```powershell
# ============================================================
# Helpers
# ============================================================
function Get-Msg {
    param(
        [string]$FR,
        [string]$EN,
        [bool]$IsFrench = $true      # French by default
    )
    if ($IsFrench) { return $FR } else { return $EN }
}
# ============================================================
# Network
# ============================================================
function Remove-AllNetworkDrives {
    $network = New-Object -ComObject "WScript.Network"
    $drives  = $network.EnumNetworkDrives()
    $count   = $drives.Count()
    if ($count -eq 0) { return }
    # Capture all drive letters BEFORE removing anything
    $driveList = @()
    for ($i = 0; $i -lt $count; $i += 2) {
        $driveList += $drives.Item($i)
    }
    foreach ($drive in $driveList) {
        try {
            $network.RemoveNetworkDrive($drive, $true, $true)
        } catch {
            # Silently ignore — drive may already be disconnected
        }
    }
}
function Wait-NetworkReady {
    param(
        [string]$UNCPath,
        [int]$TimeoutSeconds  = 30,
        [int]$RetryInterval   = 2
    )
    $deadline = (Get-Date).AddSeconds($TimeoutSeconds)
    while ((Get-Date) -lt $deadline) {
        if (Test-Path $UNCPath -ErrorAction SilentlyContinue) {
            return $true
        }
        Start-Sleep -Seconds $RetryInterval
    }
    return $false
}
function Connect-Drive {
    param(
        [string]$Drive,
        [string]$Path,
        [string]$Description,
        [bool]$IsFrench        = $true,
        [int]$NetworkTimeout   = 30,    # seconds to wait for UNC path
        [int]$RetryInterval    = 2      # seconds between retries
    )
    # Wait until the UNC share is reachable before attempting to map
    $ready = Wait-NetworkReady -UNCPath $Path `
                               -TimeoutSeconds $NetworkTimeout `
                               -RetryInterval  $RetryInterval
    if (-not $ready) {
        $title = Get-Msg -FR "Erreur réseau" -EN "Network error" -IsFrench $IsFrench
        $msg   = Get-Msg `
            -FR "Le partage réseau est inaccessible après ${NetworkTimeout}s :`n$Path" `
            -EN "Network share unreachable after ${NetworkTimeout}s:`n$Path" `
            -IsFrench $IsFrench
        [System.Windows.Forms.MessageBox]::Show($msg, $title,
            [System.Windows.Forms.MessageBoxButtons]::OK,
            [System.Windows.Forms.MessageBoxIcon]::Warning)
        return
    }
     $network = New-Object -ComObject "WScript.Network"
    try {
        $network.MapNetworkDrive($Drive, $Path)
        $shell = New-Object -ComObject "Shell.Application"
        $ns    = $shell.Namespace("$Drive\")
        if ($null -ne $ns) {
            $ns.Self.Name = $Description
        }
    } catch {
```

```
        $title = Get-Msg -FR "Erreur" -EN "Error" -IsFrench $IsFrench
        $msg   = Get-Msg `
            -FR "Connexion lecteur $Drive ($Path)`n$($_.Exception.Message)" `
            -EN "Connecting drive $Drive ($Path)`n$($_.Exception.Message)" `
            -IsFrench $IsFrench
        [System.Windows.Forms.MessageBox]::Show($msg, $title,
            [System.Windows.Forms.MessageBoxButtons]::OK,
            [System.Windows.Forms.MessageBoxIcon]::Error)
    }
}
function Resolve-NetworkDrive {
    param([string]$DriveLetter)
    $network = New-Object -ComObject "WScript.Network"
    $drives  = $network.EnumNetworkDrives()
    for ($i = 0; $i -lt $drives.Count(); $i += 2) {
        if ($drives.Item($i) -eq $DriveLetter.ToUpper()) {
            return $drives.Item($i + 1)
        }
    }
    return ""
}
# ============================================================
# Active Directory
# ============================================================
function Get-LoggedOnUserADInfo {
    try {
        $net  = New-Object -ComObject WScript.Network
        $user = $net.UserName
        if (-not $user) { return $null }
        # Extract sAMAccountName (in case DOMAIN\user format is returned)
        $sam  = ($user -split '\\')[-1]
        # Get default naming context from AD
        $root = [ADSI]"LDAP://RootDSE"
        $base = "LDAP://$($root.defaultNamingContext)"
        # Create LDAP searcher
        $searcher = New-Object System.DirectoryServices.DirectorySearcher
        $searcher.SearchRoot = $base
        $searcher.Filter     = "(&(objectClass=user)(sAMAccountName=$sam))"
        $searcher.PageSize   = 1
        # Load only required properties for better performance
        $searcher.PropertiesToLoad.AddRange(@(
            "displayname",
            "homedrive",
            "homedirectory",
            "distinguishedname"
        ))
        $result = $searcher.FindOne()
        if ($result -and $result.Properties) {
            $p = $result.Properties
            return [PSCustomObject]@{
                SamAccountName    = $sam
                DisplayName       = $p["displayname"]       | Select-Object -First 1
                HomeDrive         = $p["homedrive"]         | Select-Object -First 1
                HomeDirectory     = $p["homedirectory"]     | Select-Object -First 1
                DistinguishedName = $p["distinguishedname"] | Select-Object -First 1
            }
        }
    } catch {
        Write-Warning "Error in Get-LoggedOnUserADInfo: $_"
    }
    return $null
}
# ============================================================
# Disk
# ============================================================
function Test-DiskOccupation {
    param(
        [string]$DrivePath,
        [int]$Ratio,
        [int]$NetworkTimeout = 5,    # Max seconds to wait for drive access
        [int]$PopupTimeout   = 10,   # Seconds before popup auto-closes
        [bool]$IsFrench      = $true
    )
    if ([string]::IsNullOrEmpty($DrivePath)) { return }
    # Run FSO in a background job to avoid blocking on unreachable network drives
    $job = Start-Job -ScriptBlock {
        param($path)
        $fso = New-Object -ComObject "Scripting.FileSystemObject"
        if (-not $fso.FolderExists($path)) { return $null }
        $drive = $fso.GetDrive($fso.GetDriveName($path))
```

```
        if ($drive.TotalSize -eq 0) { return $null }
        return @{ FreeSpace = $drive.FreeSpace; TotalSize = $drive.TotalSize }
    } -ArgumentList $DrivePath
    $completed = Wait-Job $job -Timeout $NetworkTimeout
    if (-not $completed) {
        Remove-Job $job -Force
        return     # Drive unreachable within timeout
    }
    $result = Receive-Job $job
    Remove-Job $job
    if (-not $result) { return }
    $total      = $result.TotalSize
    $used       = $total - $result.FreeSpace
    $usedPct    = ($used / $total) * 100
    $usedMB     = [math]::Round($used  / 1MB)
    $totalMB    = [math]::Round($total / 1MB)
    $diskLetter = $DrivePath.ToUpper()
    if ($usedPct -gt $Ratio) {
        $msg = Get-Msg `
            -FR ("Votre espace disque disponible sera bientôt atteint !`n`n" +
                 "Vous utilisez $([math]::Round($usedPct,0))% de votre espace disque !`n`n" +
                 "Disque : $diskLetter`n" +
                 "Occupation : $usedMB Mo`n" +
                 "Taille : $totalMB Mo") `
            -EN ("Your disk space will be reached soon!`n`n" +
                 "You are using $([math]::Round($usedPct,0))% of your disk space!`n`n" +
                 "Disk : $diskLetter`n" +
                 "Occupation : $usedMB MB`n" +
                 "Size : $totalMB MB") `
            -IsFrench $IsFrench
        $title = Get-Msg -FR "Attention" -EN "Warning" -IsFrench $IsFrench
        # Buttons+Icon: 48 = OK + Warning icon
        $wsh = New-Object -ComObject "WScript.Shell"
        $ret = $wsh.Popup($msg, $PopupTimeout, $title, 48)
    }
}
# ============================================================
# System
# ============================================================
function Get-LogonServer {
    return $env:LOGONSERVER
}
function Get-OSMajorVersion {
    # Get-CimInstance is reliable on Win 10/11 — [System.Environment] may return 6
    $os = Get-CimInstance -ClassName Win32_OperatingSystem -ErrorAction Stop
    return [int]($os.Version.Split('.')[0])
}
function Get-DriveType {
    param([string]$DrvPath)
    $fso   = New-Object -ComObject "Scripting.FileSystemObject"
    $drive = $fso.GetDrive($DrvPath)
    return $drive.DriveType
}
# ============================================================
# Quick Access
# ============================================================
function Clear-QuickAccess {
    $shell  = New-Object -ComObject "Shell.Application"
    $folder = $shell.Namespace("shell:::{679f85cb-0220-4080-b29b-5540cc05aab6}")
    foreach ($item in $folder.Items()) {
        $item.InvokeVerb("unpinfromhome")
    }
}
function Add-QuickAccess {
    param([string]$Path)
    $shell  = New-Object -ComObject "Shell.Application"
    $folder = $shell.Namespace($Path)
    if ($null -ne $folder) {
        $folder.Self.InvokeVerb("pintohome")
        Start-Sleep -Milliseconds 200
        return $true
    }
    return $false
}
function Set-QuickAccess {
    param(
        [string]$Folders,      # Comma-separated list of folders/drives to pin
        [string]$LinksFolder   # Folder containing .lnk shortcuts to also pin
    )
    # Quick Access is only available on Windows 10+
```

```
    if ((Get-OSMajorVersion) -lt 10) { return }
    Clear-QuickAccess
    $networkDrivePaths = [System.Collections.Generic.List[string]]::new()
    if (-not [string]::IsNullOrEmpty($Folders)) {
        foreach ($folder in $Folders.Split(',')) {
            $folder = $folder.Trim()
            # Strip trailing backslash from bare drive letters e.g. "H:\" → "H:"
            if ($folder.Length -eq 3 -and $folder[2] -eq '\') {
                $folder = $folder.Substring(0, 2)
            }
            # If it's a network drive, store its UNC path to avoid duplicate pins
            $driveType = try { Get-DriveType $folder } catch { -1 }
            if ($driveType -eq 3) {
                $unc = Resolve-NetworkDrive $folder
                $networkDrivePaths.Add($unc.ToLower())
            }
            $ret = Add-QuickAccess $folder
        }
    }
    # Pin shortcuts from the links folder, skipping already-pinned network paths
    if (-not [string]::IsNullOrEmpty($LinksFolder)) {
        $lnkFiles = Get-ChildItem -Path $LinksFolder -Filter "*.lnk" -ErrorAction SilentlyContinue
        foreach ($lnk in $lnkFiles) {
            $target = Get-ShortcutTarget $lnk.FullName
            if (-not $networkDrivePaths.Contains($target.ToLower())) {
                Add-QuickAccess $target
            }
        }
    }
}
# ============================================================
# Main
# ============================================================
$isFrench = ((Get-Culture).TwoLetterISOLanguageName -eq "fr")
Remove-AllNetworkDrives
$user = Get-LoggedOnUserADInfo
if ($null -ne $user) {
    $homeDrive = $user.HomeDrive    # e.g. "H:"
    # Map home drive
    $label = Get-Msg `
        -FR "Espace personnel de $($user.DisplayName)" `
        -EN "Personal space of $($user.DisplayName)" `
        -IsFrench $isFrench
    Connect-Drive -Drive $homeDrive -Path $user.HomeDirectory `
                  -Description $label -IsFrench $isFrench
    # Map group drive
    $groupLabel = Get-Msg -FR "Ma classe [%SECONDARY_GROUP%]" -EN "My class [%SECONDARY_GROUP%]" -IsFrench $isFrench
    Connect-Drive -Drive "%GROUP_DRIVE%" -Path "\\%SECONDARY_GROUP_SERVER%\%SECONDARY_GROUP%$" -Description
$groupLabel -IsFrench $isFrench
    <#
    # Warn if home drive is nearly full
    Test-DiskOccupation -DrivePath $homeDrive -Ratio 80 `
                        -NetworkTimeout 5 -PopupTimeout 20 -IsFrench $isFrench
    #>
    <#
    # Pin folders to Quick Access
    Set-QuickAccess `
        -Folders     "$homeDrive\,$homeDrive\Documents,$homeDrive\Pictures,$homeDrive\Downloads,Q:\" `
        -LinksFolder "$homeDrive\Groups"
    #>
}
# Launch KoXoLabel from NETLOGON — separate script to keep group scripts lean
$logonServer = Get-LogonServer
& "\\$logonServer\NETLOGON\KoXoLabel.ps1" /BackGroundColor="#A56E3A"
```