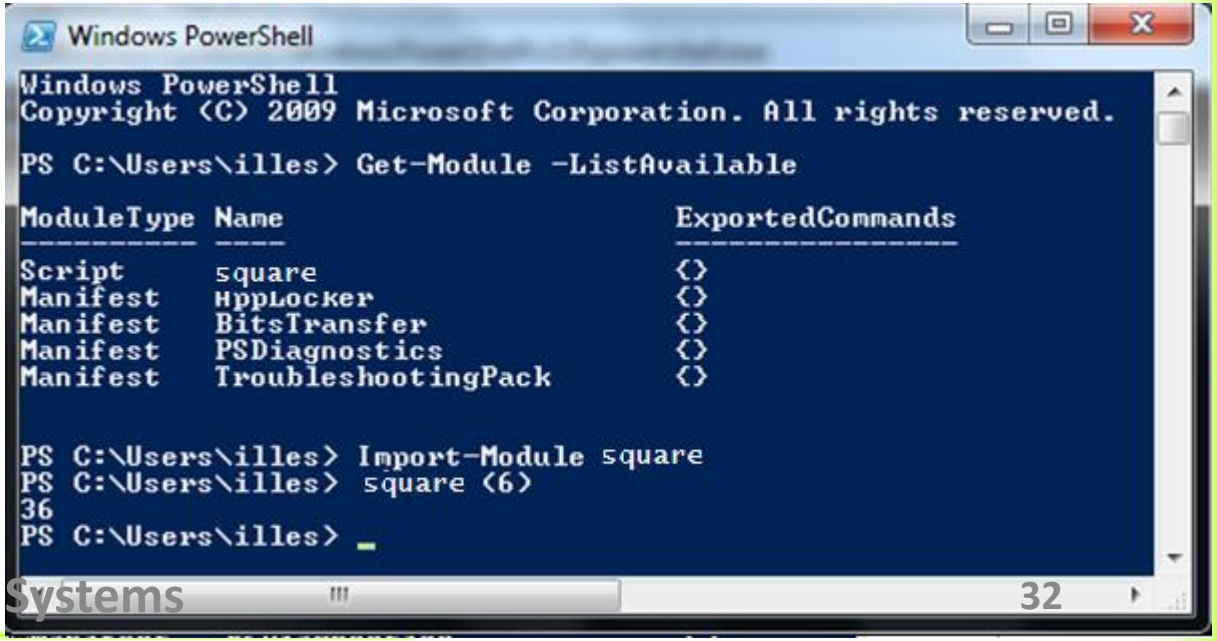
Powershell modules

- The SNAPIN is a binary format, so it has to be installed firstly.
- This module appeared first in PS 2.0, source-code
  - Get-Module – what modules are available
  - It is the collection of useful function-, alias-, variable-definitions.
  - $env:PSModulePath

```
PS D:\home\ps> $env:PSModulePath
C:\Users\illes\Documents\WindowsPowerShell\Modules;C:\Windows\system32\Windows
PowerShell\v1.0\Modules\
```

Custom module (Script module)

- 1. .psm1 is the extension for a custom module.
- 2. the name of the directory should be the same as the file and place it into directory: „My Documents\WindowsPowerShell\Modules”!
- 3. Import square



WMI

- Windows Management Instrumentation
  - Handling of the infrastructure
- WMI Tools (has to be installed separatly)
- WMI classes, namespaces
  - Get-WmiObject -Class __Namespace -Namespace root
- Get-WmiObject –list # list of wmi classes
  - Get-WmiObject Win32_Diskdrive
  - Get-WmiObject Win32_NetworkAdapter
  - etc…

Active Directory (PS 1.0)

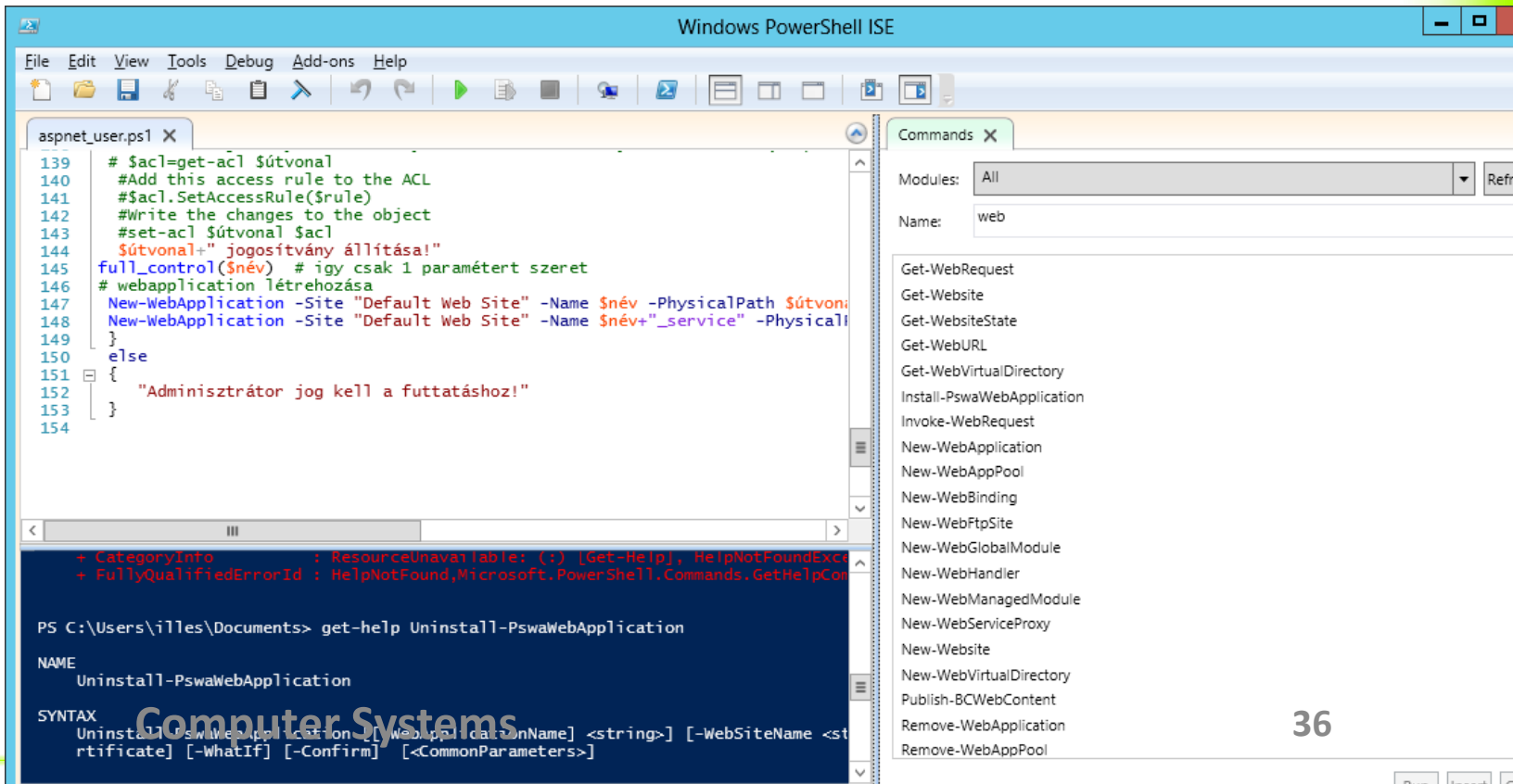
- ADSI – Active Directory Service Interface
- ADSI Providers
  - WinNT : NT4 PDC, BDC, and local users
  - LDAP : from Win2000 the AD-s are working with it
  - NDS : Novell Directory Services
  - E.g.: $a=[ADSI]”LDAP://dc=apple,dc=tree”
  - $u=$a.create(„organizationalunit”,”TestUnit”)
  - $u.setInfo();
- Etc....

Active Directory (PS 2.0)

- In Windows 2008 R2 server – appeared this module.
- Firstly it has to be installed as part of win2008 :
  - Active Directory for Windows PowerShell
- After it we import:
  - Import-module activedirectory
- We get a lot of commands:
  - Get-command –module activedirectory

Other possibilities I.

- IIS server handling
  - IIS gives the handler commands
  - aspnet.inf.elte.hu



Other possibilities II.

- Handling an exchange server
  - Exchange Management Shell-As administrator
  - Get-Excommand
  - New-ManagementRoleAssignment –Role „Mailbox Import Export” –user admin
  - New-MailboxImportRequest –mailbox username –FilePath \\computername\share\user1.pst
  - …
- SQL server handling
  - Get-help sqlserver

Pack it …

- Similar to our shell script packing!
- Usage, packing:
  - pack.ps1 file1 file2 … >package.ps1
- Packing out: package.ps1

```
# packing: pack.ps1
# Usage: pack.ps1 file1 file2 ...
"#Let us make a package"
foreach($i in $args)
{
  "echo $i"
  "@'"
  Get-Content $i # cat
  "'@ >$i"
  "echo '$i ends!'"
}
"#End of packing!"
```

Pack it out ..

```
PS C:\d\home\ps> cp .\pack.ps1 pack

PS C:\d\home\ps> cd pack

PS C:\d\home\ps\pack> .\pack.ps1
.\fradi.ps1
.\fradi.ps1 ends!
.\hajra.ps1
.\hajra.ps1 ends!

PS C:\d\home\ps\pack> ls


    Directory: C:\d\home\ps\pack


Mode                LastWriteTime         Length Name
----                -------------         ------ ----
-a----     2015. 11. 16.      8:52            526 pack.ps1
-a----     2015. 11. 16.      8:53            120 fradi.ps1
-a----     2015. 11. 16.      8:53             76 hajra.ps1
```

ELTE INFORMATIKA

Thank you!