D365 Daily Backup Script

# Contributors/Contacts

* pHil Rittenhouse ([phil.rittenhouse@microsoft.com](mailto:phil.rittenhouse@microsoft.com))
* Kevin Neely ([kevin.neely@microsoft.com](mailto:kevin.neely@microsoft.com))

## *Document Version 0.1*

# Purpose

Our goal was to build a system which would provide daily snapshots of a Dynamics 365 environment in daily flux as modifications were made continually by a development team. Such a system would guard against catastrophic tenant events and have the added benefit of speeding deployment to new tenants.

# Assumptions

For such a solution to be effective, the following conditions or their analogues should exist:

1. A git repository exists for the project, where resides the source code for any customizations (plugins, workflow actions, etc.).
2. The development team is storing their configuration changes (new/changed entities, workflow processes, etc.) in either their own solutions or in a central working solution, and not the Dynamics 365 Default solution.
3. A VM, local machine, or some other host environment exists to run the PowerShell script. (If this is not the case, alternate approaches may be used).

# Approach 1: PowerShell Script

The first approach centered on a PowerShell script and might best be applied in on-premise or IAAS scenarios.

The basis of this solution is the PowerShell script provided in this document. In our case, it was configured to run at a certain time each day on an Azure VM that was already running 24/7. A Windows Scheduler job was created on the VM using saved credentials that had appropriate permissions to both the D365 Organization and the git repo in the teams’ Azure DevOps instance.

Daily backups generated by this script were saved in the git repo under a folder structure according to Month and date.

## The Script

The PowerShell script outlines the actual process and is found below:

######################################################################

# #

# Script to backup daily git repo changes and D365 solution packages #

# Prepared by: #

# pHil Rittenhouse (philirit) #

# and #

# Kevin Neely (keneely) #

# #

######################################################################

# Allow and install the required modules

Set-ExecutionPolicy RemoteSigned -Scope CurrentUser

Import-Module Microsoft.Xrm.Data.Powershell

# Get today's date for use as the folder name

$date = Get-Date -Format MM-dd-yyyy

$month = Get-Date -Format MMMM

# Create the folder name from the date variable

$path = "C:\BlueSky\The%20Blue%20Sky%20Project\MCS.BSP.SolutionBackups\$month\$date"

$gitpath = "C:\BlueSky\The%20Blue%20Sky%20Project\"

# If today's folder already exists, delete it

If(Test-path $path) {Remove-item $path -Recurse}

# Create today's folder

New-Item -ItemType directory –Path $path

# Change working directory to new folder

Set-Location -Path $gitpath

# Pull from github

git pull

# Get the environment details

$CrmConTimeOutMinutes = 10

$CRMServerURL = "https://blueskyproj0.crm.dynamics.com"

# Build a connection string

$cs = "RequireNewInstance=True"

$cs+= ";Url=$CRMServerURL"

$cs+= ";AuthType=Office365"

$cs+= ";SkipDiscovery=True"

# REPLACE WITH YOUR USERNAME/PASSWORD

$cs+= ";Username=user@blueskyproj.onmicrosoft.com"

$cs+= ";Password=password"

# Create a connection using ConnectionString (for Online)

$Conn = Get-CrmConnection -ConnectionString $cs -MaxCrmConnectionTimeOutMinutes $CrmConTimeOutMinutes

# OR # Create a connection using ConnectionString (for On-Premise)

$Conn = Get-CrmConnection -ConnectionString "AuthType=AD;Url=https://myserver/Contoso;Domain=contosodom;UserName=user1;Password=password" $cs -MaxCrmConnectionTimeOutMinutes $CrmConTimeOutMinutes

# Publish ALL updated customizations:

Publish-CrmAllCustomization -conn $Conn -Verbose

# Export each of the solutions - APPEND TO THIS LIST IF ADDITIONAL SOLUTIONS ARE CREATED

Export-CrmSolution -SolutionName \_pHil -SolutionFilePath $path -SolutionZipFileName \_pHil.zip

Export-CrmSolution -SolutionName \_Matt -SolutionFilePath $path -SolutionZipFileName \_Matt.zip

Export-CrmSolution -SolutionName \_Kenneth -SolutionFilePath $path -SolutionZipFileName \_Kenneth.zip

Export-CrmSolution -SolutionName \_Drew -SolutionFilePath $path -SolutionZipFileName \_Drew.zip

Export-CrmSolution -SolutionName BlueSkyProj -SolutionFilePath $path -SolutionZipFileName BlueSkyMaster.zip

Export-CrmSolution -SolutionName BlueSKyMDA -SolutionFilePath $path -SolutionZipFileName BlueSkyMDA.zip

Export-CrmSolution -SolutionName \_Kevin -SolutionFilePath $path -SolutionZipFileName \_Kevin.zip

# Add changes

git add .

# Commit changes

git commit -m "Backup for $date"

# Push changes

git push

# To enable mobile push notifications signup at notify17.net

# Notify upon completion

$notifyParams = @{title="Solution Bakup Completion";content="Solutions have been backed up for $date"}

Invoke-WebRequest -Uri https://hook.notify17.net/api/raw/<api> -Method POST -Body $notifyParams

# DEPRECATED - use if want to collect all output into a single zip file

# Get the name/path for the zip archive

#$archive = "C:\BlueSky\Solutions\BlueSkySolutions\_$date.zip"

# If the archive file already exists, delete it

# If(Test-path $archive) {Remove-item $archive}

# Add-Type -assembly "system.io.compression.filesystem"

# Approach 2: Azure DevOps Pipeline

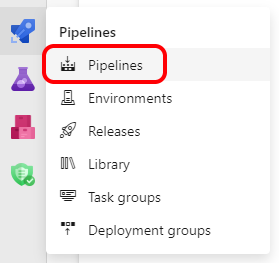
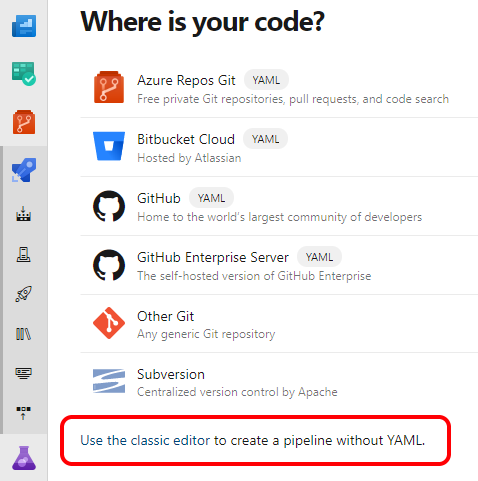
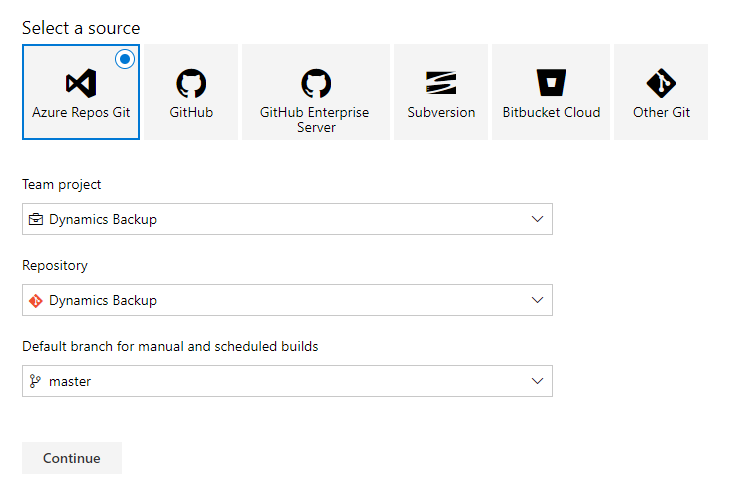
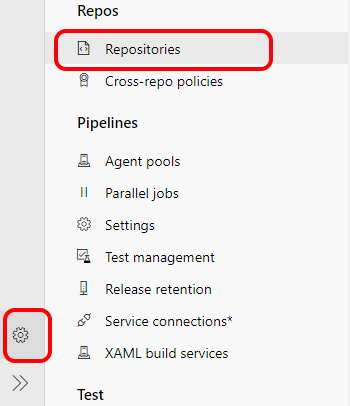
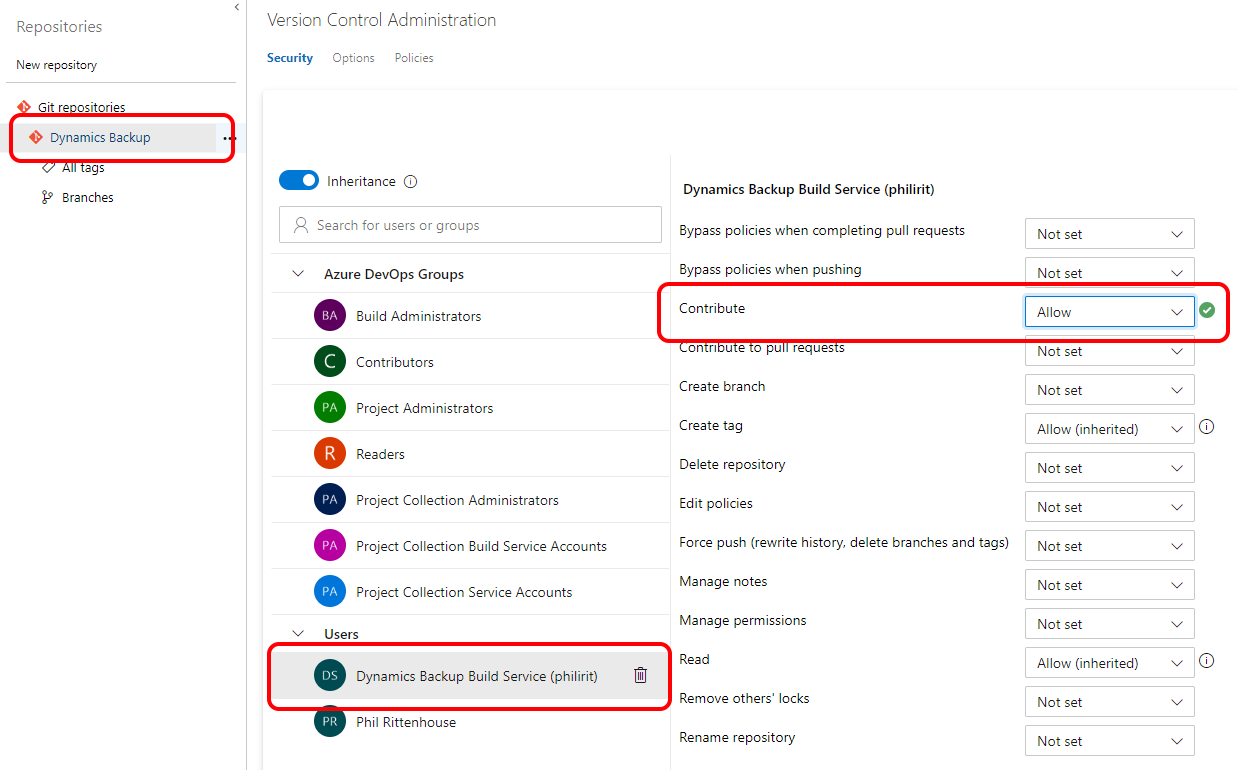
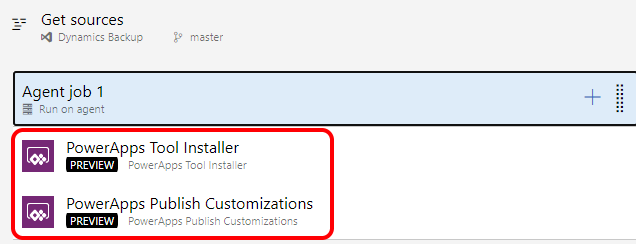
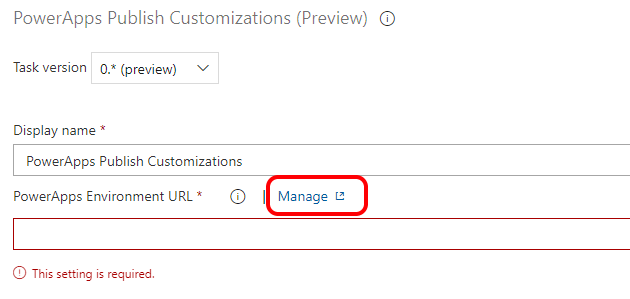
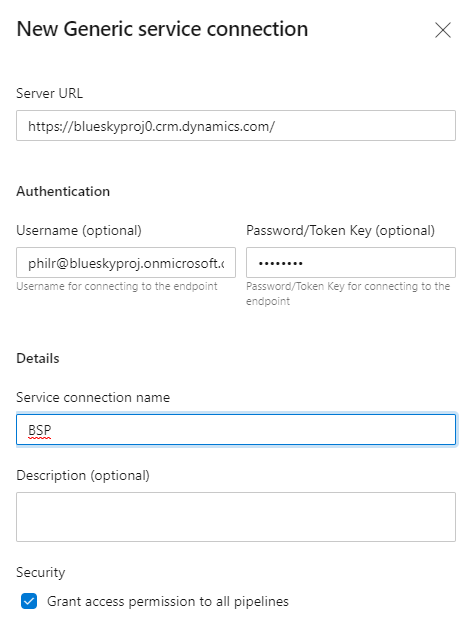
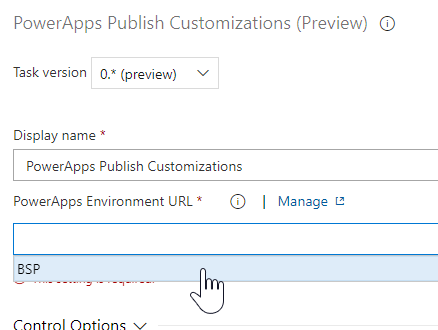
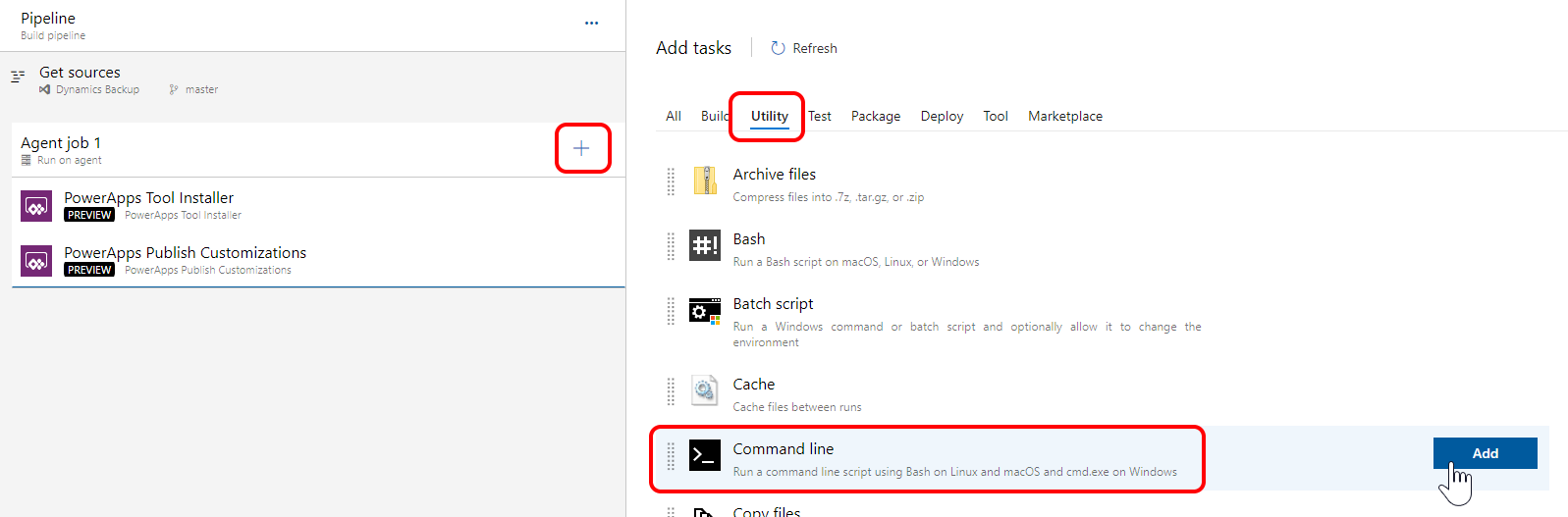
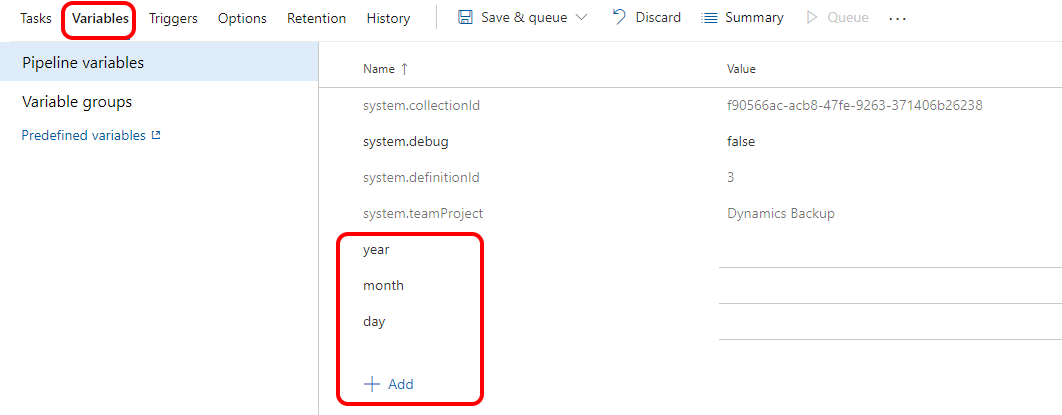
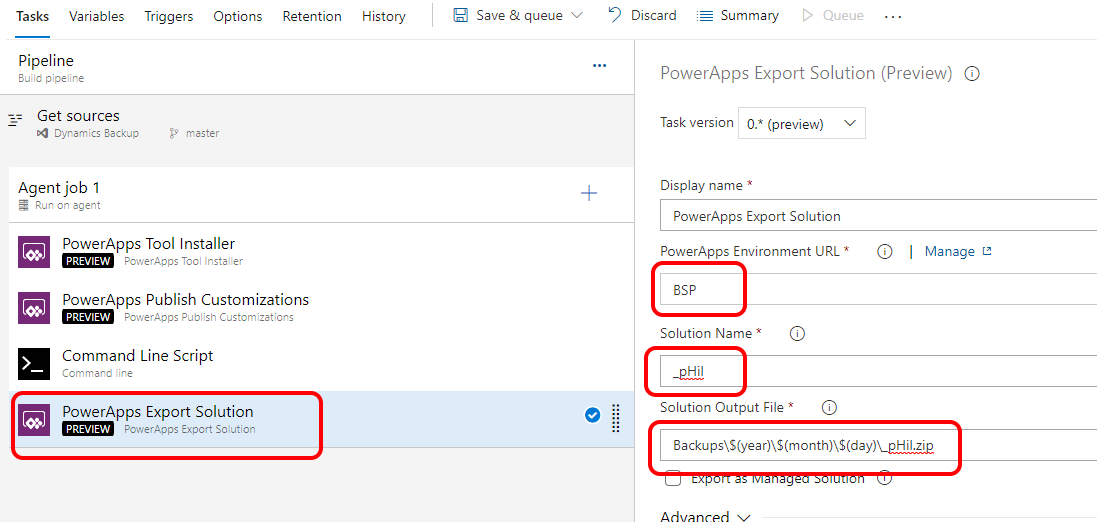
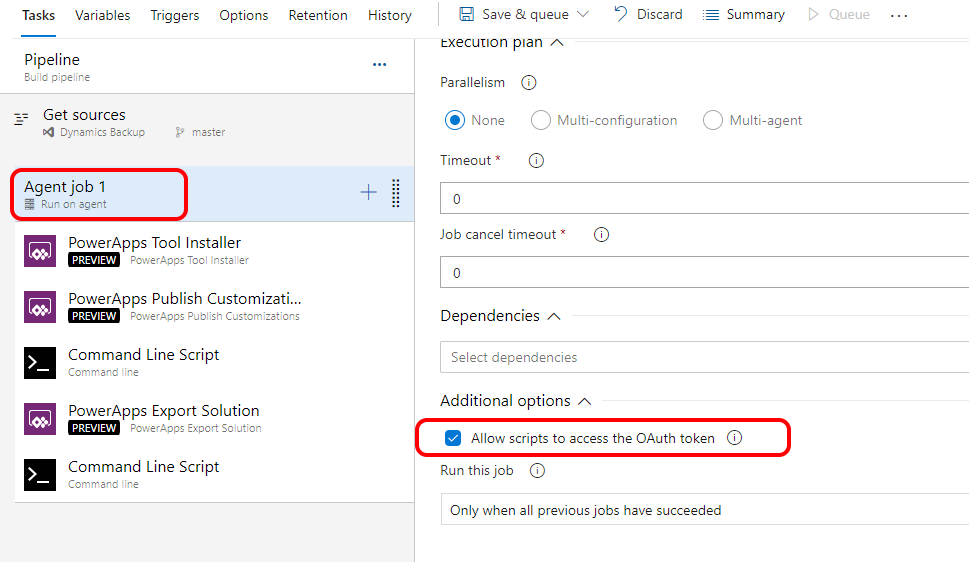
The PowerShell/on-prem/IAAS solution was effective and met the need, but we saw the advantage in leveraging newer technologies and rebuilt the solution as an Azure DevOps Pipeline.

## Prerequisites

The following components or configurations are required for this approach:

* A Dynamics Organization.
  + Appropriate access to publish customizations and export solution packages will be required.
  + As in the other approach, developers should be using named/specified solutions in Dynamics rather than customizing the Default Solution directly.
* Azure DevOps Organization, with the following configured:
  + A Repository for the project. A Git repo was used in our case. If Team Foundation Version Control is configured for your project, changes will need to be made in the scripts and configuration elements of the Azure Pipeline. These changes are out of scope of this version of the solution documentation.
  + Admin rights to the ADO Organization in order to build and configure the Pipeline.
  + [PowerApps Build Tools](https://marketplace.visualstudio.com/items?itemName=microsoft-IsvExpTools.PowerApps-BuildTools) will be used as a key component in the Pipeline.

## Building and Configuring the ADO Pipeline

1. Identify which solutions are to be backed up from Dynamics.
2. In Azure DevOps, select Pipelines from the left navigation:  
   
3. Create a new Pipeline.
4. Select “Use the classic editor” on the first screen:  
   
5. Select the appropriate options here and click Continue:  
   
6. Click Start with an Empty Job:  
   
7. Save & Queue the Pipeline. Make sure the Repo exists or the Job will fail.
8. Next, we must ensure the identity running the Pipeline has appropriate permissions. Click the Settings icon and then Repositories:  
   
9. Select the Repository to be backed up, then the User for the Build Service, and change the Contribute permission to “Allow”:  
   
10. Return to the Pipeline and enter Edit mode to start creating the steps.
11. The first two tasks needed are PowerApps Tool Installer and PowerApps Publish Customizations:  
    
12. To configure these steps, we must first configure the environment connection. Click “Manage” for the PowerApps Environment URL:  
    
13. Click on “Create Service Connection”.
14. Select “Generic” and click “Next”.
15. Configure the connection with your endpoint URL and credentials. Click Save when complete.  
    
16. Return to the Pipeline build and select the newly created [URL:](URL:)
17. Next, we will add the script to set up the dated folders. Click Add Task > Utility > Command Line:  
    
18. The script for this first Command Line step is as follows:  
    for /f "tokens=1-4 delims=/ " %%i in ("%date%") do (  
     set dow=%%i  
     set month=%%j  
     set day=%%k  
     set year=%%l  
    )  
    @echo ##vso[task.setvariable variable=year]%year%  
    @echo ##vso[task.setvariable variable=month]%month%  
    @echo ##vso[task.setvariable variable=day]%day%
19. Create three variables for the Pipeline: “year”, “month”, and “day”:  
    
20. Add and configure a PowerApps Export Solution task:  
    
21. Add another Command Line Script task to handle the Git commit, using the following script:  
    echo commit all changes  
    git config user.email philirit@microsoft.com  
    git config user.name "Automatic Build"  
    git checkout master  
    cd Backup  
    mkdir $(year)\$(month)\$(day)  
    cd $(year)\$(month)\$(day)  
    git add –all  
    git commit -m "solution init"  
    echo push code to new repo  
    git -c http.extraheader="AUTHORIZATION: bearer $(System.AccessToken)" push origin master
22. The last step is to ensure the Job can use your OAuth token. Click on the job name, and scroll down to enable the OAuth checkbox:  
    
23. Save & Queue, then Run the job.
24. Once the Job has completed, go back to your repo and verify that the backup folder and file(s) have been created:  
    