Windows PowerShell
～.NETベースのシェル・スクリプト実行環境～

むたぐち＠わんくま同盟

シェルとは？

- シェル(shell)はオペレーティングシステムの機能の一部であり、ユーザーからの指示を受けて解釈し、プログラムの起動や制御などを行うプログラムである。（By WikiPedia）
- シェルには「プログラムの起動終了」「ファイルの入出力」「パイプ」「環境変数の参照と設定」「入力履歴の参照」「エイリアス」「入力補完機能」「制御構造」「バッチ（スクリプト）処理」といった機能がある。

シェルの種類

■GUI

-Windows系

- explorer.exe（エクスプローラ）など

-UNIX系

- X-Windowなど

■CUI

- Windows系

- command.com（DOSプロンプト）、cmd.exe（コマンドプロンプト）など

- UNIX系

- sh、bash、tcsh、zshなど

これまでのCUIシェル

- すべてはテキストベースである
  - パイプを渡るデータはテキスト
  - リダイレクトはテキストの入出力
  - 変数はテキストを格納

Windows系もUNIX系も同じく
テキストベースのCUIシェルを採用

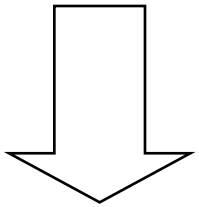
テキストベースのシェルの限界点

- 複雑なことをしようと思えば、複数のテキスト処理コマンドを複雑に組み合わせる必要がある。
- コマンドごとにさまざまに違うオプション、使い方を覚える必要がある。
- データの取り回しがテキストを介してでしか行えない。

所詮はテキストである

そこで周りを見渡すと

- 世の中はオブジェクト指向プログラミングの時代
- C++、JAVA、C#、VB.NET etc etc



クラスライブラリを活用して、
高効率、高生産性のプログラムを記述できる。

プログラマはうらやましいな

スクリプト（バッチ）だけなら結構イケてるけど…

- Windows98からWindows標準搭載のWSH（Windows Script Host）はVBScriptやJScriptからCOMコンポーネント（ActiveX DLL）のオブジェクトを呼び出して利用可能。

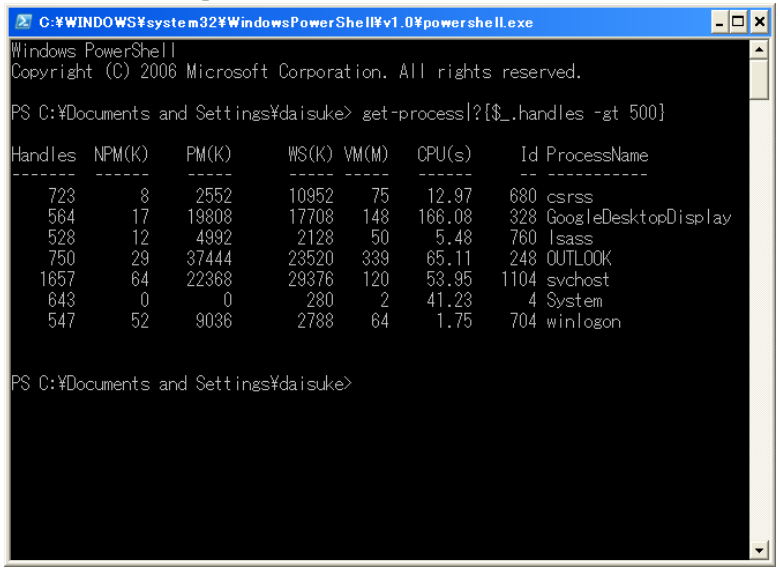
だけど

・WSHのSHはShellを意味していない。
→シェルとしての機能はない。

・COMは.NET全盛期の今となっては廃れつつある技術→トレンドではない。

そこで登場するPowerShell

- .NET Frameworkベースの新しいシェル･スクリプト実行環境
  それが、Windows PowerShellです。
  （開発コードMonad、旧称MSH(Microsoft Command Shell)）



PowerShellのダウンロード

- PowerShell v1.0の正式版がつい先日(2006/11/14)リリースされました！
  - Windows Server 2003 Service Pack 1 および Windows XP Service Pack 2 用の Windows PowerShell 1.0 ローカライズ版インストール パッケージ
  - http://support.microsoft.com/kb/926140

PowerShellのインストール条件

- Windows Server 2003 SP1
  or
  Windows XP SP2
- .NET Framework Version 2.0 (2.0.50727)

※Vista版は2007/1/31にリリース予定

PowerShellのPowerの源　コマンドレット

- PowerShellにはデフォルトで120種を超えるCmdlet（コマンドレット）が含まれている。
  - コマンドプロンプトで言うところの「内部コマンド」に相当
- コマンドレットを単独で、あるいは組み合わせることで様々な処理を実現可能。
- コマンドレットの引数も戻り値もみな.NETのオブジェクトである。
  - コマンドレット自体も…

コマンドレットの基本(1)　命名法

- コマンドレット命名法は

  ”Verb-Noun”（動詞-名詞）

- 例：ディレクトリを移動するSet-Locationコマンドレット(コマンドプロンプトのcdに相当)

```
Windows PowerShell
Copyright (C) 2006 Microsoft Corporation. All rights reserved.

PS C:¥Documents and Settings¥daisuke> Set-Location -Path C:¥
PS C:¥>
```

コマンドレットの基本(2)　ヘルプ

- どんなコマンドレットがあるのかを調べるには

  Get-Command

- コマンドレットのヘルプを引くには

  Get-Help コマンドレット名

  または

  コマンドレット名 -?

- .NETオブジェクトのメンバ（プロパティ、メソッドなど）を調べるには

  コマンドレットなどの後に|Get-Member

コマンドレットの基本(3)　パラメータ

- コマンドレットのパラメータはすべて

  -パラメータ名　または　-パラメータ名 パラメータ

  （c.f. Cmdlet -param1 value -param2 value -param3）
- パラメータによってはパラメータ名を省略できる。
- 文字列はスペースを含まない限り""で括らなくて良い。
- コマンドレットに共通のパラメータがある。
  - -?, -Verbose, -Debug, -ErrorAction, -ErrorVariable, -OutVariable, -OutBuffer, -WhatIf, -Confirm

統一的なコマンド体系

コマンドレットの基本(4)　省力化

- コマンドレットにエイリアスが定義可能。デフォルトでもいくつか定義されている。
  （Get-Aliasで一覧を取得可能）

Set-Location	sl, cd ,chdir
Get-ChildItem	gci, dir, ls
Get-Process	gps

etc

- コマンドレット、パラメータ名、パラメータ、すべて、大文字と小文字を区別しない。（変数、メソッド名なども）
- パラメータ名の省略、一部省略
  （-path→省略、-exclude→-ex）
- タブ補完
  （set-<Tab>→Set-Acl→Set-Alias→ Set-AuthenticodeSignature）

PSドライブ(1) 概要

- 従来のシェルはファイルシステムのドライブしか扱えなかったが、
- PowerShellでは、デフォルトで、ファイルシステム、レジストリ、デジタル署名、環境変数、エイリアス、スクリプト変数、関数を「PSドライブ」として扱うことができる。（Get-PSDrive）
- PSドライブを扱うための.NETプログラムが「PSプロバイダ」。
- コマンドレットとPSプロバイダを含む.NETアセンブリを「PSスナップイン」という。PSスナップインをインストールすることで機能拡張が可能。

PSドライブ(2) 項目の操作

作成	New-Item	ni
名前変更	Rename-Item	rni, ren
移動	Move-Item	mi, mv, move
コピー	Copy-Item	cpi, cp, copy
削除	Remove-Item	ri, rm, rmdir, del, erase, rd
実行	Invoke-Item	ii
プロパティ操作	Get-ItemProperty, Copy-, Clear-, Move-, Rename-, Remove-, Set-	

- これらの操作がどのドライブでも可能

PSドライブ(3) レジストリドライブの例

- レジストリプロバイダのおかげでレジストリもファイルシステムと同様にPSドライブとして操作できる。

```
PS C:¥> cd hkcu:
PS HKCU:¥> dir s*


   Hive: Microsoft.PowerShell.Core¥Registry::HKEY_CURRENT_USER

SKC  VC Name                           Property
---  -- ----                           --------
 43   0 Software                       {}
  0   1 SessionInformation             {ProgramCount}


PS HKCU:¥> cd 'HKCU:¥Software¥Microsoft¥Windows¥CurrentVersion¥'
PS HKCU:¥Software¥Microsoft¥Windows¥CurrentVersion> Get-ItemProperty Explorer
 . . .
WebFindBandHook          : {68F2D3FC-8366-4a46-8224-58EFA2749425}
FileFindBandHook         : {FFAC7A18-EDF9-40de-BA3F-49FC2269855E}
Logon User Name          : daisuke
ShellState               : {36, 0, 0, 0...}
 . . .
```

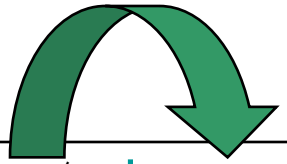
ワイルドカード使えます

タブ補完できます

従来のシェルにおけるパイプ

- カレントのファイルをファイル名で逆順ソート

  dir / bの出力＝テキストがパイプを通る



```
C:¥WINDOWS¥system32¥drivers¥etc>dir /b | sort /r
services
protocol
networks
lmhosts.sam
hosts
```

- 上の例は単純なテキストなのでソートできるが、ではサイズでソートするには？

  ？？？

オブジェクトが渡るパイプ(1) 概要

- ファイルサイズでソート、PowerShellなら可能です。
  dirの出力＝オブジェクトの配列がパイプを通る

```
PS C:¥WINDOWS¥system32¥drivers¥etc> dir | Sort-Object -property Length


    ディレクトリ: Microsoft.PowerShell.Core¥FileSystem::C:¥WINDOWS¥system32¥dri
    vers¥etc


Mode                 LastWriteTime     Length Name
----                 -------------     ------ ----
-a---          2004/08/05     21:00        407 networks
-a---          2004/08/05     21:00        734 hosts
-a---          2004/08/05     21:00        799 protocol
-a---          2004/08/05     21:00       3683 lmhosts.sam
-a---          2004/08/05     21:00       7116 services
```

FileInfoオブジェクトのLengthプロパティを元にソートされる。

オブジェクトが渡るパイプ(2) 実際にパイプを通っているもの

- コマンドレットの戻り値は.NETのオブジェクト

```
PS C:¥WINDOWS¥system32¥drivers¥etc> dir | get-member


   TypeName: System.IO.FileInfo

Name                      MemberType      Definition
----                      ----------      ----------
AppendText                Method          System.IO.StreamWriter AppendText()
CopyTo                    Method          System.IO.FileInfo CopyTo(String de...
Create                    Method          System.IO.FileStream Create()
CreateObjRef              Method          System.Runtime.Remoting.ObjRef Crea...
CreateText                Method          System.IO.StreamWriter CreateText()
・・・
LastAccessTime            Property        System.DateTime LastAccessTime {get...
LastAccessTimeUtc         Property        System.DateTime LastAccessTimeUtc {...
LastWriteTime             Property        System.DateTime LastWriteTime {get;...
LastWriteTimeUtc          Property        System.DateTime LastWriteTimeUtc {g...
Length                    Property        System.Int64 Length {get;}
Name                      Property        System.String Name {get;}
・・・
```

System.IO名前空間に属するFileInfoクラスのインスタンス(オブジェクト)

オブジェクトが渡るパイプ(3) パイプの連携

- パイプは繋げることももちろん可能

```
PS C:¥WINDOWS> dir -recurse | sort length -descending | select -first 3

    ディレクトリ: Microsoft.PowerShell.Core¥FileSystem::C:¥WINDOWS¥Driver Cache
    ¥i386
Mode                 LastWriteTime     Length Name
----                 -------------     ------ ----
-a---         2004/08/05     21:00   71286581 driver.cab

    ディレクトリ: Microsoft.PowerShell.Core¥FileSystem::C:¥WINDOWS¥system32¥con
    fig
Mode                 LastWriteTime     Length Name
----                 -------------     ------ ----
-a---         2006/11/27      1:08   43253760 software

    ディレクトリ: Microsoft.PowerS
    nstallations¥{59C4F14F-7590-45
Mode                 LastWriteTime
----                 -------------
-a---         2005/12/19     20:33   34321552 QuickTimeInstaller.exe
```

Windowsフォルダ配下を再帰的に検索し、サイズで降順ソートし、そのうち最初の3ファイルを取得

オブジェクトが渡るパイプ(4) フィルタ

- Where-Objectを使うとさらに細かくフィルタ可能

?またはwhereでも可

省略可

スクリプトブロック。$_にはパイプに渡されたオブジェクトが格納

```
PS C:¥> Get-Process | Where-Object -filterScript {$_.handles -gt 500}

Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
    699       7     2232       9568    74     6.72    696 csrss
    585      18    18476      12444   171    42.69   2312 GoogleDesktopDisplay
    523      12     4720       1908    50     1.77    776 lsass
    674      28    35224      54164   314    42.27   3692 OUTLOOK
   1781      63    28232      36856   176    53.22   1120 svchost
    640       0        0        280     2    14.00      4 System
    590      20    30236      48004   188   134.11   1220 unDonut
    541      51     9144       4680    64     1.50    720 winlogon
```

稼働中のプロセスからハンドル数が500より多いものを列挙

オブジェクトが渡るパイプ(5) 列挙

- Foreach-Objectでパイプを渡ったオブジェクト配列の要素それぞれに対してコマンド実行可能。

%またはforeachでも可

スクリプトブロック。$_にはパイプに渡されたオブジェクトが格納

```
PS C:¥Documents and Settings¥daisuke> dir | Foreach-Object -process{Write-Host $
_.FullName}
C:¥Documents and Settings¥daisuke¥Application Data
C:¥Documents and Settings¥daisuke¥Contacts
C:¥Documents and Settings¥daisuke¥Favorites
C:¥Documents and Settings¥daisuke¥Local Settings
C:¥Documents and Settings¥daisuke¥My Documents
C:¥Documents and Settings¥daisuke¥NetHood
C:¥Documents and Settings¥daisuke¥PrintHood
C:¥Documents and Settings¥daisuke¥Recent
C:¥Documents and Settings¥daisuke¥SendTo
C:¥Documents and Settings¥daisuke¥Templates
C:¥Documents and Settings¥daisuke¥UserData
C:¥Documents and Settings¥daisuke¥WINDOWS
C:¥Documents and Settings¥daisuke¥スタート メニュー
C:¥Documents and Settings¥daisuke¥デスクトップ
```

省略可

カレントにあるファイルのフルパスの一覧を表示

オブジェクトが渡るパイプ(5) 出力

- 出力形式も自由自在

  テーブル？　→　～ | Format-Table

  リスト？　→　～ | Format-List

  一覧？　→　～ | Format-Wide

  etc

- 出力先も自由自在

  コンソール？　→　～ | Out-Host (デフォルト)

  ファイル？　→　～ | Out-File もしくは>（リダイレクト）

  プリンタ？　→　～ | Out-Printer

  etc

WMIも自由自在(1) Before & After

- WMI (Windows Management Instrumentation)のクラスのインスタンスを簡単に呼び出せる。

WSH with VBScriptでは…

```
Set wbemServices = GetObject("winmgmts:¥¥.")
Set wbemObjectSet = wbemServices.InstancesOf("Win32_LogicalMemoryConfiguration")

For Each wbemObject In wbemObjectSet
    WScript.Echo "物理メモリの合計 (kb): " & wbemObject.TotalPhysicalMemory
Next
```

長ったらしい…

PowerShellではこんなに簡単！

```
PS C:¥> Get-WMIObject -class Win32_LogicalMemoryConfiguration -property TotalPhy
sicalMemory
```

省略すればさらに簡単！

```
PS C:¥> gwmi Win32_LogicalMemoryConfiguration -p TotalPhysicalMemory
```

WMIも自由自在(2) 実践編

```
PS C:¥> gwmi -list |? {$_.name -like "*disk*"}

CIM_LogicalDisk                         Win32_MappedLogicalDisk
Win32_LogicalDisk                       CIM_DiskPartition
 ・・・・

PS C:¥> gwmi Win32_LogicalDisk

DeviceID      : C:
DriveType     : 3
ProviderName  :
FreeSpace     : 13378641920
Size          : 160031014912
VolumeName    :
 ・・・・

PS C:¥> gwmi Win32_LogicalDisk -filter "DeviceID='C:'"
 ・・・・
```

"disk"という文字を含むクラスを探す

Win32_LogicalDiskクラスを取得

DeviceIDがC:のドライブだけを取得

WMIも自由自在(3) 応用編

- ちょっと凝ったこともできますよ

ユーザーの一覧を表示する

```
PS C:¥> gwmi Win32_OperatingSystem | select *user*
```

IPアドレスの一覧を表示する

```
PS C:¥> gwmi Win32_NetworkAdapterConfiguration -filter IPEnabled=TRUE | select -
ExpandProperty IPAddress
```

ログオフする

```
PS C:¥> (gwmi Win32_OperatingSystem).Win32Shutdown(0)
```

C:ドライブのボリュームネームを変更する

```
PS C:¥> $disk=gwmi Win32_LogicalDisk -filter "DeviceID='C:'" #オブジェクトを変数に
代入
PS C:¥> $disk.volumename="aaa"                               #プロパティ設定
PS C:¥> $disk.put()                                          #メソッド呼び出し
```

PowerShellスクリプティング

時間がないので次回予告的になりますが…

- スクリプトは*.ps1ファイルに記述。関連付けは手動で。
- デフォルトではセキュリティ上、スクリプトの実行ができないので、
  Set-ExecutionPolicy Unrestricted
  などを実行しておく必要がある。
- 基本はシェルの延長（バッチ的にコマンドラインを記述していく）。
- 文法はC#とPerlのあいの子のような雰囲気です。
- あらゆる.NETクラスをインスタンス化できる。New-Objectコマンドレット使用。スタティックメンバ呼び出しも可能。
- New-ObjectでCOMオブジェクトも呼び出せます。
- 容易なエラー処理。

スクリプトサンプル:RSS1.0/2.0のタイトル一覧を取得

```
function ReadRSS
{
    param ([string]$url, [int]$maxResults)              # パラメータ　URLと取得するタイトル数
    $client = new-object System.Net.WebClient;          # WebClientオブジェクト作成
    $client.Encoding = [System.Text.Encoding]::UTF8;    # EncodingクラスのUTF8プロパティ（スタティック）参照
    $xmldoc =  [xml]$client.downloadstring($url);       # ダウンロードした結果をテキストで得て[XML]にキャスト
    if ($xmldoc.rss -eq $null)                          # rssプロパティがなければ
    {
    # RSS1.0
        "[" + $xmldoc.RDF.channel.title + "]";          # ブログのタイトルを取得。
        $node = $xmldoc.RDF;                            # RDFプロパティを変数に代入
    }else{
    # RSS2.0
        "[" + $xmldoc.rss.channel.title + "]";          # ブログのタイトルを取得
        $node = $xmldoc.rss.channel;                    # RSSプロパティを変数に代入
    }

    for ($i = 0; $i -lt $maxResults; $i ++)             # 0からmaxResultの値までループ
    {
         $node.item[$i].Title;                          # channelのItem配列からTitleプロパティを取得。
    }
}

ReadRSS "http://rss.rssad.jp/rss/itm/rss2dc.xml" 10      #RSS1.0の例
ReadRSS "http://blogs.wankuma.com/mutaguchi/Rss.aspx" 10 #RSS2.0の例
```

PowerShell実演

DEMO