Progress report

Don Jones

I was recently writing a fairly long and complicated Windows PowerShell script that, as it was running, became fairly unresponsive. I had written it to be run as a scheduled task, so it didn't really produce much in the way of visible output. When I ran it for its first big test, I nevertheless started to get a bit worried that I'd accidentally written an infinite loop or some other problematic bit of script.

As the shell just sat there, pathetically blinking its little cursor, I asked myself, "Did I kill it?" Apparently, I have no confidence in myself because I quickly hit Ctrl+C to break the script. Time to add some progress reporting.

Blather, blather, blather

The first thing I wanted to do was to add a bunch of status messages, letting me know exactly what the script was doing. The shell lets you do this quite easily with the Write-Verbose cmdlet. Go ahead and try it in the shell:

```
Write-Verbose "Test Message"
```

If you just tried this, you will have noticed that it didn't do anything. That's because Write-Verbose sends objects to the special Verbose pipeline, which, by default, doesn't display its output. A built-in shell variable, $VerbosePreference, controls this pipeline. The default value for this variable is SilentlyContinue, which suppresses verbose output. Setting it to Continue, however, opens the pipeline:

```
$VerbosePreference = "Continue"
```

Now I can add a bunch of Write-Verbose statements to my script and get a detailed look at what's going on as it runs. And the beauty of this technique is that, when I've finished testing and troubleshooting, I can shut off all that extra chatter by setting $VerbosePreference back to SilentlyContinue at the beginning of my script.

There's no need to go and remove all the Write-Verbose statements. In fact, since they stay there in the script, anytime I need to run the script manually, I can easily switch the Verbose pipeline back on if necessary.

But I need real progress

Once I was satisfied that the script wasn't caught in an infinite loop and was, in fact, working perfectly, I shut

Cmdlet of the month: Tee-Object

This month, I want to take a look at one of my favourite troubleshooting cmdlets. Take this, for example:

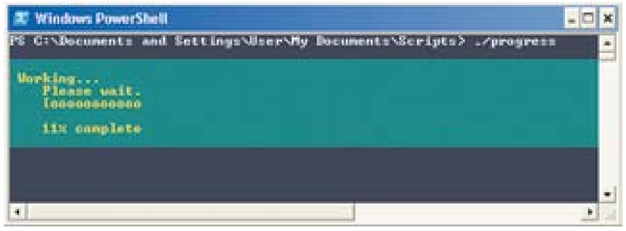
```
Get-WMIObject Win32_Service | Where { $_.State -ne "Running" -and $_.StartMode -eq "Automatic"
} | ForEach-Object { $_.Start() }
```

On the surface, this would seem to start all services that are set to start automatically but have not yet started for some reason. This doesn't actually work, though, and finding out why it doesn't work can be tricky since you can't peer inside the middle of the pipeline. That is, you can't peer inside the pipeline unless you use Tee-Object.

Tee-Object redirects objects to a file (or into a variable) and passes them down the pipeline. For example:

```
Get-WMIObject Win32_Service | Tee-Object AllServices.csv | Where { $_.State -ne "Running" -and
$_.StartMode -eq "Automatic" } | Tee-Object FilteredServices.csv | ForEach-Object { $_.Start()
}
```

This modification lets me see what happens after each pipeline command, and I quickly discover that my FilteredServices.csv file contains nothing! No wonder this script didn't work! A bit more research reveals the root cause of the problem – StartMode is 'Auto' not 'Automatic' – and Tee-Object let me pinpoint exactly where the problem was occurring.

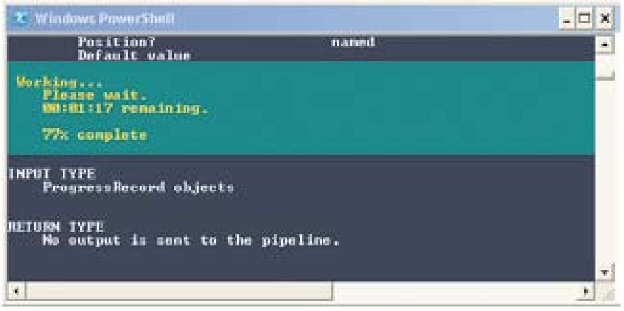


```
PS C:\Documents and Settings\User\My Documents\Scripts> ./progress

Working...
   Please wait.
   [oooooooooooo

   11% complete
```

Figure 1 **How far has your script progressed?**



```
        Position?                    named
        Default value

Working...
   Please wait.
   00:01:17 remaining.

   77% complete

INPUT TYPE
   ProgressRecord objects


RETURN TYPE
   No output is sent to the pipeline.
```

Figure 2 **How long until your script is complete?**

off the Verbose pipeline and ran it again – just to be sure.

The problem now is that, despite knowing that the script is functioning perfectly fine, I just can't bear staring at a blinking cursor. (I have attention span issues. I desperately looked for some paint so I could sit and watch it dry instead.)

What I needed was a general indication of how far along the script had progressed and some idea of when it would be done. Basically, I wanted something like a progress bar.

Fortunately, Windows PowerShell™ includes the Write-Progress cmdlet. This cmdlet doesn't provide a graphical progress bar like you see in Windows, but it does produce a nice progress bar nonetheless, as shown in **Figure 1**. It looks a bit like the file copy progress bar used by the text-based portion of Setup in Windows Server 2003 or even Windows XP.

Using Write-Progress requires a bit of explanation. Actually, I think an example would be even better. Consider this script:

```
for ($a=1; $a -lt 100; $a++) {
  Write-Progress -Activity "Working..." `
   -PercentComplete $a -CurrentOperation
   "$a% complete" `
   -Status "Please wait."
  Start-Sleep 1
}
```

It uses Write-Progress to display a progress bar. I've used Start-Sleep to make the script pause one second each time through the loop, so that it executes slowly enough for us to actually see the progress – without the pause, the loop counts from 0 to 100 so quickly that the progress bar merely flashes briefly on the screen.

As you can see, the Activity – which I've set to Working – shows up in the top of the progress bar. The Status is shown right below it, and the CurrentOperation is shown at the bottom. The shell only supports a single progress bar at a time. Any use of Write-Progress will either create a new progress bar if one isn't currently shown or update the bar that is currently shown.

What I haven't done here is told the bar to actually go away when it's finished. I can do that by simply adding this to the end of my script:

```
Write-Progress -Activity "Working..." `
 -Completed -Status "All done."
```

Generally speaking, the progress bar will vanish by itself once your script is complete, but if your script has other things to do you'll want to hide the progress bar when you're done with it; the -Completed parameter simply removes the bar from the display.

Tick, tick, tick

Another common use for Write-Progress is to create a 'seconds remaining' display, rather than an actual progress bar. Here's an example:

```
for ($a=100; $a -gt 1; $a--) {
  Write-Progress -Activity "Working..." `
   -SecondsRemaining $a -CurrentOperation
   "$a% complete" `
   -Status "Please wait."
  Start-Sleep 1
}
```

All I've done here is change the loop to count from 100 to 1, and I used the SecondsRemaining parameter of Write-Progress, rather than PercentComplete. The result is shown in **Figure 2**. As you can see, the progress meter is gone, replaced by a countdown clock. The shell automatically converts the total number of seconds remaining into hours, minutes and seconds, offering more user-friendly information. The percent complete shown here actually counts down from 100, since that's simply the CurrentOperation parameter I provided. There is no actual calculation of percent completed; the Windows PowerShell is simply displaying the current value of $a followed by the string "% complete."

Scripts that communicate

I'm a big fan of writing scripts that communicate what they're up to. It can take the form of verbose output or just a simple progress bar. It can be intended for my own attention-deficient benefit or for the benefit of someone else who will need to run my script months from now. In the end, displaying some kind of status and progress information will be a huge benefit.

For more help, read the Windows Powershell blog at: *http://blogs.msdn.com/powershell* ■

Don Jones *is a contributing editor for* TechNet Magazine *and co-author of* Windows PowerShell: TFM *(SAPIEN Press, 2007). He teaches Windows PowerShell (www.ScriptingTraining.com) and can be reached through the ScriptingAnswers.com web site.*