Thinking

Reading and analyzing PowerShell script file

from pathlib import Path, PurePosixPath, PureWindowsPath file\_path = Path("/mnt/data/setup-win-start.ps1") print(file\_path.exists()) print(file\_path)

True

/mnt/data/setup-win-start.ps1

Reading and displaying script file content

text = Path("/mnt/data/setup-win-start.ps1").read\_text(encoding="utf-8", errors="ignore") len(text), text[:5000]

(68249,

'<#\n setup-win-start.ps1\n Features:\n - Tokens: emit-cygwin-bootstrap, emit-wsl-setup\n - Archive switch: -Archive4Doc y|n (copies emissions into .4doc\_\* dirs and appends .{timestamp}.4doc to filenames)\n - Emits END\_OF\_DAY\_README\_{timestamp}.4doc.md at current working directory\n - Safe, no-install emission mode; writes completed\_install\_report even if .gitignore not present\n - .gitignore advisory (non-fatal): prints missing lines or entire block if file absent\n#>\n\n# ------------------------------------------------------------\n# TEMPORARY EXECUTION POLICY (SAFE FOR THIS SESSION ONLY)\n# ------------------------------------------------------------\n# Before running this setup script, open PowerShell \*\*as Administrator\*\*\n# and allow script execution only for the current session:\n#\n# Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass -Force\n#\n# This change is temporary and reverts automatically when PowerShell closes.\n# It does NOT modify your system-wide policy.\n#\n# Example full run (no interaction, archival enabled):\n#\n# cd "$HOME\\Desktop"\n# .\\setup-win-start.ps1 -Installs "emit-cygwin-bootstrap,emit-wsl-setup" -Interactive n -Archive4Doc y\n#\n# ------------------------------------------------------------\n\n[CmdletBinding()]\nparam(\n [string]$Installs,\n [ValidateSet(\'y\',\'n\'\n [ValidateSet("All","PreInitialUpdate","PostBeginningPreBigSetup","PostBigSetup",\n [switch]$EnableRestorePoints)][string]$RestorePointPhase = "All",\nfunction New-SafeRestorePoint {\n [CmdletBinding()]\n param([Parameter(Mandatory)][string]$Description)\n try {\n Enable-ComputerRestore -Drive "C:\\" -ErrorAction SilentlyContinue | Out-Null\n } catch {}\n try {\n Checkpoint-Computer -Description $Description -RestorePointType "MODIFY\_SETTINGS"\n Write-Host "[restore] Created: $Description"\n } catch {\n Write-Warning "[restore] Could not create restore point. $\_"\n }\n}\n\n [switch]$SkipRestorePoints)] [string]$Interactive = \'y\',\n [ValidateSet(\'y\',\'n\')] [string]$VerifyManualDownloads = \'y\',\n [ValidateSet(\'y\',\'n\')] [string]$RestorePoints = \'y\',\n [ValidateSet(\'y\',\'n\')] [string]$Archive4Doc = \'n\',\n [switch]$Help\n)\n\n# Output Help if requested. First, we define usage/help message.\n\nif ($Help) {\n # (print help and exit 0)\n exit 0\n}\n\nif ([string]::IsNullOrWhiteSpace($Installs)) {\n Write-Host "Error: -Installs is required. Try: -Installs `"emit-cygwin-bootstrap,emit-wsl-setup`"" -ForegroundColor Red\n Write-Host "Or get help/usage: .\\setup-win-start.ps1 -Help"\n exit 1\n}\n\n$ErrorActionPreference = \'Stop\'\n\nfunction NowStr {\n # Use Unix-style seconds, human date, and timezone offset automatically\n $raw = Get-Date -UFormat "%s\_%Y-%m-%dT%H%M%S%Z00"\n # Clean the fractional seconds PowerShell sometimes injects (e.g., .12345\_)\n return ($raw -replace \'[.][0-9]{1,5}\_\',\'\_\')\n}\n$startTimestamp = NowStr\n\n$desk = [Environment]::GetFolderPath(\'Desktop\')\n$cwd = (Get-Location).Path\n\n$logPartial = Join-Path $desk "partial\_install\_report\_$startTimestamp.log"\n$logComplete = Join-Path $desk "completed\_install\_report\_$startTimestamp.log"\n$logCurrent = $logPartial\n\n# collections\n$nextSteps = New-Object System.Collections.Generic.List[string]\n$doneList = New-Object System.Collections.Generic.List[string]\n$emittedDirs = New-Object System.Collections.Generic.List[string]\n\nfunction Log($s){ $s | Tee-Object -FilePath $logCurrent -Append | Out-Host }\nfunction Section($t){ Log "`n=== $t ===`n" }\n\n# Require elevation (admin)\n$admin = ([Security.Principal.WindowsPrincipal] [Security.Principal.WindowsIdentity]::GetCurrent()\n).IsInRole([Security.Principal.WindowsBuiltinRole]::Administrator)\nif (-not $admin) { Write-Host "Run this script in an elevated Windows PowerShell (Admin)." -ForegroundColor Yellow; exit 1 }\n\n# Parse tokens\n$want = @()\nif ($Installs.Trim().ToLower() -ne \'none\') {\n $want = $Installs.Split(\',\').ForEach({ $\_.Trim().ToLower() }) | Where-Object { $\_ -ne \'\' }\n}\n\nSection "Emission Plan"\nLog ("Requested tokens: " + (($want) -join ", "))\nLog ("Archive4Doc: " + $Archive4Doc)\n\n# ---------------- emit-cygwin-bootstrap ----------------\nif ($want -contains \'emit-cygwin-bootstrap\') {\n Section "Emit: Cygwin Bootstrap Pack"\n $outDir = Join-Path $desk ("Cygwin\_Bootstrap\_{0}" -f $startTimestamp)\n New-Item -ItemType Directory -Path $outDir -Force | Out-Null\n\n $bootstrap\_cmd = @"\n@echo off\nsetlocal enableextensions enabledelayedexpansion\nREM Cygwin Headless Bootstrap (verified, non-GUI)\nset CYG\_ROOT=C:\\cygwin64\nset CYG\_SETUP=%CYG\_ROOT%\\setup-x86\_64.exe\nset CYG\_SITE=https://mirrors.kernel.org/sourceware/cygwin/\nset PKGS=wget,curl,git,nano,vim,openssh,ca-certificates,tar,gzip,bzip2,unzip\nset CYG\_USER\_HOME=%CYG\_ROOT%\\home\\%USERNAME%\n\nwhere winget >nul 2>&1\nif %ERRORLEVEL%==0 (\n echo [\*] winget detected. Installing Cygwin silently...\n winget install --id=Cygwin.Cygwin -e --silent\n if %ERRORLEVEL%==0 ( goto SETUPRUN ) else ( echo [!] winget install failed; falling back. )\n)\n\n:DOWNLOAD\nif not exist "%CYG\_ROOT%" mkdi')

Reading and parsing PowerShell script file

*# Let's print more content readable* print(text[:2000])

<#

setup-win-start.ps1

Features:

- Tokens: emit-cygwin-bootstrap, emit-wsl-setup

- Archive switch: -Archive4Doc y|n (copies emissions into .4doc\_\* dirs and appends .{timestamp}.4doc to filenames)

- Emits END\_OF\_DAY\_README\_{timestamp}.4doc.md at current working directory

- Safe, no-install emission mode; writes completed\_install\_report even if .gitignore not present

- .gitignore advisory (non-fatal): prints missing lines or entire block if file absent

#>

# ------------------------------------------------------------

# TEMPORARY EXECUTION POLICY (SAFE FOR THIS SESSION ONLY)

# ------------------------------------------------------------

# Before running this setup script, open PowerShell \*\*as Administrator\*\*

# and allow script execution only for the current session:

#

# Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass -Force

#

# This change is temporary and reverts automatically when PowerShell closes.

# It does NOT modify your system-wide policy.

#

# Example full run (no interaction, archival enabled):

#

# cd "$HOME\Desktop"

# .\setup-win-start.ps1 -Installs "emit-cygwin-bootstrap,emit-wsl-setup" -Interactive n -Archive4Doc y

#

# ------------------------------------------------------------

[CmdletBinding()]

param(

[string]$Installs,

[ValidateSet('y','n'

[ValidateSet("All","PreInitialUpdate","PostBeginningPreBigSetup","PostBigSetup",

[switch]$EnableRestorePoints)][string]$RestorePointPhase = "All",

function New-SafeRestorePoint {

[CmdletBinding()]

param([Parameter(Mandatory)][string]$Description)

try {

Enable-ComputerRestore -Drive "C:\" -ErrorAction SilentlyContinue | Out-Null

} catch {}

try {

Checkpoint-Computer -Description $Description -RestorePointType "MODIFY\_SETTINGS"

Write-Host "[restore] Created: $Description"

} catch {

Write-Warning "[restore] Could not create restore point. $\_"

}

}

[switch]$SkipRestorePoints)] [string]$Interactive = 'y',

[ValidateSet('y','n')] [string]$Veri

Searching for parameter block in text

import re, pprint m = re.search(r'param\s\*\((.\*?)\)\s\*', text, re.S|re.I) print(bool(m)) print(text[m.start():m.start()+1000])

True

param(

[string]$Installs,

[ValidateSet('y','n'

[ValidateSet("All","PreInitialUpdate","PostBeginningPreBigSetup","PostBigSetup",

[switch]$EnableRestorePoints)][string]$RestorePointPhase = "All",

function New-SafeRestorePoint {

[CmdletBinding()]

param([Parameter(Mandatory)][string]$Description)

try {

Enable-ComputerRestore -Drive "C:\" -ErrorAction SilentlyContinue | Out-Null

} catch {}

try {

Checkpoint-Computer -Description $Description -RestorePointType "MODIFY\_SETTINGS"

Write-Host "[restore] Created: $Description"

} catch {

Write-Warning "[restore] Could not create restore point. $\_"

}

}

[switch]$SkipRestorePoints)] [string]$Interactive = 'y',

[ValidateSet('y','n')] [string]$VerifyManualDownloads = 'y',

[ValidateSet('y','n')] [string]$RestorePoints = 'y',

[ValidateSet('y','n')] [string]$Archive4Doc = 'n',

[switch]$Help

)

# Output Help if requested. First, we define usage/help message.

if ($Help) {

# (print help and exit 0)

exit 0

Finding function definitions and emit calls

emit\_sections = re.findall(r'(?im)function\s+([A-Za-z0-9\_-]+)\s\*\{', text) len(emit\_sections), emit\_sections[:50]

(25,

['New-SafeRestorePoint',

'NowStr',

'Get-SharedTimestamp',

'Resolve-DefaultBaseDir',

'Write-StringToFileUtf8',

'New-DocArchiveCopy',

'Write-CheckNewLinuxArtifacts',

'Write-PortableBashrc',

'Add-PortableProfileHelpers',

'Test-IsAdmin',

'Get-Timestamp',

'Get-IsoTimestampCompact',

'Get-IsoTimestampWeb',

'Get-IsoOnlyShort',

'Get-IsoOnlyLong',

'Use-SessionPolicy',

'Use-StrictPolicy',

'Use-UserRemoteSigned',

'sudo',

'sudo-pwsh',

'Set-Brightness',

'Dim-ForReading',

'Max-Brightness',

'Open-NightLightSettings',

'Set-Brightness'])

Searching for Write-CheckNewLinuxArtifacts function

def region(name): start = text.lower().find(f'function {name.lower()}') if start==-1: return None end = text.find('function ', start+1) return text[start:end] if end!=-1 else text[start:] print(region('Write-CheckNewLinuxArtifacts')[:2000])

function Write-CheckNewLinuxArtifacts {

[CmdletBinding()]

param([Parameter(Mandatory)][string[]]$DestRoots)

$scriptText = @"

#!/usr/bin/env bash

set -euo pipefail

TS="{" + r"$(date +'%s\_%Y-%m-%dT%H%M%S%z')" + r"}"

BACK="$HOME/.important\_backups"

OUT="${BACK}/new\_linux\_bash\_${TS}.out"

mkdir -p "$BACK"

{

echo "==== new\_linux\_bash snapshot @ $(date) ===="

echo "user: $USER host: $(hostname)"

echo

echo "== shell =="

echo "BASH\_VERSION=$BASH\_VERSION"

echo "SHELL=$SHELL"

echo "TTY=$(tty || true)"

echo

echo "== key env =="

printf "IFS=("; printf %q "$IFS"; printf ")

"

echo "PATH=$PATH"

echo "MANPATH=${MANPATH-}"

echo "PS1=${PS1-}"

echo "PROMPT\_COMMAND=${PROMPT\_COMMAND-}"

echo "TITLE\_STRING=${TITLE\_STRING-}"

echo

echo "== locale =="

locale || true

echo

echo "== versions =="

{ uname -a || true; } ; echo

{ lsb\_release -a 2>/dev/null || true; } ; echo

{ cat /etc/os-release 2>/dev/null || true; } ; echo

{ rpm -qa 2>/dev/null | wc -l | xargs echo "rpm\_count=" || true; }

{ dpkg -l 2>/dev/null | wc -l | xargs echo "dpkg\_count=" || true; }

echo

echo "== binaries =="

for b in bash sh zsh fish python3 python node npm git tree screen tmux brightnessctl xrandr; do

printf "%-14s" "$b"; command -v "$b" || echo "not found"

done

echo

echo "== dotfiles (backups) =="

} | tee "$OUT"

for f in .bashrc .profile .vimrc .gitconfig .bash\_profile .inputrc; do

[ -f "$HOME/$f" ] && cp -a "$HOME/$f" "$BACK/${f}.${TS}.bak" || true

done

syslist=(/etc/profile /etc/bashrc /etc/bash.bashrc /etc/environment /etc/manpath.config)

for s in "${syslist[@]}"; do

[ -r "$s" ] && cp -a "$s" "$BACK/$(basename "$s").${TS}.bak" || true

done

printf %s "$IFS" > "$BACK/IFS.${TS}.bak"

: "${PATH}" ; printf %s "$PATH" > "$BACK/PATH.${TS}.bak"

: "${MANPATH-}"; printf %s "${MANPATH-}" > "$BACK/MANPATH.${TS}.bak"

: "${PS1-}" ; printf %s "${PS1-}" > "$BACK/PS1.${TS}.bak"

: "${PROMPT\_COMMAND-}" ; printf %s "${PROMPT\_COMMAND-}" > "$BACK

Searching for Write-CheckNewLinuxArtifacts output files

print(region('Write-CheckNewLinuxArtifacts')[2000:4000])

/PROMPT\_COMMAND.${TS}.bak"

: "${TITLE\_STRING-}" ; printf %s "${TITLE\_STRING-}" > "$BACK/TITLE\_STRING.${TS}.bak"

echo "Snapshot written to: $OUT"

echo "Backups directory: $BACK"

"@

$readmeText = @"

check\_new\_linux\_bash — quick baseline capture (WSL, Fedora, Ubuntu, Cygwin)

What it does

• Saves a timestamped environment snapshot to ~/.important\_backups/new\_linux\_bash\_<epoch>\_<ISO>.out

• Backs up common dotfiles (.bashrc, .profile, .vimrc, .gitconfig, etc.) with the same timestamp.

• Backs up pre-dotfile system files when readable (/etc/profile, /etc/bashrc or /etc/bash.bashrc, /etc/environment, /etc/manpath.config).

• Saves single-value captures (IFS, PATH, MANPATH, PS1, PROMPT\_COMMAND, TITLE\_STRING) into separate .bak files.

How to run

1) Place both files in any shell environment (WSL, Fedora, Ubuntu, or Cygwin):

- check\_new\_linux\_bash.sh

- .portable.bashrc (optional, but nice!)

2) Make the script executable and run it:

chmod +x check\_new\_linux\_bash.sh

./check\_new\_linux\_bash.sh

3) Review outputs in ~/.important\_backups/

"@

foreach ($root in $DestRoots) {

$outDir = Join-Path $root "linux\_baseline"

New-Item -ItemType Directory -Force -Path $outDir | Out-Null

$sh = Join-Path $outDir "check\_new\_linux\_bash.sh"

$txt = Join-Path $outDir "check\_new\_linux\_bash.txt"

[System.IO.File]::WriteAllText($sh, $scriptText, [System.Text.Encoding]::UTF8)

[System.IO.File]::WriteAllText($txt, $readmeText, [System.Text.Encoding]::UTF8)

Write-Host "[emit] $sh"

Write-Host "[emit] $txt"

}

}

Displaying Write-PortableBashrc content

print(region('Write-PortableBashrc')[:2200])

function Write-PortableBashrc {

[CmdletBinding()]

param([Parameter(Mandatory)][string[]]$DestRoots)

$merged = @"

# ~/.portable.bashrc (portable core)

[ -f /etc/bash.bashrc ] && . /etc/bash.bashrc

# Quiet defaults that won't hurt

alias rmi='rm -i'

alias cpi='cp -i'

alias mvi='mv -i'

alias lessraw='less -r'

alias whence='type -a' # where, of a sort

alias dir='ls --color=auto --format=vertical'

alias vdir='ls --color=auto --format=long'

alias ll='ls -l' # long list

alias lh='ls -lah'

alias la='ls -A' # all but . and ..

alias l='ls -CF'

alias ls\_name\_and\_size='ls -Ss1pq --block-size=1'

alias runlog='runscriptreplayclean'

# Timestamp helpers, note ttdate and timestamp equivalent, muscle memory

dbldate() { date && date +'%s'; }

tripledate() { date && date +'%s' && date +'%s\_%Y-%m-%dT%H%M%S%z'; }

trpdate() { tripledate; }

ttdate() { date +'%s\_%Y-%m-%dT%H%M%S%z'; } # muscle memory

timestamp() { date +"%s\_%Y-%m-%dT%H%M%S%z"; }

##### PORTABLE ADDITIONS ###########################################

# ---- Portable functions and prompt setup brought from ---- #

# ---- ~/.bballdave025\_bash\_functions ---- #

# ---- Checks for exernal dependencies, other safe things, ---- #

# ---- still to be checked with ChatGPT. ---- #

## Try to import your

Extracting specific portion of text from file

print(region('Write-PortableBashrc')[2200:5000])

Evaluating Add-PortableProfileHelpers content output

print(region('Add-PortableProfileHelpers')[:1800])

function Add-PortableProfileHelpers {

[CmdletBinding()]

param([Parameter(Mandatory)][string]$PortableProfilePath)

$extra = @"

# portable\_profile.ps1

# A portable PowerShell profile you can dot-source from any host profile.

# Provides timestamp helpers, execution policy helpers, sudo, and screen-brightness utilities.

Extracting content for Add-PortableProfileHelpers function

print(region('Add-PortableProfileHelpers')[1800:4000])

Extracting options from param block text

*# Let's print more around param to see options list fully* start = text.lower().find('param(') print(text[start:start+2000])

param(

[string]$Installs,

[ValidateSet('y','n'

[ValidateSet("All","PreInitialUpdate","PostBeginningPreBigSetup","PostBigSetup",

[switch]$EnableRestorePoints)][string]$RestorePointPhase = "All",

function New-SafeRestorePoint {

[CmdletBinding()]

param([Parameter(Mandatory)][string]$Description)

try {

Enable-ComputerRestore -Drive "C:\" -ErrorAction SilentlyContinue | Out-Null

} catch {}

try {

Checkpoint-Computer -Description $Description -RestorePointType "MODIFY\_SETTINGS"

Write-Host "[restore] Created: $Description"

} catch {

Write-Warning "[restore] Could not create restore point. $\_"

}

}

[switch]$SkipRestorePoints)] [string]$Interactive = 'y',

[ValidateSet('y','n')] [string]$VerifyManualDownloads = 'y',

[ValidateSet('y','n')] [string]$RestorePoints = 'y',

[ValidateSet('y','n')] [string]$Archive4Doc = 'n',

[switch]$Help

)

# Output Help if requested. First, we define usage/help message.

if ($Help) {

# (print help and exit 0)

exit 0

}

if ([string]::IsNullOrWhiteSpace($Installs)) {

Write-Host "Error: -Installs is required. Try: -Installs `"emit-cygwin-bootstrap,emit-wsl-setup`"" -ForegroundColor Red

Write-Host "Or get help/usage: .\setup-win-start.ps1 -Help"

exit 1

}

$ErrorActionPreference = 'Stop'

function NowStr {

# Use Unix-style seconds, human date, and timezone offset automatically

$raw = Get-Date -UFormat "%s\_%Y-%m-%dT%H%M%S%Z00"

# Clean the fractional seconds PowerShell sometimes injects (e.g., .12345\_)

return ($raw -replace '[.][0-9]{1,5}\_','\_')

}

$startTimestamp = NowStr

$desk = [Environment]::GetFolderPath('Desktop')

$cwd = (Get-Location).Path

$logPartial = Join-Path $desk "partial\_install\_report\_$startTimestamp.log"

$logComplete = Join-Path $desk "completed\_install\_report\_$startTimestamp.log"

$logCurrent = $logPartial

# collections

$nextSteps = New-Object System.Collections.Generic.List[string]

$doneList = New-Object System.Collections.Generic.List[string]

$emitte

Extracting 'completed\_install\_report' and 'Archive4Doc' logs

re.findall(r'completed\_install\_report\_[^\s"]+', text)

['completed\_install\_report\_$startTimestamp.log',

'completed\_install\_report\_\*.log`',

'completed\_install\_report\_\*.log',

'completed\_install\_report\_$startTimestamp.log']

Extracting partial install report entries

re.findall(r'partial\_install\_report\_[^\s"]+', text)

['partial\_install\_report\_$startTimestamp.log', 'partial\_install\_report\_\*.log']

Searching for 'Archive4Doc' and 'completed\_install\_report'

*# Find 'Archive4Doc' uses* for m in re.finditer(r'Archive4Doc', text): print(m.start(), text[m.start()-200:m.start()+200])

112

1170 y your system-wide policy.

#

# Example full run (no interaction, archival enabled):

#

# cd "$HOME\Desktop"

# .\setup-win-start.ps1 -Installs "emit-cygwin-bootstrap,emit-wsl-setup" -Interactive n -Archive4Doc y

#

# ------------------------------------------------------------

[CmdletBinding()]

param(

[string]$Installs,

[ValidateSet('y','n'

[ValidateSet("All","PreInitialUpdate","PostBeginn

2114 SkipRestorePoints)] [string]$Interactive = 'y',

[ValidateSet('y','n')] [string]$VerifyManualDownloads = 'y',

[ValidateSet('y','n')] [string]$RestorePoints = 'y',

[ValidateSet('y','n')] [string]$Archive4Doc = 'n',

[switch]$Help

)

# Output Help if requested. First, we define usage/help message.

if ($Help) {

# (print help and exit 0)

exit 0

}

if ([string]::IsNullOrWhiteSpace($Installs)

4020 ToLower() -ne 'none') {

$want = $Installs.Split(',').ForEach({ $\_.Trim().ToLower() }) | Where-Object { $\_ -ne '' }

}

Section "Emission Plan"

Log ("Requested tokens: " + (($want) -join ", "))

Log ("Archive4Doc: " + $Archive4Doc)

# ---------------- emit-cygwin-bootstrap ----------------

if ($want -contains 'emit-cygwin-bootstrap') {

Section "Emit: Cygwin Bootstrap Pack"

$outDir = Join-Path $

4038 e') {

$want = $Installs.Split(',').ForEach({ $\_.Trim().ToLower() }) | Where-Object { $\_ -ne '' }

}

Section "Emission Plan"

Log ("Requested tokens: " + (($want) -join ", "))

Log ("Archive4Doc: " + $Archive4Doc)

# ---------------- emit-cygwin-bootstrap ----------------

if ($want -contains 'emit-cygwin-bootstrap') {

Section "Emit: Cygwin Bootstrap Pack"

$outDir = Join-Path $desk ("Cygwin\_Boot

14128 rs.Add($outDir) | Out-Null

$doneList.Add("Emitted WSL Companion Pack → $outDir") | Out-Null

$nextSteps.Add("Copy into your WSL home and run `bash wsl-setup.sh`.") | Out-Null

}

# ---------------- Archive4Doc logic ----------------

if ($Archive4Doc -eq 'y' -and $emittedDirs.Count -gt 0) {

Section "Archive4Doc"

foreach ($dir in $emittedDirs) {

$name = Split-Path $dir -Leaf

if ($name

14168 dd("Emitted WSL Companion Pack → $outDir") | Out-Null

$nextSteps.Add("Copy into your WSL home and run `bash wsl-setup.sh`.") | Out-Null

}

# ---------------- Archive4Doc logic ----------------

if ($Archive4Doc -eq 'y' -and $emittedDirs.Count -gt 0) {

Section "Archive4Doc"

foreach ($dir in $emittedDirs) {

$name = Split-Path $dir -Leaf

if ($name -like "Cygwin\_Bootstrap\_\*") {

$ad

14232 ps.Add("Copy into your WSL home and run `bash wsl-setup.sh`.") | Out-Null

}

# ---------------- Archive4Doc logic ----------------

if ($Archive4Doc -eq 'y' -and $emittedDirs.Count -gt 0) {

Section "Archive4Doc"

foreach ($dir in $emittedDirs) {

$name = Split-Path $dir -Leaf

if ($name -like "Cygwin\_Bootstrap\_\*") {

$ad = Join-Path (Split-Path $dir -Parent) (".4doc\_Cygwin\_{0}" -f $st

16441 ME\_{0}.4doc.md" -f $startTimestamp

$eodPath = Join-Path $cwd $eodName

$eodContent = @"

# End-of-Day Snapshot — $startTimestamp

\*\*What this includes\*\*

- Emission tokens run: `$(($want) -join ", ")`

- Archive4Doc: $Archive4Doc

- Emitted folders:

$(($emittedDirs | ForEach-Object { " - " + $\_ }) -join "`n")

- Logs written on Desktop:

- completed\_install\_report\_$startTimestamp.log

\*\*Git hygiene\*\*

16455 " -f $startTimestamp

$eodPath = Join-Path $cwd $eodName

$eodContent = @"

# End-of-Day Snapshot — $startTimestamp

\*\*What this includes\*\*

- Emission tokens run: `$(($want) -join ", ")`

- Archive4Doc: $Archive4Doc

- Emitted folders:

$(($emittedDirs | ForEach-Object { " - " + $\_ }) -join "`n")

- Logs written on Desktop:

- completed\_install\_report\_$startTimestamp.log

\*\*Git hygiene\*\*

This repo expe

65289 SilentlyContinue)) { $script:Installs = "" }

if (-not (Get-Variable -Name Interactive -Scope Script -ErrorAction SilentlyContinue)) { $script:Interactive = "n" }

if (-not (Get-Variable -Name Archive4Doc -Scope Script -ErrorAction SilentlyContinue)) { $script:Archive4Doc = "y" }

$actions = @()

if ($Installs) { $actions = $Installs.Split(",") | ForEach-Object { $\_.Trim().ToLowerInv

65357 able -Name Interactive -Scope Script -ErrorAction SilentlyContinue)) { $script:Interactive = "n" }

if (-not (Get-Variable -Name Archive4Doc -Scope Script -ErrorAction SilentlyContinue)) { $script:Archive4Doc = "y" }

$actions = @()

if ($Installs) { $actions = $Installs.Split(",") | ForEach-Object { $\_.Trim().ToLowerInvariant() } }

$doCygwin = $actions -contains "emit-cygwin-boo

66232 ItemType Directory -Force -Path $cygRoot | Out-Null

Write-PortableBashrc -DestRoots @($cygRoot)

if ($doArtifacts) { Write-CheckNewLinuxArtifacts -DestRoots @($cygRoot) }

if ($Archive4Doc -match '^(y|Y)$') {

$archDir = Join-Path $baseDir (".4doc\_Cygwin\_" + $sharedTS)

$vez = Join-Path $cygRoot "vezde\.portable.bashrc"

if (Test-Path $vez) { New-D

67093 ItemType Directory -Force -Path $wslRoot | Out-Null

Write-PortableBashrc -DestRoots @($wslRoot)

if ($doArtifacts) { Write-CheckNewLinuxArtifacts -DestRoots @($wslRoot) }

if ($Archive4Doc -match '^(y|Y)$') {

$archDir = Join-Path $baseDir (".4doc\_WSL\_" + $sharedTS)

$vez = Join-Path $wslRoot "vezde\.portable.bashrc"

if (Test-Path $vez) { New-DocA

Extracting Cygwin and WSL bootstrap strings

for pat in ['Cygwin\_Bootstrap','WSL\_Setup','linux\_baseline','portable\_profile.ps1']: for m in re.finditer(pat, text): print(pat, m.start(), text[m.start()-200:m.start()+200])

Cygwin\_Bootstrap 4227 4Doc: " + $Archive4Doc)

# ---------------- emit-cygwin-bootstrap ----------------

if ($want -contains 'emit-cygwin-bootstrap') {

Section "Emit: Cygwin Bootstrap Pack"

$outDir = Join-Path $desk ("Cygwin\_Bootstrap\_{0}" -f $startTimestamp)

New-Item -ItemType Directory -Path $outDir -Force | Out-Null

$bootstrap\_cmd = @"

@echo off

setlocal enableextensions enabledelayedexpansion

REM Cygwin He

Cygwin\_Bootstrap 14335 4Doc logic ----------------

if ($Archive4Doc -eq 'y' -and $emittedDirs.Count -gt 0) {

Section "Archive4Doc"

foreach ($dir in $emittedDirs) {

$name = Split-Path $dir -Leaf

if ($name -like "Cygwin\_Bootstrap\_\*") {

$ad = Join-Path (Split-Path $dir -Parent) (".4doc\_Cygwin\_{0}" -f $startTimestamp)

} elseif ($name -like "WSL\_Setup\_\*") {

$ad = Join-Path (Split-Path $dir -Parent

Cygwin\_Bootstrap 15114 es suffixed with .$startTimestamp.4doc)") | Out-Null

}

}

# ---------------- .gitignore advisory ----------------

Section ".gitignore advisory"

$giBlock = @"

# --- Bootstrap emissions (ignored) ---

Cygwin\_Bootstrap\_\*/

WSL\_Setup\_\*/

completed\_install\_report\_\*.log

partial\_install\_report\_\*.log

bootstrap\_cyg.cmd

cyg-bootstrap.sh

wsl-setup.sh

SHA256SUMS.txt

CYG\_README.md

WSL\_SETUP\_README.md

# --- All

Cygwin\_Bootstrap 65982 redTS = Get-SharedTimestamp

$baseDir = Resolve-DefaultBaseDir

$report = New-Object System.Collections.Generic.List[string]

if ($doCygwin) {

$cygRoot = Join-Path $baseDir ("Cygwin\_Bootstrap\_" + $sharedTS)

New-Item -ItemType Directory -Force -Path $cygRoot | Out-Null

Write-PortableBashrc -DestRoots @($cygRoot)

if ($doArtifacts) { Write-CheckNewLinu

WSL\_Setup 11785 er, run bootstrap\_cyg.cmd") | Out-Null

}

# ---------------- emit-wsl-setup ----------------

if ($want -contains 'emit-wsl-setup') {

Section "Emit: WSL Companion Pack"

$outDir = Join-Path $desk ("WSL\_Setup\_{0}" -f $startTimestamp)

New-Item -ItemType Directory -Path $outDir -Force | Out-Null

$wslbash = @'

#!/usr/bin/env bash

set -Eeuo pipefail

ORIG\_IFS="$IFS"; IFS=$'\n\t'

trap 'IFS="$ORIG\_

WSL\_Setup 13302 \*\*Fedora\*\* or \*\*Ubuntu\*\* WSL: base CLI tools and commented add-ons.

## Quick Start

1. Copy this directory into WSL (example):

```powershell

wsl -- cd ~ && cp -r /mnt/c/Users/<YourUser>/Desktop/WSL\_Setup\_\* .

```

2. Inside WSL, run:

```bash

bash wsl-setup.sh

```

3. (Optional) Open `wsl-setup.sh` and enable extras (Python, SSH, media tools).

## Verification & Next Steps

- Uses `dn

WSL\_Setup 14473 irs) {

$name = Split-Path $dir -Leaf

if ($name -like "Cygwin\_Bootstrap\_\*") {

$ad = Join-Path (Split-Path $dir -Parent) (".4doc\_Cygwin\_{0}" -f $startTimestamp)

} elseif ($name -like "WSL\_Setup\_\*") {

$ad = Join-Path (Split-Path $dir -Parent) (".4doc\_WSL\_{0}" -f $startTimestamp)

} else {

continue

}

Copy-Item -Recurse -Force $dir $ad

# append .{timestamp}.

WSL\_Setup 15134 tartTimestamp.4doc)") | Out-Null

}

}

# ---------------- .gitignore advisory ----------------

Section ".gitignore advisory"

$giBlock = @"

# --- Bootstrap emissions (ignored) ---

Cygwin\_Bootstrap\_\*/

WSL\_Setup\_\*/

completed\_install\_report\_\*.log

partial\_install\_report\_\*.log

bootstrap\_cyg.cmd

cyg-bootstrap.sh

wsl-setup.sh

SHA256SUMS.txt

CYG\_README.md

WSL\_SETUP\_README.md

# --- Allow explicit 4doc doc

WSL\_Setup 66850 ocArchiveCopy -SourceFile $\_.FullName -ArchiveDir $archDir -SharedTS $sharedTS } }

}

$report.Add("emit: $cygRoot")

}

if ($doWSL) {

$wslRoot = Join-Path $baseDir ("WSL\_Setup\_" + $sharedTS)

New-Item -ItemType Directory -Force -Path $wslRoot | Out-Null

Write-PortableBashrc -DestRoots @($wslRoot)

if ($doArtifacts) { Write-CheckNewLinuxArtifa

linux\_baseline 22343 it:

chmod +x check\_new\_linux\_bash.sh

./check\_new\_linux\_bash.sh

3) Review outputs in ~/.important\_backups/

"@

foreach ($root in $DestRoots) {

$outDir = Join-Path $root "linux\_baseline"

New-Item -ItemType Directory -Force -Path $outDir | Out-Null

$sh = Join-Path $outDir "check\_new\_linux\_bash.sh"

$txt = Join-Path $outDir "check\_new\_linux\_bash.tx

linux\_baseline 66542 $vez = Join-Path $cygRoot "vezde\.portable.bashrc"

if (Test-Path $vez) { New-DocArchiveCopy -SourceFile $vez -ArchiveDir $archDir -SharedTS $sharedTS }

$lb = Join-Path $cygRoot "linux\_baseline"

if (Test-Path $lb) { Get-ChildItem -LiteralPath $lb -File | ForEach-Object { New-DocArchiveCopy -SourceFile $\_.FullName -ArchiveDir $archDir -SharedTS $sharedTS } }

}

linux\_baseline 67400 $vez = Join-Path $wslRoot "vezde\.portable.bashrc"

if (Test-Path $vez) { New-DocArchiveCopy -SourceFile $vez -ArchiveDir $archDir -SharedTS $sharedTS }

$lb = Join-Path $wslRoot "linux\_baseline"

if (Test-Path $lb) { Get-ChildItem -LiteralPath $lb -File | ForEach-Object { New-DocArchiveCopy -SourceFile $\_.FullName -ArchiveDir $archDir -SharedTS $sharedTS } }

}

portable\_profile.ps1 59994 )

Write-Host "[emit] $out"

}

}

function Add-PortableProfileHelpers {

[CmdletBinding()]

param([Parameter(Mandatory)][string]$PortableProfilePath)

$extra = @"

# portable\_profile.ps1

# A portable PowerShell profile you can dot-source from any host profile.

# Provides timestamp helpers, execution policy helpers, sudo, and screen-brightness utilities.

function

portable\_profile.ps1 63918 eading { Set-Brightness -Percent 30 }

function Max-Brightness { Set-Brightness -Percent 100 }

Set-Alias ll Get-ChildItem

Set-Alias grep Select-String

Set-Alias which Get-Command

Write-Host ("Loaded portable\_profile.ps1 — " + (Get-Date).ToString("yyyy-MM-dd HH:mm:ss")) -ForegroundColor Gray

# ==== Night Light & brightness helpers =================================

function Open-NightLightSettin

portable\_profile.ps1 67732 e $\_.FullName -ArchiveDir $archDir -SharedTS $sharedTS } }

}

$report.Add("emit: $wslRoot")

}

# Emit patched portable PS profile at base

$ppath = Join-Path $baseDir "portable\_profile.ps1"

Add-PortableProfileHelpers -PortableProfilePath $ppath

$report.Add("emit: $ppath")

$log = Join-Path $baseDir ("completed\_install\_report\_" + $sharedTS + ".log")

Displaying surrounding area of emit-cygwin-bootstrap

start = text.find('# ---------------- emit-cygwin-bootstrap') print(text[start:start+2000])

# ---------------- emit-cygwin-bootstrap ----------------

if ($want -contains 'emit-cygwin-bootstrap') {

Section "Emit: Cygwin Bootstrap Pack"

$outDir = Join-Path $desk ("Cygwin\_Bootstrap\_{0}" -f $startTimestamp)

New-Item -ItemType Directory -Path $outDir -Force | Out-Null

$bootstrap\_cmd = @"

@echo off

setlocal enableextensions enabledelayedexpansion

REM Cygwin Headless Bootstrap (verified, non-GUI)

set CYG\_ROOT=C:\cygwin64

set CYG\_SETUP=%CYG\_ROOT%\setup-x86\_64.exe

set CYG\_SITE=https://mirrors.kernel.org/sourceware/cygwin/

set PKGS=wget,curl,git,nano,vim,openssh,ca-certificates,tar,gzip,bzip2,unzip

set CYG\_USER\_HOME=%CYG\_ROOT%\home\%USERNAME%

where winget >nul 2>&1

if %ERRORLEVEL%==0 (

echo [\*] winget detected. Installing Cygwin silently...

winget install --id=Cygwin.Cygwin -e --silent

if %ERRORLEVEL%==0 ( goto SETUPRUN ) else ( echo [!] winget install failed; falling back. )

)

:DOWNLOAD

if not exist "%CYG\_ROOT%" mkdir "%CYG\_ROOT%"

echo [\*] Downloading setup-x86\_64.exe and signature...

powershell -NoProfile -ExecutionPolicy Bypass -Command ^

"Invoke-WebRequest -UseBasicParsing https://www.cygwin.com/setup-x86\_64.exe -OutFile '%CYG\_SETUP%'; ^

Invoke-WebRequest -UseBasicParsing https://www.cygwin.com/setup-x86\_64.exe.sig -OutFile '%CYG\_SETUP%.sig'"

where gpg >nul 2>&1

if %ERRORLEVEL% NEQ 0 ( echo [!] gpg not found; cannot verify; goto SETUPRUN )

echo [\*] Scraping official fingerprint from cygwin.com/install.html ...

for /f "usebackq tokens=\* delims=" %%F in (`powershell -NoProfile -ExecutionPolicy Bypass -Command ^

"$p=(Invoke-WebRequest -UseBasicParsing https://www.cygwin.com/install.html).Content; ^

$m=[regex]::Match($p,'Primary key fingerprint:\s\*([0-9A-F]{{4}}(?:\s+[0-9A-F]{{4}}){{9}})','IgnoreCase'); ^

if(-not $m.Success){{exit 1}}; $m.Groups[1].Value.ToUpper()"`) do set CYG\_FP\_WEB=%%F

if "%CYG\_FP\_WEB%"=="" ( echo [!] Could not scrape fingerprint; aborting. & exit /b 1 )

echo [\*] Locating signing key via WKD/keyserver...

gpg --bat

Searching for New-Item file creation commands

for m in re.finditer(r'New-Item\s+-ItemType\s+File\s+-Path\s+([^\r\n]+)', text): print(text[m.start():m.start()+200])

Searching for "emit" lines in text

*# Search for [emit] lines* for m in re.finditer(r'\[emit\]\s+([^\r\n]+)', text): print(text[m.start():m.start()+200])

[emit] $sh"

Write-Host "[emit] $txt"

}

}

function Write-PortableBashrc {

[CmdletBinding()]

param([Parameter(Mandatory)][string[]]$DestRoots)

$merged = @"

# ~/

[emit] $txt"

}

}

function Write-PortableBashrc {

[CmdletBinding()]

param([Parameter(Mandatory)][string[]]$DestRoots)

$merged = @"

# ~/.portable.bashrc (portable core)

[emit] $out"

}

}

function Add-PortableProfileHelpers {

[CmdletBinding()]

param([Parameter(Mandatory)][string]$PortableProfilePath)

$extra = @"

# portable\_profile.ps1

#

Analyzing detected strings

The regex identified relevant strings, but not specific filenames. It looks like the Cygwin\_Bootstrap folder contains a batch script for setting up Cygwin, possibly including a README and a subfolder with bash scripts. Similarly, WSL\_Setup might have similar content. I'll continue searching to confirm.

Extracting WSL setup file information

start = text.find('# ---------------- emit-wsl-setup') print(text[start:start+2000])

# ---------------- emit-wsl-setup ----------------

if ($want -contains 'emit-wsl-setup') {

Section "Emit: WSL Companion Pack"

$outDir = Join-Path $desk ("WSL\_Setup\_{0}" -f $startTimestamp)

New-Item -ItemType Directory -Path $outDir -Force | Out-Null

$wslbash = @'

#!/usr/bin/env bash

set -Eeuo pipefail

ORIG\_IFS="$IFS"; IFS=$'\n\t'

trap 'IFS="$ORIG\_IFS"' EXIT

if command -v dnf >/dev/null 2>&1; then

PKG="sudo dnf -y install"; UPDATE="sudo dnf -y update"; DISTRO="fedora"

elif command -v apt-get >/dev/null 2>&1; then

PKG="sudo apt-get -y install"; UPDATE="sudo apt-get update -y"; DISTRO="ubuntu"

else

echo "Unsupported WSL distro (need dnf or apt-get)."; exit 1

fi

echo "WSL companion running on: $DISTRO"

$UPDATE

$PKG git curl wget zip unzip tar ca-certificates

# Optional examples:

# $PKG imagemagick vlc audacity

# $PKG python3 python3-pip

# if [ "$DISTRO" = "fedora" ]; then

# $PKG openssh-server && sudo systemctl enable --now sshd

# else

# $PKG openssh-server && sudo service ssh start

# fi

echo "Base packages installed. Customize this script as needed."

'@

$wsl\_readme = @"

# WSL Companion Setup — README

\*\*Emitted by:\*\* setup-win-start.ps1

\*\*Timestamp:\*\* $startTimestamp

This directory is automatically generated by the Windows bootstrap.

Make sure you've run \*\*setup-win-start.ps1\*\* first — it performs prechecks, logging, and may emit other helpers.

## Purpose

`wsl-setup.sh` configures a minimal, signed-package baseline inside \*\*Fedora\*\* or \*\*Ubuntu\*\* WSL: base CLI tools and commented add-ons.

## Quick Start

1. Copy this directory into WSL (example):

```powershell

wsl -- cd ~ && cp -r /mnt/c/Users/<YourUser>/Desktop/WSL\_Setup\_\* .

```

2. Inside WSL, run:

```bash

bash wsl-setup.sh

```

3. (Optional) Open `wsl-setup.sh` and enable extras (Python, SSH, media tools).

## Verification & Next Steps

- Uses `dnf`/`apt` official repos (GPG-verified).

- Integrate Docker/conda/etc. after this baseline.

- See the desktop `completed\_i

Organizing script details

I’m focusing on the "completed\_install\_report" mention. To move forward, I’ll search for the 'Archive4Doc' flow earlier. I need to create 5 lists, starting with one that details emitted scripts, READMEs, and instructions. The Cygwin Bootstrap pack seems to have a bootstrap .cmd. I need to figure out how files are saved.

Searching for emitted scripts and README files

*# Search for WriteAllText writing $bootstrap\_cmd and README etc.* for m in re.finditer(r'WriteAllText\(([^)]+)\)', text): print(text[m.start():m.start()+180])

WriteAllText($Path, $Body, [System.Text.Encoding]::UTF8)

}

function New-DocArchiveCopy {

param(

[Parameter(Mandatory)][string]$SourceFile,

[Parameter

WriteAllText($sh, $scriptText, [System.Text.Encoding]::UTF8)

[System.IO.File]::WriteAllText($txt, $readmeText, [System.Text.Encoding]::UTF8)

Write-Host "[emit] $sh"

WriteAllText($txt, $readmeText, [System.Text.Encoding]::UTF8)

Write-Host "[emit] $sh"

Write-Host "[emit] $txt"

}

}

function Write-PortableBashrc {

WriteAllText($out, $merged, [System.Text.Encoding]::UTF8)

Write-Host "[emit] $out"

}

}

function Add-PortableProfileHelpers {

[CmdletBinding()]

pa

Inspecting helper functions

There are helper functions named Write-StringToFileUtf8 and New-DocArchiveCopy. I should take a closer look at both to understand what they are doing. This could help clarify how files are being saved and may provide insight into the process I’m investigating. Let's dive into these functions.

Inspecting Write-StringToFileUtf8 function

print(region('Write-StringToFileUtf8')[:1000])

function Write-StringToFileUtf8 {

param([string]$Path,[string]$Body)

$dir = Split-Path -Parent $Path

if ($dir -and -not (Test-Path -LiteralPath $dir)) { New-Item -ItemType Directory -Force -Path $dir | Out-Null }

[System.IO.File]::WriteAllText($Path, $Body, [System.Text.Encoding]::UTF8)

}

Inspecting New-DocArchiveCopy function

print(region('New-DocArchiveCopy')[:1200])

function New-DocArchiveCopy {

param(

[Parameter(Mandatory)][string]$SourceFile,

[Parameter(Mandatory)][string]$ArchiveDir,

[Parameter(Mandatory)][string]$SharedTS

)

New-Item -ItemType Directory -Force -Path $ArchiveDir | Out-Null

$name = [System.IO.Path]::GetFileName($SourceFile)

$archName = "$name.$SharedTS.4doc"

Copy-Item -LiteralPath $SourceFile -Destination (Join-Path $ArchiveDir $archName) -Force

}

Searching for parameter flags

We need a list of options and their values. The parameter block seems to be malformed, so I need to find recognized flags in the code, such as -Installs, -Interactive, -Archive4Doc, -EnableRestorePoints, or -RestorePoints. I'll start by searching for how "$Interactive" is used.

Searching for parameter flag usage

for var in ['Installs','Interactive','Archive4Doc','EnableRestorePoints','RestorePointPhase','DryRun','VerifyManualDownloads','RestorePoints','SkipRestorePoints']: print(var, '->', var in text)

Installs -> True

Interactive -> True

Archive4Doc -> True

EnableRestorePoints -> True

RestorePointPhase -> True

DryRun -> False

VerifyManualDownloads -> True

RestorePoints -> True

SkipRestorePoints -> True

Checking DryRun and Install tokens

There's no DryRun option present, so I should mention that it's not implemented or scaffolded. Now, I need to find the list of accepted install tokens by looking into the parsing for the "$want" variable. Let me search for how "-Installs" is being parsed in the code.

Identifying missing DryRun implementation and parsing tokens

print(re.findall(r'\$want\s\*=\s\*.+', text)[0])

$want = @()

Finding installs parsing tokens in code

*# Find where want is populated* for m in re.finditer(r'want\s\*=.\*', text): print(text[m.start()-100:m.end()+200])

ript in an elevated Windows PowerShell (Admin)." -ForegroundColor Yellow; exit 1 }

# Parse tokens

$want = @()

if ($Installs.Trim().ToLower() -ne 'none') {

$want = $Installs.Split(',').ForEach({ $\_.Trim().ToLower() }) | Where-Object { $\_ -ne '' }

}

Section "Emission Plan"

Log ("Requested tokens: " + (($wan

dColor Yellow; exit 1 }

# Parse tokens

$want = @()

if ($Installs.Trim().ToLower() -ne 'none') {

$want = $Installs.Split(',').ForEach({ $\_.Trim().ToLower() }) | Where-Object { $\_ -ne '' }

}

Section "Emission Plan"

Log ("Requested tokens: " + (($want) -join ", "))

Log ("Archive4Doc: " + $Archive4Doc)

# ---------------- emit-cygwin-bootstrap ----------------

if ($want -contains 'emit-

Looking for tokens in conditions

I need to find the tokens referenced in conditions, so I’ll search for "if ($want -contains '" in the code. This should help me spot any relevant tokens being checked against the "$want" variable. Let’s see what turns up in that search.

Finding tokens in conditional statements

tokens=set() for m in re.finditer(r"\$want\s+-contains\s+'([^']+)'", text): tokens.add(m.group(1)) tokens

{'emit-cygwin-bootstrap', 'emit-wsl-setup'}

Reporting on missing tokens and options

The 'emit-wsl-cygwin-artifacts' token is missing, meaning it's not integrated, so that’s something to report. As for the options flags, I need to search for param items. The parameter block is a bit messy, but later code does refer to several flags like $Interactive, $VerifyManualDownloads, and others. I need to understand how they’re used.

Searching for parameter usage in code

for var in ['Interactive','VerifyManualDownloads','RestorePoints','EnableRestorePoints','RestorePointPhase','SkipRestorePoints']: for m in re.finditer(r'\$'+var+r'\b.\*', text): line = text[m.start(): text.find('\n', m.start())] print(var, '::', line[:200])

Interactive :: $Interactive = 'y',

VerifyManualDownloads :: $VerifyManualDownloads = 'y',

RestorePoints :: $RestorePoints = 'y',

EnableRestorePoints :: $EnableRestorePoints)][string]$RestorePointPhase = "All",

RestorePointPhase :: $RestorePointPhase = "All",

SkipRestorePoints :: $SkipRestorePoints)] [string]$Interactive = 'y',

Looking for usage examples

Okay, the flags are present in the parameter area, but I need to search for where these are actually used in the code. That’s where I’ll see how they’re being applied, and I can dig deeper into how they influence functionality—or if they even get used at all.

Searching for runtime variable usage

for var in ['Interactive','VerifyManualDownloads','RestorePoints','EnableRestorePoints','RestorePointPhase','SkipRestorePoints']: found=False for m in re.finditer(r'\$'+var+r'\b', text): if m.start()<2000: continue ctx = text[m.start()-60:m.start()+120] if 'param' in ctx[:120]: continue print(var, '::', ctx.replace('\n',' ')[:180]) found=True break if not found: print(var, ':: no runtime usage found')

Interactive :: no runtime usage found

VerifyManualDownloads :: no runtime usage found

RestorePoints :: rifyManualDownloads = 'y', [ValidateSet('y','n')] [string]$RestorePoints = 'y', [ValidateSet('y','n')] [string]$Archive4Doc = 'n', [switch]$Help ) # Output Help if requested

EnableRestorePoints :: no runtime usage found

RestorePointPhase :: no runtime usage found

SkipRestorePoints :: no runtime usage found

Exploring restore point calls

Hmm, it looks like restore points might be implemented as direct calls later on, without being controlled by flags. To check this, I’ll search for any instances of the New-SafeRestorePoint call. That should give me a better idea of how these points are actually handled in the code.

Searching for New-SafeRestorePoint calls

list(re.findall(r'New-SafeRestorePoint\s+"([^"]+)"', text))

[]

Investigating unused restore point function

It looks like the restore point function exists, but isn’t used yet. That’s curious. Next, I’ll search for any references to the “Section” function. Hopefully, it’ll shed some light on why this function might be defined but not actively used at this stage.

Searching for "Section" function usage

print(re.findall(r'function\s+Section', text)[:1])

['function Section']

Searching for "Section" function

print(region('Section')[:600])

function Section($t){ Log "`n=== $t ===`n" }

# Require elevation (admin)

$admin = ([Security.Principal.WindowsPrincipal] [Security.Principal.WindowsIdentity]::GetCurrent()

).IsInRole([Security.Principal.WindowsBuiltinRole]::Administrator)

if (-not $admin) { Write-Host "Run this script in an elevated Windows PowerShell (Admin)." -ForegroundColor Yellow; exit 1 }

# Parse tokens

$want = @()

if ($Installs.Trim().ToLower() -ne 'none') {

$want = $Installs.Split(',').ForEach({ $\_.Trim().ToLower() }) | Where-Object { $\_ -ne '' }

}

Section "Emission Plan"

Log ("Requested tokens: " + (($want) -join

Exploring report output usage

I need to check how the report is being built, specifically looking for references to $report. Searching for any usage of the completed\_install\_report function or related logs will help clarify how the report is being constructed and if there are any relevant outputs I need to review.

Searching for completed\_install\_report usage

print(re.findall(r'\$report', text)[:10])

['$report', '$report', '$report', '$report', '$report', '$report', '$report']

Finding completed\_install\_report usage in text

locs=[m.start() for m in re.finditer(r'\$report', text)] for s in locs[:5]: print(text[s-200:s+300])

"emit-wsl-cywin-artifacts") -or ($actions -contains "emit-wsl-cywin-artifacts")

if ($doCygwin -or $doWSL) {

$sharedTS = Get-SharedTimestamp

$baseDir = Resolve-DefaultBaseDir

$report = New-Object System.Collections.Generic.List[string]

if ($doCygwin) {

$cygRoot = Join-Path $baseDir ("Cygwin\_Bootstrap\_" + $sharedTS)

New-Item -ItemType Directory -Force -Path $cygRoot | Out-Null

Write-PortableBashrc -DestRoots @($cygRoot)

if ($doArtifac

eline"

if (Test-Path $lb) { Get-ChildItem -LiteralPath $lb -File | ForEach-Object { New-DocArchiveCopy -SourceFile $\_.FullName -ArchiveDir $archDir -SharedTS $sharedTS } }

}

$report.Add("emit: $cygRoot")

}

if ($doWSL) {

$wslRoot = Join-Path $baseDir ("WSL\_Setup\_" + $sharedTS)

New-Item -ItemType Directory -Force -Path $wslRoot | Out-Null

Write-PortableBashrc -DestRoots @($wslRoot)

if ($doArtifacts) { Write-CheckNewLinuxArtifac

eline"

if (Test-Path $lb) { Get-ChildItem -LiteralPath $lb -File | ForEach-Object { New-DocArchiveCopy -SourceFile $\_.FullName -ArchiveDir $archDir -SharedTS $sharedTS } }

}

$report.Add("emit: $wslRoot")

}

# Emit patched portable PS profile at base

$ppath = Join-Path $baseDir "portable\_profile.ps1"

Add-PortableProfileHelpers -PortableProfilePath $ppath

$report.Add("emit: $ppath")

$log = Join-Path $baseDir ("completed\_install\_report\_

"emit: $wslRoot")

}

# Emit patched portable PS profile at base

$ppath = Join-Path $baseDir "portable\_profile.ps1"

Add-PortableProfileHelpers -PortableProfilePath $ppath

$report.Add("emit: $ppath")

$log = Join-Path $baseDir ("completed\_install\_report\_" + $sharedTS + ".log")

$report.Insert(0, "shared\_ts: " + $sharedTS)

$report.Insert(1, "base\_dir: " + $baseDir)

[System.IO.File]::WriteAllLines($log, $report, [System.Text.Encoding]::UTF8)

e\_profile.ps1"

Add-PortableProfileHelpers -PortableProfilePath $ppath

$report.Add("emit: $ppath")

$log = Join-Path $baseDir ("completed\_install\_report\_" + $sharedTS + ".log")

$report.Insert(0, "shared\_ts: " + $sharedTS)

$report.Insert(1, "base\_dir: " + $baseDir)

[System.IO.File]::WriteAllLines($log, $report, [System.Text.Encoding]::UTF8)

Write-Host "Done (emit). See report: $log"

}

# =============================================================

Analyzing artifact logic

The code logic includes variables for handling doCygwin, doWSL, and doArtifacts functions. Interestingly, it accepts the misspelled ‘cywin’ instead of ‘cygwin’. We’ll report this mismatch. I need to check which outputs are generated by this, including the files like cygwin\_bootstrap.cmd and others.

Extracting output file paths from script

*# Find where $bootstrap\_cmd is written to a file name:* for m in re.finditer(r'\$bootstrap\_cmd\s\*=\s\*@', text): sec\_start = m.start() *# search after for a file path including .cmd* s = text[sec\_start:sec\_start+4000] m2 = re.search(r'Join-Path\s+\$outDir\s+"([^"]+)"', s) print(m2.group(1) if m2 else None)

None

Extracting output filenames from code block

*# Search for Join-Path $outDir occurrences to find file names* names=set() for m in re.finditer(r'Join-Path\s+\$outDir\s+"([^"]+)"', text): names.add(m.group(1)) names

{'CYG\_README.md',

'SHA256SUMS.txt',

'WSL\_SETUP\_README.md',

'bootstrap\_cyg.cmd',

'check\_new\_linux\_bash.sh',

'check\_new\_linux\_bash.txt',

'cyg-bootstrap.sh',

'wsl-setup.sh'}

Listing output files

The emitted files in the outDir include names like wsl-setup.sh, WSL\_SETUP\_README.md, cyg-bootstrap.sh, bootstrap\_cyg.cmd, and CYG\_README.md. It seems like linux\_baseline might be managed earlier and could be saved in multiple locations. Additionally, I spotted vezde\.portable.bashrc in a path—let's search for ‘vezde’ to investigate further.

Searching for vezde portable bashrc path

re.findall(r'vezde[\\/]\.portable\.bashrc', text)

['vezde/.portable.bashrc',

'vezde/.portable.bashrc',

'vezde\\.portable.bashrc',

'vezde\\.portable.bashrc']

Searching for 'vezde' occurrences in script

*# get context to see where created* for m in re.finditer(r"Join-Path\s+\$[^ ]+\s+\"vezde[\\/]\\?\.portable\.bashrc\"", text): print(text[m.start()-200:m.start()+200])

Artifacts) { Write-CheckNewLinuxArtifacts -DestRoots @($cygRoot) }

if ($Archive4Doc -match '^(y|Y