



Bash versus Powershell

Embate das shells

A Powershell, da Microsoft, baseia-se no framework .NET.

Mas isso supera as funcionalidades do Bash?

por Marcus Nasarek

Tanto o *Bash* quanto a *Powershell*, apresentada ao mundo através do novo sistema operacional da Microsoft, incluem comandos para navegar através dos diretórios, gerenciar arquivos e chamar outros programas. A administração do sistema é uma importante função da *shell*, e os dois representantes comparados neste artigo são equipados para ajudar a gerenciar sistemas a partir da linha de comando.

Enquanto o *Bash* normalmente depende de uma combinação de novas ferramentas e utilitários clássicos do Unix, a *Powershell* possui seu próprio conjunto de programas de linha de comando. O Windows se refere aos comandos da *Powershell* como *cmdlets*. O *cmdlet* chamado *Get-Process* é o equivalente ao *ps*, e o *Get-Content* corresponde ao *less*. A *Powershell* tem diferenças significativas em relação às shells de comando presentes nas versões anteriores do Windows. Neste artigo mostramos

como a *Powershell* do Windows Vista se compara ao *Bash*.

Para suportar o controle de programas, uma shell precisa de elementos para execução condicional. Os termos *for* e *while* avaliam uma variável para suportar um número definido de iterações. O **quadro 1** compara a saída de um contador com *for* na *Powershell* e no *Bash*.

Ambas as shells são semelhantes com relação ao controle de fluxo com *if* e *switch*. A definição de funções, o uso de variáveis de ambiente, a definição de escopos, o uso de expressões regulares e a avaliação do valor de retorno dos programas são semelhantes nos dois sistemas.

Permissão limitada

Uma diferença inicial entre a *Powershell* e o *Bash* se mostra quando um script é executado. A *Powershell* não executa arquivos de script por padrão, e portanto só suporta o uso interativo. Entretanto, a shell da Microsoft roda

scripts que tenham sido assinados digitalmente. A assinatura digital identifica o autor do script, pois essa é a única pessoa capaz de criar a assinatura por meios digitais. Aceitar o certificado de assinatura do autor significa que o usuário merece sua confiança.

Quadro 1: Laços

Um laço *for* na *Powershell*:

```
PS:> for ($i=1;$i -le 3;$i++) {
Write-Host $i }
1
2
3
PS:>
```

O mesmo laço no *Bash*:

```
bash[~]$ for ((i=1;i<=3;i++)); do
echo $i; done
1
2
3
bash[~]$
```

Exemplo 1: Saída de Get-Member

```
01 PS:> Get-Content Textdatei.txt | Get-Member
02
03     TypeName: System.String
04
05 Name      MemberType  Definition
06 ----      -
07 Clone     Method        System.Object Clone()
08 CompareTo Method        System.Int32 CompareTo(Object value), System.Int32 CompareTo(String strB)
09 Contains  Method        System.Boolean Contains(String value)
10 CopyTo    Method        System.Void CopyTo(Int32sourceIndex, Char[] destination, Int32 destinationIn...
11 ...
12 Length   Property
13
14 PS:>
```

A Powershell não executa nenhum script “desconhecido”. Para executar um script sem assinatura, é necessário mudar a política de execução para *RemoteSigned* (assinado remotamente) na linha de comando, assim:

```
PS:> Set-ExecutionPolicy RemoteSigned
```

Esse comando ordena que a Powershell aceite todos os scripts locais. Caso os dados tenham sido baixados, ou se tiverem sido anexados a um email, a shell continuará insistindo com relação à assinatura. A capacidade de evitar a execução de comandos de fontes externas é um novo recurso de segurança, e vírus em scripts como o *Love Letter*^[1] perdem sua virulência.

De forma bastante diferente, o Bash não depende apenas de assinaturas digitais para avaliar as permissões de execução de um script. Em vez disso, as permissões do sistema de arquivos determinam se o script pode ser executado.

As duas shells compartilham um número surpreendente de recursos na forma como lidam com a configuração do sistema. Uma novidade no mundo do Windows, contudo, é a forma como sua shell trata tudo como um sistema de arquivos, navegando não apenas nos sistemas de arquivo e nas unidades, mas também no temível *Registro*, o armazém de certi-

ficados e as variáveis de ambiente. Pode-se copiar, renomear e mover valores do Registro da mesma forma simples como se faz com arquivos em uma unidade de armazenamento. A Powershell se refere a esses sistemas de arquivo virtuais como *Providers* (provedores), implementando, então, uma filosofia que o Linux sempre ofereceu no Bash: “tudo é um arquivo”.

Encaminhamento

A ferramenta mais poderosa da Powershell é o *pipe*, que suporta a passagem de valores ordenada, permitindo assim que se use a saída de um comando como entrada de outro. O Bash evidentemente também suporta pipes, mas não exige nada dos arquivos de entrada e saída. Ele confia na capacidade do próximo comando em fazer algo de útil com a saída do comando anterior.

Os objetos podem ser consultados com o comando *Get-Member*, que gera como saída os elementos e funções do objeto, como mostra o **exemplo 1**. Por exemplo, o comando:

```
PS:> Get-Content Arquivotexto.txt | Sort {
    > $_.Length }
```

permite que se ordene as linhas de um arquivo texto de acordo com a propriedade *Length* (comprimento) do objeto.

Apesar de a passagem de dados ser um pouco mais complexa, essa orientação a objetos ajuda a padronizar as operações, e suporta o tratamento de estruturas de dados complexas. O Bash não pode competir nesse ponto; em vez disso, ele utiliza as capacidades de programas externos para lidar com estruturas de dados.

Por exemplo, o Bash precisa de um *parser* de XML externo, como o *Saxon* ou o *Xalan-J*, para examinar arquivos XML. O **exemplo 2** é um pequeno script de Powershell que carrega um *feed* RSS da Internet na forma de um arquivo XML. O script define a função *Mostra-LinuxRSS*, que obtém os feeds RSS atuais.

Conclusões

De uma forma, a Powershell se baseia no conceito do Unix segundo o qual vários pequenos utilitários são melhores que um único e grande programa. Ao mesmo tempo, ela adota uma abordagem orientada a objetos que simplifica a composição de grandes projetos ao custo de uma curva de aprendizado mais íngreme. O maior problema com objetos é a necessidade de investir muito tempo para descobrir de qual função ou objeto se precisa. O cmdlet *Get-Member* provavelmente verá um uso intensivo na Powershell.

O Bash é útil como ferramenta simples e direta para a maioria das tarefas cotidianas. Se houver a necessidade de se utilizar estruturas de dados mais complexas, pode-se lançar mão de linguagens mais complexas, como *Python*, *Perl*, *Ruby*, *Tcl*... ■

Mais Informações

[1] Alerta do CERT CA-2000-04 “Love Letter Worm”: <http://www.cert.org/advisories/CA-2000-04.html>

Exemplo 2: Mostra-LinuxRSS.ps1

```
01 # Show-RSS.ps1
02 # Declaracao de variaveis para a URL
03     $feed="http://rss.news.yahoo.com/rss/linux")
04     Write-Host -ForegroundColor "green" "RSS-Feed: " $feed
05 # Baixar o feed RSS
06     $wco = New-Object System.Net.WebClient
07     $rss = [xml]$wco.DownloadString($feed)
08 # Mostrar o titulo
09     Write-Host -ForegroundColor "red" $rss.rss.channel.title
10 # Mostrar o formulario pequeno
11     $rss.rss.item | Select-Object title,description | format-table
12 # Mostrar titulo e descricao das entradas
13     $rss.rss.channel.item | Select-Object title,pubDate,description | format-list
```