Advanced Scripting   
Jobs

Last Updated: 5/25/2024 11:35 PM Version 1  
Document Prepared for: CYBER360 Student

# Name Click here to enter name ID Click here to enter id

# Instructions

Answer all questions directly in this document. You will save and upload this completed document as your homework submission.

# Overview

*Asynchronous processing* is one of the oldest automation paradigms in multiprocess computing. The default behavior of almost every CLI is *synchronous processing*: after you type a command and press [Enter], the shell synchronously waits for the command’s process to finish before prompting for the next command line. But an asynchronously executed command or script is a *background process*, in which the CLI instantly presents another prompt, ready to accept another command, while the background process continues running independently of the shell.

In Powershell, an asynchronously executed command or script is called a *job*. The following PowerShell cmdlets are used to work with jobs:

* **Start-Job** # creates a new background process; same as the **&** suffix operator
* **Get-Job** #shows a list of jobs
* **Receive-Job** # gets results from a specific job
* **Remove-Job** # removes job from job table
* **Stop-Job** # stops a job
* **Wait-Job** # waits for one or more jobs to complete
* **Debug-Job** # debug a ScriptBlock executed by a job

Windows PowerShell (desktop edition) provides two more cmdlets used with workflows:

* **Suspend-Job** # saves process state and pauses job (workflows only)
* **Resume-Job** # loads and continues previously saved job (workflows only)

# Requirements

PowerShell (any platform/edition), Tasks 1-3.  
Windows PowerShell (Desktop edition), Tasks 4-7.

# Setup

No setup, but take a moment to study this “infinite loop” script:

**while ($true) {   
 $i++   
 Write-Output $i   
 Start-Sleep 2.5   
}**

Try it! Press **[Ctrl]+[C]** to interrupt the loop. You will use variations of this script block in these exercises. If you haven’t already figured out what the **Start-Sleep** cmdlet does:   
**Get-Help Start-Sleep**

# Task 1—Starting Jobs

## Steps

1. Create a new job that counts by one every 3 seconds.   
   Start-Job -ScriptBlock {while($true){$i++;$i;Start-Sleep 3}}
   1. The job information is returned.
      1. What is the job’s **Id**? Click or tap here to enter text.
      2. What is its **Name**? Click or tap here to enter text.
2. Start another job that counts by tens. This time, you will give the job a name.   
   Start-Job -Script {while($true){$i+=10;$i;Start-Sleep 3}} -name NotBy1
   1. From the job information returned:
      1. What is its **Id**? Click or tap here to enter text.
      2. What is its **Name?** Click or tap here to enter text.
3. Start another job, this time counting by hundreds. Use the same NotBy1 name.   
   Start-Job -Script {while($true){$i+=100;$i;Start-Sleep 3}} -name NotBy1
   1. From the job information returned:
      1. What is its **Id**? Click or tap here to enter text.
      2. What is its **Name**? Click or tap here to enter text.
4. Start one more job, this time counting by thousands. Again, name it NotBy1.  
   Start-Job -Script {while($true){$i+=1000;$i;Start-Sleep 3}} -name NotBy1
   1. From the job information returned:
      1. What is its Id? Click or tap here to enter text.
      2. What is its Name ? Click or tap here to enter text.
5. View a list of the background jobs.   
   **Get-Job**   
   What is the **Id** of the job that is counting by 1? Click or tap here to enter text. What is its name? Click or tap here to enter text.

# Task 2—Getting Data from Jobs

To get the output from a job, you need to know its name or its id.

## Steps

1. The **Receive-Job** cmdlet accepts a named parameter, **-Name** or **-Id**. This cmdlet can also take a positional parameter: use a string argument to specify the job’s name, or use an integer argument to specify the job’s id:   
   Receive-Job *<your-count-by-1-job’s-name-or-id>*
   1. Did you get results from the correct job? Click or tap here to enter text.
   2. How high has that job counted so far? Click or tap here to enter text.
2. Wait at least 3 seconds since the previous command then get the job data again from the count by 1 job  
   Receive-Job *<your-count-by-1-job’s-name-or-id>*
   1. Now how high has that job counted? Click or tap here to enter text.
   2. Was the data you received in step 1 also included in the data you received in this step? Click or tap here to enter text.
3. Get the data from the job named NotBy1  
   Receive-Job NotBy1
   1. Scroll carefully through your received output. From which of the “count-by-10”, “count-by-100”, or “count-by-1000” jobs did you receive data? Click or tap here to enter text.
4. Figure out how to just receive the output of just the “count by 10” job only. Your command line: Click or tap here to enter text.
5. You can store jobs in variables as well. Store each of the jobs in a variable, using the appropriate job Id numbers:   
   $j1=Get-Job *<your-count-by-1-job’s-id-number>*   
   $j10=Get-Job *<your-count-by-10-job’s-id-number>*   
   $j100=Get-Job *<your-count-by-100-job’s-id-number>*   
   $j1000=Get-Job *<your-count-by-1000-job’s-id-number>*
6. Get the count-by-1 job’s data:   
   **Receive-Job $j1**
7. You can also store data received from a job in a variable:   
   $j1data= Receive-Job $j1
   1. What data type is in **$j1data** ?   
      **$j1data.GetType().Name**   
      Interpret your output: Click or tap here to enter text.
   2. What data type are the elements of **$j1Data**? Click or tap here to enter text.

# Task 3—Terminating Jobs

## Steps

1. You can identify jobs by Id, Name, or by passing a job object. Let’s try using the object to get rid of the “count by 1” job:   
   Remove-Job $j1   
   What exception was produced? Click or tap here to enter text.
2. Try again, this time with the **–Force** switch parameter:   
   Remove-Job $j1 -force   
   Did you get an exception this time? If not, what is output? Click or tap here to enter text.
3. List the running jobs:   
   Get-Job   
   Is the “count by 1” job listed? Click or tap here to enter text.
4. Stop the “count by 10” job:   
   Stop-Job $j10
5. View the job object:   
   $j10
   1. What is the job’s **State**? Click or tap here to enter text.
   2. Does the job have more output data? Click or tap here to enter text.
6. Receive the “count by 10” job’s data:   
   Receive-Job $j10
7. View the job status  
   $j10
8. Now does this job have more data? Click or tap here to enter text.
9. Remove the job:   
   Remove-Job $j10
10. List the jobs again. Is the count by 10 job listed? Click or tap here to enter text.
11. Remove all the remaining jobs   
    Get-Job|Remove-Job -force
12. List the jobs again. Are they all gone? Click or tap here to enter text.

# Task 4—Create and Schedule a Job

To schedule a task for later execution, you can use the Windows *Task Scheduler*, or the Linux *cron* service (or its variants). Search or prompt a generative AI to learn how to use either of those tools.

Windows PowerShell (Desktop edition) provides cmdlets for scheduling jobs in Windows. For this task, you will create a job that executes every minute. Each time the job runs, it will update a log file that records the time and the number of running processes.

Creating a scheduled Windows job is a two-step process. First, you create a **JobTrigger** that specifies when and how often the job should run. Then you register the job in the Windows Task Scheduler, using the job trigger and the script block you want execute.

## Setup

For tasks 4-7, use Windows PowerShell (Desktop edition) with administrator privilege. If you don’t know the local account password for your login user, create a new user and password for this assignment. For example, here’s how to use PowerShell to create a new user named **admin1**:

* **[String]$plain = Read-Host** *# enter a password for your new user*
* **$pw = ConvertTo-SecureString $plain -AsPlainText -Force**
* **New-LocalUser -Name "admin1" -Password $pw -Description "Sch. Job User"**
* **Add-LocalGroupMember -Group Administrators -Member admin1**

## Steps

1. (*Note: this step isn’t strictly necessary* *unless you’re using Windows 11!* )  
   Prepare a **PSCredential** object with the correct username and password for your user.
   1. Make sure you know your current username:  
      **whoami**   
      You should notice that it follows one of these patterns:   
       **domain\user** (such as **BYUI\mer23012**), or   
       **host\user** (such as **TRIP-PC\mer23012**).   
      If you set up a new user, your username will be *<hostname>***\admin1** (or whatever name you chose for your new user), rather than your **whoami** result.
   2. Enter:  
      **$c = Get-Credential**  
      You’ll see a “Windows PowerShell credential request” pop up in a separate dialog window. Enter your username (exactly matching your **whoami** result, including the domain or machine prefix and backslash) and your password, then tap **[OK]**.
2. Prepare a new directory and a new log file in that directory:   
     
   md c:\programdata\cyber360   
   New-Item c:\programdata\cyber360\processes.log   
      
   If you created a new user **admin1** for this assignment, you must also grant permission to that user to write to the new log directory and file:   
      
   **$rule = [System.Security.AccessControl.FileSystemAccessRule]::new(**   
    **"admin1","Modify","Allow")**   
   **"C:\programdata\cyber360", "C:\programdata\cyber360\processes.log" | % {**   
    **$acl = Get-Acl $\_**   
    **$acl.AddAccessRule($rule); Set-Acl $\_ $acl**   
   **}**
3. Also, let’s prepare a script block with the logging code. Enter (all on one line):   
     
   $sb={ "$(get-date): Process count: $((get-process).count)" >>   
    c:\programdata\cyber360\processes.log }
4. Let’s set a couple of options for the scheduled job. Enter (all on one line):  
     
   **$opts = New-ScheduledJobOption**  **-StartIfOnBattery -ContinueIfGoingOnBattery**   
     
   *(If you’re running on a laptop, the default values for these options could prevent your job from starting or running.)*
5. The trigger is easy to set up. Depending on how you want your job to run, use different options to create the trigger. For this job we want it to start right away, then run every minute for two days. But we’ll need time to type more commands to register the scheduled job, so instead of starting it “right away,” we’ll schedule it to start three minutes after right now. Enter:   
     
   $rd = New-TimeSpan -Days 2   
   $ri = New-TimeSpan -Minutes 1   
   $at = (Get-Date).AddMinutes(3)   
   $t=New-JobTrigger -At $at -Once -RepetitionInterval $ri -RepetitionDuration $rd   
   1. *Alternatively, you could compress the above four command lines into just one:*   
      $t=new-jobtrigger -at (get-date).addminutes(3) -once -repetitioninterval (new-timespan -minutes 1) -repetitionduration (new-timespan -days 2)
6. Finally, register the job. Enter (all on one line):   
     
   Register-ScheduledJob -Credential $c -Trigger $t -Name LogProcessCount   
    -ScheduledJobOptions $opts -ScriptBlock $sb   
     
   *Note: again, the parameter -Credential $c is necessary on Windows 11, but you may not need it if you’re using a previous Windows version.*

# Task 5—View the Scheduled Jobs

You can Manage you scheduled jobs via PowerShell or the Windows Task Scheduler.

## Steps

1. View the scheduled jobs.   
   Get-ScheduledJob
2. Use the **Format-List** cmdlet to get a better view of the job’s details.   
   Get-ScheduledJob logprocesscount |fl
3. You can also view your Jobs with the Task Scheduler. Open your start menu and type **task s**, then under “Best match” click Task Scheduler. Find the PowerShell jobs at  **Task Scheduler Library/Microsoft/Windows/PowerShell/ScheduledJobs**   
   by navigating the left pane.
4. Explore the information.
   1. From the General tab, what user account is used to run the job? Click or tap here to enter text.
   2. From the Settings tab, may this task be run on demand? Click or tap here to enter text.

# Task 6—View Your Log

## Steps

1. To view your log data, just use PowerShell to read the file:   
    **Get-Content C:\ProgramData\cyber360\processes.log**   
   *(If there is nothing there, it either has not been three minutes, or your script is wrong. Wait or Fix.)*
2. To get the last line that was appended to the log:   
   Get-Content C:\ProgramData\cyber360\processes.log -tail 1
   1. How many processes are running? Click or tap here to enter text.
3. Start a few notepad instances  
   1..5|%{notepad}
4. Wait for a minute, then see the results.   
   Start-Sleep 60; Get-Content C:\ProgramData\cyber360\processes.log -tail 1
   1. Now how many processes are running? Click or tap here to enter text.
5. Get rid of the notepad processes.   
   get-process notepad|stop-process
6. Wait for a minute to see the results.   
   Start-Sleep 60; Get-Content C:\ProgramData\cyber360\processes.log -tail 1
   1. Now how many processes are running? Click or tap here to enter text.

# Task 7—Managing Scheduled Jobs

## Steps

1. You can enable or disable jobs. Disable your job:   
   Disable-ScheduledJob -Name LogProcessCount
2. Check the time when the log file was last updated:   
   **(Get-Item C:\ProgramData\cyber360\processes.log).LastWriteTime**   
   Does it match the time recorded in the file’s last line? Click or tap here to enter text.
   1. Wait for a minute, then check your logfile again to make sure the job is no longer triggering. What was the last line appended to the log file? Click or tap here to enter text.
3. Enable the job again:   
   Enable-ScheduledJob -Name LogProcessCount
   1. Wait a minute, then verify that the log file was updated with a new appended line. Copy the logfile’s new last line here: Click or tap here to enter text.
4. Windows keeps a log from your scheduled jobs are in this folder:   
   **~\AppData\Local\Microsoft\Windows\PowerShell\ScheduledJobs**   
   It will contain subfolders for each of your scheduled jobs, each named by its job name. Change to the subfolder named **LogProcessCount**:   
   cd ~\AppData\Local\Microsoft\Windows\PowerShell\ScheduledJobs\LogProcessCount
5. In that subdirectory you should see a folder named **output** and a file named **ScheduledJobDefinition.xml**. Get a directory of the output folder:   
   dir output
6. You should see subfolders with the most recent runs of the script. (By default, Windows only keeps up to the last 32.)
7. Explore one of those subfolders.
   1. What files does the subfolder contain? Click or tap here to enter text.
   2. Explore the files.
      1. What data format do you recognize? Click or tap here to enter text.
      2. What kinds of data do they contain? *(Hint: if it’s not obvious, try using a browser or VSCode to view the files.)* Click or tap here to enter text.
8. Finally, change back to your home directory and remove your scheduled job:   
   cd ~; Unregister-ScheduledJob LogProcessCount
9. Verify it is gone:   
   Get-ScheduledJob
10. Wait a minute, then have another look at the log file in **C:\ProgramData\cyber360**. Did it stop getting updates? Click or tap here to enter text.
11. Look again in **~\AppData\Local\Microsoft\Windows\PowerShell\ScheduledJobs**. Did unregistering the job remove the log data that was there? Click or tap here to enter text.

# Wrap-up

Clean up the log directory you created at the start of task 4:   
rm C:\ProgramData\cyber360\ -force

# Deliverable

Upload this document with completed answers to I-Learn Canvas.