Cyrus Backup Solution: A Powershell-based Modular Approach to Backups

Eric S. Claus

***Abstract*—Vital, yet often overlooked aspects of Information Technology (IT) systems in small and medium nonprofits are backup and recovery. There are numerous backup and recovery solutions on the market today. However, they are too expensive, complicated, and time consuming for these organizations, and are not flexible enough to keep up with the constantly changing IT infrastructure. This paper proposes a simple Powershell-based backup and recovery solution in a modular format with central monitoring and scheduling, which addresses the aforementioned struggles small nonprofit organizations face. This solution is called Cyrus Backup Solution.**

***Index Terms*—business continuity, backup, recovery, powershell, disaster recovery.**

I. Introduction

DISASTERS happen every day. They include tornadoes, hurricanes, fires, terrorist attacks, hardware failures, software failures, data loss, and security disasters such as data theft, malicious configuration, data corruption, and ransomware. Businesses of all sizes should prepare for disasters of any size and type.

Documents and other electronic data often need to be retained for legal, ethical, and financial reasons [1], [2], [3]. Backups are one of the most crucial aspects of disaster preparedness and response because they help protect electronic data in case the data becomes lost or corrupted [4]. These backups should be frequent, regular, complete, reliable, secure, organized, and quickly able to be restored. Because of this, IT departments need to log and test their backups frequently.

Small nonprofit organizations, those with around one hundred or less employees, often struggle with small Information Technology (IT) budgets, small and overworked IT teams, and quickly changing IT environments. Despite these challenges, such organizations are still in need of comprehensive and automated disaster recovery (DR) solutions that provide backup and restoration of their critical data and systems. Backup solutions require three main aspects: a price point that is affordable to small nonprofits, the simplicity to be used by already over-worked IT teams, and the flexibility to back up an extreme variance in service types, such as single applications, entire virtual machines, file shares, network appliances, and more. Existing DR solutions, however, are too expensive, complicated, and time-consuming for these organizations, and they are not flexible enough to keep up with the constantly changing IT infrastructure. This paper proposes a simple Powershell-based backup and recovery solution in a modular format with central monitoring and scheduling, which addresses the aforementioned struggles of small nonprofit organizations. This solution is called Cyrus Backup Solution, or CBS.

E. Claus is with the School of Computing, Southern Adventist University, Collegedale, TN, 37315 USA (e-mail: ericclaus@southern.edu).

II. Background

A. Existing Solutions

Several backup and restore solutions currently exist. Two of the biggest names on the market are Veeam Backup & Replication and Unitrends. Both provide backup, restoration, and high availability functionality for physical, virtual, and cloud systems. Both products back up Windows and Linux workstations via an installable client. Both also back up virtual machines on VMware vSphere (paid editions) and Microsoft Hyper-V. Furthermore, both allow for easily expandable backup storage on multiple platforms. In regards to recovery, both provide multiple ways to quickly recover virtual machines, individual files and folders, and individual database tables.

One of the biggest drawbacks to Veeam and Unitrends is their prices. Veeam's Enterprise Plus Edition, the only version to offer its full feature set, has a price much higher than what the small, or non-existent, budgets of small nonprofits can afford. Veeam offers three cheaper editions; however, these lack many of the features that make Veeam a good solution. Veeam even also has a free version, although this edition is not sufficient for any organization wanting to take backups seriously because it lacks the ability to natively schedule and automate backup operations. Likewise, Unitrends offers backup appliances that are too expensive for many small nonprofits. Additionally, both Unitrends and Veeam Backup and Replication come with steep learning curves.

While there are numerous other backup solutions by smaller vendors, these have many of the same pros and cons as Unitrends and Veeam. Products such as Acronis, Barracuda Backup, and others are often cheaper than Unitrends and Veeam, but they do not come with the same level of support, nor offer as many features.

B. Best Practices Regarding Powershell Code Organization

Powershell offers two main ways to organize code and break large code projects into smaller chunks. The first method is dot sourcing [5]. Dot sourcing allows funtions from other scripts to be included. This is quick to implement, as simply including the following line of code will include any function contained inside of the referenced Powershell script:

```
. c:\path\to\file.ps1
```

The biggest downside to this, however, is that it requires excessive manual labor to maintain when moving Powershell scripts or renaming them [6]. To address this, Powershell modules provide a means to easily import functions automatically when a Powershell session starts. A Powershell module is a script containing various functions and variables saved as a .psm1 file type [7]. Saving this module into the directory specified in PsModulePath, an environment variable loaded by default, will cause the module, and all functions contained within it, to be loaded during each Powershell session [8]. Modules also provide a way to centralize all of your functions and easily move your Powershell solution from one location to another.

III. Proposed Project

One possible solution to the problems described in the Introduction and Existing Solutions sections is to utilize Powershell. Cyrus Backup Solution (CBS) is based on Powershell, which can be used to automatically backup IT services such as SSH-enabled network appliances, file shares, Group Policy objects (GPOs), and virtual machines on Hyper-V. Veeam Backup and Recovery Free Edition does not allow for scheduled backups, as does its paid counterpart. It does, however, offer Powershell integration with a Veeam Powershell module (see Table I for a full list of prerequisites for CBS). This opens up the door for home-grown VM backup solutions to be built using free technology. In addition to its backup functionality, CBS Powershell base allows for easy scalability and adaption to new environments. CBS offers many essential backup features for no cost other than low man-power costs. For a high level overview of the functional requirements for CBS, see Fig. 1. For the full list of requirements, see Appendix A.

CBS is not a closed, sealed system. It is a scripting solution with the purpose of automating manual tasks. Subsequently, it does not use proprietary file formats for its backup files. CBS instead used the same file types for its backups as would often be used if manually backing up a system. For example, backups of file shares are 7Zip (.7z) files and backups of virtual machines are Veeam Backup & Recovery (.vbk) files. This means that Backup Admins can take backups that have been done automatically via CBS and restore them manually. Because of this, automatic restore functionality is a ”should have” functional requirement and not a ”must have” functional requirement.

A. Why Powershell?

With a large range of available programming and scripting languages available to IT teams, it can be difficult to pick one language to create new solutions with. Microsoft Powershell is able to fully manage Windows systems. In addition, Powershell can be used to manage many non-Windows systems. Modules providing SSH capabilities lead to Powershell being able to manage just about any system with relative ease.

Being a high-level scripting language, Powershell is easier to read and write than most languages. Additionally, it requires less of a learning curve for system administrators familiar with Windows than other languages. This means that IT teams can easily expand a Powershell-based backup solution, like CBS, to meet the needs of changing IT environments.

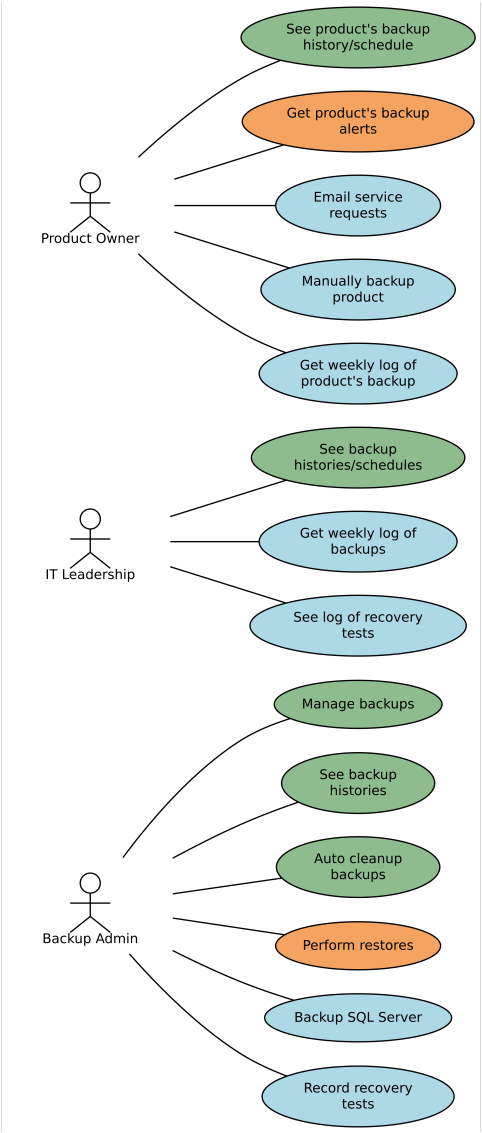

Fig. 1. A high level overview of the functional requirements for CBS. Key: Green = must have, Orange = should have, Blue = could have.

B. How Cyrus Backup Solution Works

CBS operates on these core principles: centrality and modularity. The temptation with scripting languages like Powershell can be to create multiple scripts and run them separately. This, however, makes it difficult to make changes that affect multiple scripts. The opposite of that approach is combining all functions into one massive script. While this eliminates the need to touch multiple script files for changes, this method comes with its own challenges. For example, it can be horrendous to try and shift through the code in such a large file.

CBS seeks to take the middle way between these two paths, that of complete separation and that of one massive script. As shown in Fig. 2, at the core of CBS is a single Powershell script which is run via Windows Task Scheduler. The script

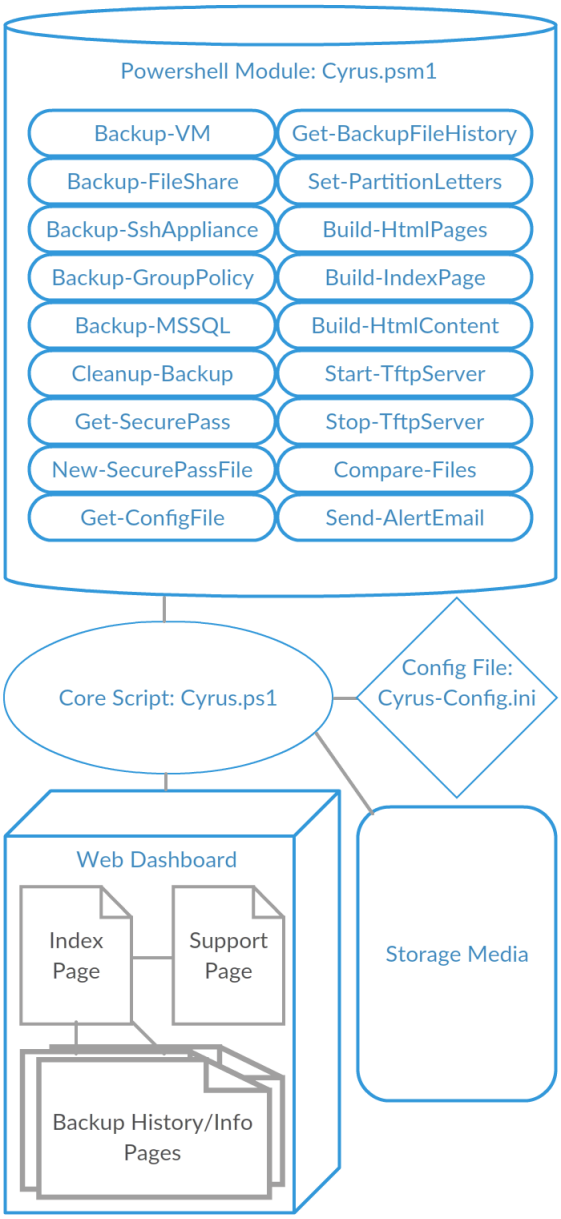

Fig. 2. An overview of the major components of CBS.

calls multiple functions,, based in a Powershell module (see Appendix C), which perform the bulk of CBS's functionality. The core script integrates with, and ties together, each of the major components of CBS with these Powershell functions.

The Powershell functions contained within the module perform tasks such as:

1) Backing up the various services
2) Automatically deleting backup files older than their retention periods
3) Managing secure storage and retrieval of passwords
4) Reading in configuration files containing information about each service to be backed up
5) Getting histories of each backup
6) Automatically setting the assigned drive letters for rotating external hard drives
7) Building the HTML pages for the web dashboard
8) Starting and stopping the TFTP server
9) Comparing backup files and looking for any changes (can be used for incremental backups)
10) Sending alert emails upon a backup failure
11) Restoring backups of select services

TABLE I
SOFTWARE AND HARDWARE PREREQUISITES FOR RUNNING CBS.

Item	Reason
Windows Server 2016	Operating system to run CBS on
Veeam Backup & Recovery Free Edition	Integrate with Powershell to backup VMs on Hyper-V
SolarWinds TFTP Server	Accept files transfered via TFTP from SSH appliances
Microsoft IIS	Web server to host the web dashboard
Storage media for backups (at least two external hard drives are recommended)	Store backup files

A configuration file contains the following information about each service to be backed up (see Appendix B for an example of the config file):

1) Service name
2) Backup schedules/frequencies
3) Retention periods
4) Directory where backup files will be stored
5) What type of service it is (VM, file share, Group Policy, SSH-enabled appliance, application)
6) Information on how to connect to the service (such as IP addresses or DNS hostnames, username, name of secure password file)
7) Product owner's email address

Cyrus Backup Solution contains a web dashboard used to monitor the backups. As shown in Fig. 2, the web dashboard has an index (home) page that contains links to pages displaying the history of each backup along with the information about each backup, including schedules, retention periods, and locations for backup files. These pages are generated via Powershell functions. The histories are dynamically generated by scanning the backup directory, where the backup files are stored, for each item being backed up and creating HTML tables containing each backup file's name, size, creation time, and last modified time. The backup information for each page is pulled from the configuration file.

The expected final deliverables for this project include: the Powershell module, the core script, the configuration file, and a skeleton site for the web dashboard. These will also include documentation on the project's Github repository.

C. Time Line

As shown in Table II, the development of CBS will be broken up into seven phases. The first phase will include the creation and configuration of a virtual machine to run CBS on. This will also include installation of Windows Server 2016 and the other prerequisites shown in Table I. The second phase will include the creation and testing of the Powershell functions used to perform the various backups. The third phase will include the creation and testing of the rest of the Powershell

TABLE II
TIME TABLE FOR THE DEVELOPMENT OF CYRUS BACKUP SOLUTION.

Task	Number of Hours
Create Windows Server 2016 VM to run CBS on	5
Write Powershell functions for performing backups	20
Write auxiliary Powershell functions	20
Write core script and configuration file	30
Design and develop web dashboard	10
Integrate major components	10
Create any additional documentation, perform testing	20
Total	**115**

functions that will be contained within the Powershell module. The fourth phase will include the creation and testing of the core Powershell script and the configuration file. The fifth phase will include the design and development of the web dashboard. The sixth phase will include the integration and testing of each of the major components of CBS (shown in Fig. 2). The last phase will include the creation of any additional documentation not already created, along with the acceptance testing.

IV. TESTING AND EVALUATION

As shown in Table III, CBS will have acceptance, unit, module, integration, and system testing performed on it throughout its development.

A. Acceptance Testing

This project will have five acceptance tests completed in order to evaluate it. These metrics will be numerical survey results evaluating five ”must have” functional requirements for the Backup Admin user role. This will be accomplished via a focus group of three system administrators responsible for backup and recovery. The Backup Admins in this focus group will be given CBS, along with all documentation on the project's Github repository, for one week.

Additionally, the focus group participants will be given a survey to answer questions about each functional requirement they are testing. The survey will evaluate on a numerical scale ranging from one to four, with the ratings below. Each test will be successful if it receives an average score of three or higher.

1) One meaning the functional requirement was not achieved at all
2) Two meaning it was partially achieved
3) Three meaning it was achieved but was too slow (subjective to the focus group participant)
4) Four meaning it was achieved and performed quickly (subjective to the focus group participant)

The functional requirements that will be tested by this focus group are:

1) “As a Backup Admin, I must be able to back up file shares.” To test this, participants will be asked to configure CBS to back up a file share incrementally every hour, on the bottom of the hour, with full backups completed daily. This function requirement is successful if the file share is correctly backed up incrementally and hourly on the bottom of the hour and if a full backup is correctly completed daily.
2) “As a Backup Admin, I must be able to back up SSH-enabled appliances.” To test this, participants will be asked to configure CBS to back up an SSH enabled appliance (such as a network switch) every hour, on the bottom of the hour. This function requirement is successful if the SSH appliance is correctly backed up hourly on the bottom of the hour.
3) “As a Backup Admin, I must be able to change the schedules of all backups via one centralized configuration file.” To test this, participants will be asked on day three of the focus group testing to change the incremental file share and SSH appliance backups being tested to run daily at 11:00 p.m. This functional requirement is successful if the modified backups are correctly backed up hourly on the top of the hour.
4) “As a Backup Admin, I must be able to define retention periods for all backups via one centralized configuration file.” To test this, participants will be asked to set the retention period for the SSH enabled appliance backup to two days. Participants will then be asked whether or not the backup files older than two days are deleted. This function requirement is successful if the SSH appliance's backup files are successfully deleted when older than its retention period.
5) “As a Backup Admin, I must be able to see web pages showing the history of all of the backups, including file names, creation date and time, last modified date and time, and size.” To test this, participants will be asked to view the CBS web dashboard daily to monitor their backups. This function requirement is successful if participants are able to see the correct history of their backups, including the automatic deletion of backups files older than their retention period.

B. Unit Testing

The Pester Powershell unit testing framework may be used to test each unit of code contained within the Powershell module. In addition, unit testing will be done manually in order to test five Powershell functions during development:

1) Backup-VM: used to back up virtual machines (VM) in order to satisfy functional requirement number 1 (see Appendix A). To test this, the function will be used to attempt to backup a VM. If successful, the VM will be successfully backed up.
2) Backup-FileShare: used to backup file shares in order to satisfy functional requirement number 2 (see Appendix A). To test this, the function will be used to attempt to backup file share on the network. If successful, the file share will be successfully backed up.
3) Backup-GroupPolicy: used to back up Group Policy objects (GPOs) in order to satisfy functional requirement number 3 (see Appendix A). To test this, the function will be used to attempt to back up all GPOs in a domain. If successful, the GPOs will be successfully backed up.
4) Backup-SshAppliance: used to back up SSH enabled appliances in order to satisfy functional requirement number 4 (see Appendix A). To test this, the function

TABLE III
TESTS TO BE PERFORMED DURING DEVELOPMENT.

Type	Item Being Tested	Tool
Acceptance	5 ”must have” functional requirements	Focus group
Unit	All functions within the Powershell Module	Pester
Unit	5 mission critical Powershell functions	Developer
Module	Powershell module and all functions within it	Developer
Integration	Powershell module and web dashboard	Developer
System	All ”must have” functional requirements	Developer

will be used to attempt to back up a network switch. If successful, the switch will be successfully backed up.

5) Get-ConfigFile: used to read in the configuration file, which specifies information regarding each backup-such as name, backup frequency/schedule, and retention periods-in order to help satisfy functional requirements 8 and 9 (see Appendix A). To test this, a configuration file will be written and Get-ConfigFile will be used to read it. If successful, the output of Get-ConfigFile will be a hast table containing all of the information contained in the configuration file.

C. Module Testing

Module testing will be conducted once the first draft of the Powershell module for CBS is built. The module will be imported into a computer not used for development and each function inside of it will be tested. The modules will be tested by calling them manually from a Powershell script. If successful, each function will perform correctly. This will be repeated on a minimum of five different computers, and the module testing will be successful is the testing succeeds on at least 80% of the computers tested on.

D. Integration Testing

Integration Testing will be performed to ensure that the Powershell module integrates successfully with the web dashboard. CBS will be installed on a non-development machine and run for two days. At the end of the test, if successful, the web dashboard will properly display backup histories and backup schedules. This will be repeated on a minimum of five different computers, and the integration testing will be successful is the testing succeeds on at least 80% of the computers tested on.

E. System Testing

System testing will be performed by the CBS developer before the acceptance testing is performed by the focus group mentioned above. All ”must have” functional requirements will be tested by the CBS developer on a non-development machine. Each requirement is successful if it is correctly met. This will be repeated on a minimum of five different computers and in at least two different networks and domains. The system testing will be successful if the testing succeeds on at least 80% of the computers tested on. If successful, CBS will move on to acceptance testing.

V. CONCLUSION

There is a great need among the small nonprofit market for an affordable, simple, and flexible backup and recovery solution in order to help prepare for disaster recovery and business continuity. As proposed in this paper, Cyrus Backup Solution (CBS) meets the challenges faced by small nonprofit organizations' IT departments. CBS provides Backup Administrators with the ability to quickly and easily backup critical data and services. The expected deliverables-the Powershell module, the core script, the configuration file, the web dashboard, and the documentation-will help IT departments secure their ability to successfully respond to any type of disaster.

REFERENCES

[1] Document Retention Policies for Nonprofits, National Council of Nonprofits. [Online]. Available: https://www.councilofnonprofits.org/tools-resources/document-retention-policies-nonprofits. [Accessed: 24-Sep-2018].

[2] P. Dorion, Developing an electronic data retention policy, TechTarget, Nov-2010. [Online]. Available: https://searchdatabackup.techtarget.com/tip/Developing-an-electronic-data-retention-policy. [Accessed: 24-Sep-2018].

[3] J. Phillips, IT and records managers should team up on data retention policies, Computerworld, 19-Oct-2005. [Online]. Available: https://www.computerworld.com/article/2559157/data-center/it-and-records-managers-should-team-up-on-data-retention-policies.html. [Accessed: 25-Sep-2018].

[4] Childs, Donna R. Prepare for the Worst, Plan for the Best: Disaster Preparedness and Recovery for Small Businesses. Edited by Donna R Childs. 2nd ed. . Hoboken, N.J.: Hoboken, N.J.: Wiley, 2008.

[5] . (source or dot operator), SS64.com. [Online]. Available: https://ss64.com/ps/source.html. [Accessed: 14-Oct-2018].

[6] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.

[7] J. Aiello, M. Blachman, and S. Wheeler, Writing a Windows PowerShell Module, Microsoft Docs, 12-Sep-2016. [Online]. Available: https://docs.microsoft.com/en-us/powershell/developer/module/writing-a-windows-powershell-module. [Accessed: 24-Sep-2018].

[8] A. Bertram, PowerShell Modules: How and When to Create and Use Modules, business.com, 05-Sep-2018. [Online]. Available: https://www.business.com/articles/powershell-modules/. [Accessed: 24-Sep-2018].

APPENDIX A
REQUIREMENTS LIST

Roles:

- Backup Admin: IT staff members responsible for the backup and recovery of all IT systems.
- IT Leader: High level IT leadership, such as managers and directors, along with C-suite executives.
- Product Owner: Users who are responsible for an individual system (ie. a virtual machine or an application).

Functional Requirements:

1) As a Backup Admin, I must be able to back up virtual machines on Hyper-V hypervisors.
2) As a Backup Admin, I must be able to back up file shares.
3) As a Backup Admin, I must be able to back up group policy objects.
4) As a Backup Admin, I must be able to back up SSH enabled appliances.
5) As a Backup Admin, I must be able to quickly create scripts and functions, and utilize existing scripts and functions, to backup various other software systems.
6) As a Backup Admin, I must be able to see web pages showing the history of all of the backups, including file names, creation date and time, last modified date and time and size.
7) As a Backup Admin, I must be able to change the schedules of all backups via one centralized configuration file.
8) As a Backup Admin, I must be able to define retention periods for all backups via one centralized configuration file.
9) As a Backup Admin, I must be able to have backup files automatically deleted if they are older than their specified retention policy.
10) As an IT Leader, I must be able to see web pages showing the history of all backups, including file names, creation date and time, last modified date and time and size.
11) As an IT Leader, I must be able to see a web page showing all backup schedules.
12) As a Product Owner, I must be able to see a web page showing the history of my products backup, including creation date and time.
13) As a Product Owner, I must be able to see a web page showing my products backup schedule.
14) As a Backup Admin, I should be able to perform restores of auto restore-enabled backups from the backup solution.
15) As a Product Owner, I should be able to receive alerts when my products backup is not working.
16) As a Backup Admin, I could be able to backup databases from Microsoft SQL Server.
17) As a Backup Admin, I could be able to record all restore tests in a central location.
18) As an IT Leader, I could be able to see a history of all restore tests.
19) As an IT Leader, I could be able to receive a weekly email with the history of all backups.
20) As a Product Owner, I could be able to send an email via the web page to request a restore of my products backup.
21) As a Product Owner, I could be able to send an email via the web page to request a change to my products backup schedule.
22) As a Product Owner, I could be able to manually run backups of my product, up to a predetermined number of times in a predetermined span of time.
23) As a Product Owner, I could be able to receive a weekly email with the history of my products backup.

Non-functional Requirements:

- The system shall run on Windows Server 2016 R2.
- The system shall be built with no technologies other than Powershell, HTML, CSS, JavaScript, Microsoft IIS, and SQL Server.
- The web dashboard shall function and appear correctly on Google Chrome version 70.

APPENDIX B
EXAMPLE CONFIGURATION SCRIPT "CYRUS-CONFIG.INI"

```
; Place this file in the same
;  directory as the Cyrus.ps1 script.
; Types: VM, FileShareFull, GP,
;        FileShareIncremental,
;        Fortigate, Procurve,
;        SshFull, SshIncremental,
;        Other (specify backup
;        function name)
; Frequency: [Hourly,top|bottom],
;            [Daily,<time>],
;  [Weekly,<day>,<time>]

[ExampleVM-1]
Name=ExampleVM-1
Type=VM
Frequency=Daily,2200
Retention=60
BkDir=V:\VMBackups\ExampleVM-1
Host=myHypervHost
Owner=user@domain.com

[ExampleFileShareFull]
Name=ExampleFileShareFull
Type=FileShareFull
Frequency=Weekly,Saturday,0200
Retention=90
BkDir=V:\FileShareBackups
NetPath=\\fileSvr-1\ExampleFileShare
Owner=user2@domain.com

[ExampleApplication]
Name=ExampleApplication
Type=Other
Function=Backup-MyApp
Frequency=Hourly,top
Retention=7
```

```
BkDir=V:\MyAppBackups
NetPath=\\myAppSvr-1
Owner=user2@domain.com

[ExampleFortigate]
Name=ExampleFortigate
Type=Fortigate
Frequency=Daily,2300
Retention=90
BkDir=V:\FirewallBackups
NetPath=10.0.0.1
Owner=user3@domain.com
```

APPENDIX C
SKELETON POWERSHELL MODULE "CYRUS.PSM1"

```
<#
.SYNOPSIS
    Powershell module for the Cyrus Backup Solution

.DESCRIPTION
    This module contains functions used for the Cyrus Backup Solution.
    This is an example module file.

.NOTES
    Author: Eric Claus
    Last Modified: 11/25/2018

.LINK
    https://github.com/ericclaus14/Cyrus
#>

# Import required modules
Import-Module Posh-SSH
Import-Module 7Zip4PowerShell

# Perform backups
function Backup-VM {}
function Backup-FileShare{}
function Backup-SshAppliance{}
function Backup-GroupPolicy{}
function Backup-MSSQL {}

# Automatically delete backups as per retention policies
function Cleanup-Backup {}

# Manage secure storage and retieval of passwords
function Get-SecurePass {}
function New-SecurePassFile {}

# Read in and process the config file
function Get-ConfigFile{}

# Get the histories of each backup
function Get-BackupFileHistory{}

# Automatically assign drive letters for rotating external hard drives
function Set-PartitionLetters{}

# Dynamically create HTML pages for the web dashbo
function Build-HtmlPages{}
function Build-IndexPage{}
function Build-HtmlContent{}

# Manage the TFTP server
function Start-TftpServer{}
function Stop-TftpServer{}

# Compare files for incremental backups
function Compare-Files{}

# Send alert emails upon backup failure
function Send-AlertEmail{}
```