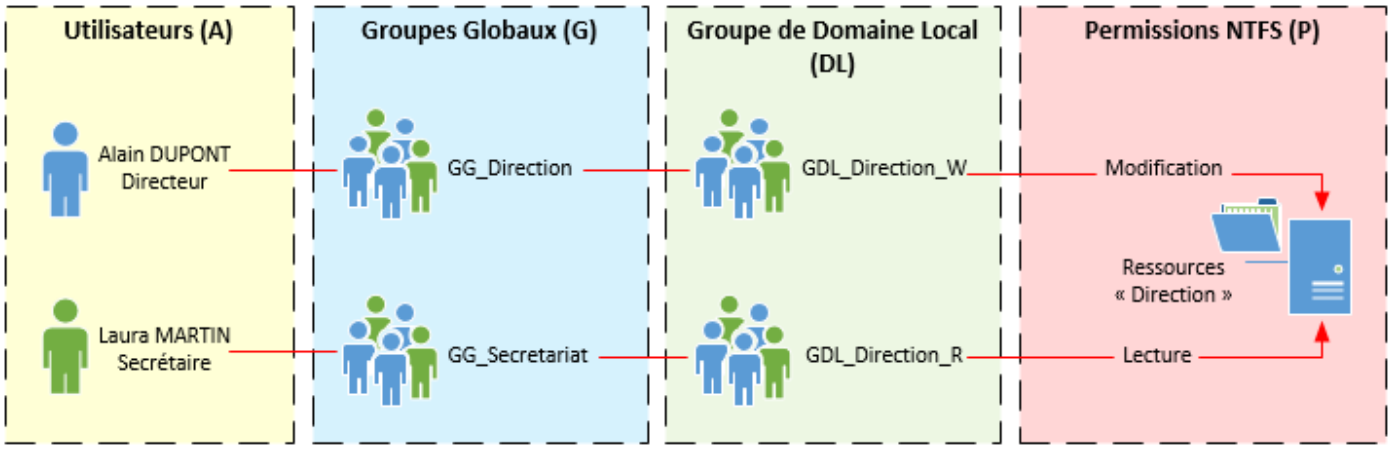
Script Master



Introduction

L'ensemble des scripts vu précédemment est découpé d'une façon logique.
Ainsi, il est possible de modifier le fichier source CSV et relancer les scripts sans devoir relancer toute la machine.

Pour simplifier encore plus tout cela, on peut faire un script master qui gère tout ça.
Voici ce qu'il peut faire :

- Lancer tout les script déjà vu
- Lancer chaque étape individuellement
- Préparer le CSV (nettoyage, tri, export)

Powershell

Voici donc le script :

```
<#
================================================================================
SCRIPT : 0 - Maitre_AGDLP.ps1
OBJET  : Piloter l'ensemble du deploiement AGDLP + utilitaires CSV GG
================================================================================

FONCTIONS MAJEURES
------------------
- Matrice de progression multi-etapes (barre ASCII 40)
- Validations souples (colonnes minimales uniquement avant 1.1/2.1)
- Menu :
```

```
    1) Tout lancer (1.1 -> 2.1 -> 3.1 -> 4.1 -> 5.1 -> 6.1)
    2..7) Etapes individuelles
    8) Audit / Dry-run (dossiers, groupes AD manquants)
    9) Preparer CSV GG (nettoyage + tri topologique + export)
   10) Quitter
- Audit GG : groupes references (dans Arbo) introuvables dans AD, suggestions
- Sorties visibles sans accents (compatibilite), commentaires en francais

REMARQUES
---------
- Les scripts 1.1 a 6.1 doivent exister dans $ScriptsDir

================================================================================
#>

Param(
    [string]$CsvArbo     = "C:\Scripts\Arbo\LCC_Arbo_DATAS.csv",
    [string]$CsvGG       = "C:\Scripts\Arbo\GG_LCC_SVC.csv",
    [string]$Share       = "\\10.0.10.182\datas\",
    [string]$BaseOU_GDL  = "OU=GDL,OU=AGDLP,DC=ccpl,DC=local",
    [string]$BaseOU_GG   = "OU=GG,OU=AGDLP,DC=ccpl,DC=local",
    [string]$ScriptsDir  = "C:\Scripts\Arbo",
    [string]$OutDir      = "C:\Scripts\Arbo\Out"
)

$ErrorActionPreference = "Stop"

# ================================================================================
#  UTILITAIRES AFFICHAGE (barre ASCII 40 + matrice)
# ================================================================================
$AsciiBarLen = 40

function New-AsciiBar {
    param([double]$Percent)
    $Percent = [Math]::Max(0, [Math]::Min(100, $Percent))
    $filled  = [Math]::Floor(($Percent/100) * $AsciiBarLen)
    if ($filled -gt $AsciiBarLen) { $filled = $AsciiBarLen }
    $hash = "#" * $filled
    $dash = "-" * ($AsciiBarLen - $filled)
    return "[{0}{1}] {2}%" -f $hash, $dash, ([Math]::Round($Percent))
```

```
}

# Etapes
$Steps = @(
    @{ Id=1; Name="1/6  Arborescence";   File="1.1-Creation_Arborescence.ps1"; Args={@("-
CsvPath `"$CsvArbo`"","-Destination `"$Share`"")};  State="EN ATTENTE"; Pct=0 },
    @{ Id=2; Name="2/6  GDL + OU";       File="2.1-Creation_GDL.ps1";          Args={@("-
CsvPath `"$CsvArbo`"","-BaseOU `"$BaseOU_GDL`"")};  State="EN ATTENTE"; Pct=0 },
    @{ Id=3; Name="3/6  Parentalite";    File="3.1-Creation_Parentalite.ps1";  Args={@("-
CsvPath `"$CsvArbo`"")};                            State="EN ATTENTE"; Pct=0 },
    @{ Id=4; Name="4/6  ACL";            File="4.1-Appliquer_ACL.ps1";         Args={@("-
RootPath `"$Share`"","-CsvPath `"$CsvArbo`"")};    State="EN ATTENTE"; Pct=0 },
    @{ Id=5; Name="5/6  Creation GG";    File="5.1-Creation_GG.ps1";           Args={@("-
CSVPath `"$CsvGG`"","-OUPath `"$BaseOU_GG`"")};    State="EN ATTENTE"; Pct=0 },
    @{ Id=6; Name="6/6  GG -> GDL";      File="6.1-Ajout_GG_dans_GDL.ps1";      Args={@("-
CsvPath `"$CsvArbo`"")};                            State="EN ATTENTE"; Pct=0 }
)

function Draw-Matrix {
    Clear-Host
    Write-Host "====================== PROGRESSION AGDLP ======================"
    foreach ($s in $Steps) {
        $bar = New-AsciiBar -Percent $s.Pct
        Write-Host ("{0,-26} {1,-4} {2}" -f $s.Name, $s.State, $bar)
    }
    Write-Host "================================================================"
}

# ============================================================================
#  OUTILS VALIDATION / AUDIT
# ============================================================================

function Test-Admin {
    return ([Security.Principal.WindowsPrincipal]
[Security.Principal.WindowsIdentity]::GetCurrent()
    ).IsInRole([Security.Principal.WindowsBuiltInRole]::Administrator)
}

function Require-ModuleAD {
    try { Import-Module ActiveDirectory -ErrorAction Stop; return $true }
```

```
    catch { Write-Warning "Module ActiveDirectory manquant."; return $false }
}

function Test-CsvColumns {
    param([string]$Path,[string[]]$RequiredCols)
    if (!(Test-Path $Path)) { return @("Fichier introuvable: $Path") }
    $rows = Import-Csv -Path $Path -Delimiter ';' -Encoding UTF8
    if (!$rows) { return @("CSV vide: $Path") }
    $cols = ($rows | Select-Object -First 1).PSObject.Properties.Name
    $missing = $RequiredCols | Where-Object { $_ -notin $cols }
    return $missing
}

# Nettoyage leger cellules (NBSP, ZWSP, guillemets typographiques, espaces)
function Clean-Cell([string]$s) {
    if (-not $s) { return "" }
    $s = $s -replace " ", " "
    $s = $s -replace "\u00A0", " "
    $s = $s -replace "\u200B", ""
    $s = $s -replace "[«»“”‘’]", ""
    return ($s -replace "\s+", " ").Trim()
}

# Calcul distance d'edition (Levenshtein) 100% ASCII
function Get-EditDistance {
    param(
        [string]$a,
        [string]$b
    )

    if ($null -eq $a) { $a = "" }
    if ($null -eq $b) { $b = "" }

    $la = $a.Length
    $lb = $b.Length

    # matrice (la+1) x (lb+1)
    $d = New-Object 'int[,]' ($la + 1), ($lb + 1)

    for ($i = 0; $i -le $la; $i++) { $d[$i,0] = $i }
```

```
    for ($j = 0; $j -le $lb; $j++) { $d[0,$j] = $j }

    for ($i = 1; $i -le $la; $i++) {
        for ($j = 1; $j -le $lb; $j++) {

            # PAS de ternaire ici. If clair et propre :
            if ($a[$i-1] -eq $b[$j-1]) {
                $cost = 0
            }
            else {
                $cost = 1
            }

            $delete  = $d[$i-1, $j] + 1
            $insert  = $d[$i, $j-1] + 1
            $replace = $d[$i-1, $j-1] + $cost

            $d[$i,$j] = [Math]::Min([Math]::Min($delete, $insert), $replace)
        }
    }

    return $d[$la,$lb]
}
# Validation generale legere
function Validate-Core {
    $ok=$true
    if (!(Test-Admin)) { Write-Warning "Doit etre lance en administrateur."; $ok=$false }
    if (!(Test-Path $Share)) { Write-Warning "Partage introuvable: $Share"; $ok=$false }
    if (!(Require-ModuleAD)) { $ok=$false }
    if (!(Test-Path $CsvArbo)) { Write-Warning "CSV Arbo introuvable: $CsvArbo"; $ok=$false }
    if (!(Test-Path $CsvGG)) { Write-Warning "CSV GG introuvable: $CsvGG"; $ok=$false }
    if (!(Test-Path $ScriptsDir)) { Write-Warning "ScriptsDir introuvable: $ScriptsDir";
$ok=$false }

    $reqMin = @("NIVEAU 1","NIVEAU 2","NIVEAU 3","NIVEAU 4","NIVEAU 5","NIVEAU 6","NIVEAU
7","NIVEAU 8","CREER_GROUPES")
    $missMin = Test-CsvColumns -Path $CsvArbo -RequiredCols $reqMin
    if ($missMin.Count -gt 0) { Write-Warning "Colonnes minimales manquantes (Arbo):
$($missMin -join ', ')"; $ok=$false }
```

```
    return $ok
}

# Validation specifique par etape (souple)
function Validate-ForStep {
    param([ValidateSet("ARBO","GDL","PARENT","ACL","GG","GGtoGDL","PREP_GG","AUDIT")]
[string]$Step)

    switch ($Step) {
        "ARBO"   { return $true }
        "GDL"    { return $true }
        "PARENT" {
            $need=@("FolderPath","GDL_ECRITURE","GDL_LECTURE","GDL_VUE","Parent_GDL")
            $missing = Test-CsvColumns -Path $CsvArbo -RequiredCols $need
            if ($missing.Count -gt 0) { Write-Warning "Manque pour 3.1: $($missing -join ',
')";
                $ans=Read-Host "Continuer quand meme ? (O/N)"; return ($ans -match '^(?i)o') }
            return $true
        }
        "ACL"    {
            $need=@("FolderPath","GDL_ECRITURE","GDL_LECTURE","GDL_VUE")
            $missing = Test-CsvColumns -Path $CsvArbo -RequiredCols $need
            if ($missing.Count -gt 0) { Write-Warning "Manque pour 4.1: $($missing -join ',
')";
                $ans=Read-Host "Continuer quand meme ? (O/N)"; return ($ans -match '^(?i)o') }
            return $true
        }
        "GG"     {
            $need=@("GG","Membre")
            $missing = Test-CsvColumns -Path $CsvGG -RequiredCols $need
            if ($missing.Count -gt 0) { Write-Warning "Manque pour 5.1: $($missing -join ',
')";
                $ans=Read-Host "Continuer quand meme ? (O/N)"; return ($ans -match '^(?i)o') }
            return $true
        }
        "GGtoGDL" {
            $need=@("FolderPath","GDL_ECRITURE","GDL_LECTURE")
            $missing = Test-CsvColumns -Path $CsvArbo -RequiredCols $need
            if ($missing.Count -gt 0) { Write-Warning "Manque pour 6.1: $($missing -join ',
')";
```

```
                $ans=Read-Host "Continuer quand meme ? (O/N)"; return ($ans -match '^(?i)o') }
            return $true
        }
        "PREP_GG" { return $true }
        "AUDIT"   { return $true }
    }
}

# ================================================================================
#  LANCEMENT D'UNE ETAPE
# ================================================================================
function Run-Step {
    param([ValidateSet("ARBO","GDL","PARENT","ACL","GG","GGtoGDL")] [string]$Step)

    if (-not (Validate-ForStep $Step)) { return }

    # recuperer la structure etape
    $s = switch ($Step) {
        "ARBO"    { $Steps[0] }
        "GDL"     { $Steps[1] }
        "PARENT"  { $Steps[2] }
        "ACL"     { $Steps[3] }
        "GG"      { $Steps[4] }
        "GGtoGDL" { $Steps[5] }
    }

    $s.State = "EN COURS"; $s.Pct = 10; Draw-Matrix

    $path = Join-Path $ScriptsDir $s.File
    if (!(Test-Path $path)) {
        $s.State="ERREUR"; $s.Pct=0; Draw-Matrix
        Write-Warning "Script introuvable: $path"
        return
    }

    try {
        & powershell.exe -ExecutionPolicy Bypass -File $path @(& $s.Args)
        $s.State = "TERMINE"; $s.Pct = 100; Draw-Matrix
    }
    catch {
```

```
        $s.State = "ERREUR"; $s.Pct = 0; Draw-Matrix
        Write-Warning "Erreur a l'execution: $path -> $_"
    }
}

# ============================================================================
#  AUDIT / DRY-RUN
# ============================================================================
function Audit-Only {
    Write-Host ""
    Write-Host "=== AUDIT (dry-run) ==="

    # 1) Dossiers manquants
    $rows = Import-Csv -Path $CsvArbo -Delimiter ';' -Encoding UTF8
    $missingFolders = @()
    foreach ($r in $rows) {
        if ([string]::IsNullOrWhiteSpace($r.FolderPath)) { continue }
        $fp = Join-Path $Share $r.FolderPath
        if (!(Test-Path -LiteralPath $fp)) { $missingFolders += $fp }
    }
    Write-Host "Dossiers manquants : $($missingFolders.Count)"
    if ($missingFolders.Count -gt 0) { $missingFolders | Select-Object -Unique | ForEach-
Object { "  - $_" } }

    # 2) Groupes GDL manquants (si colonnes presentes)
    $cols = ($rows | Select-Object -First 1).PSObject.Properties.Name
    if (@("GDL_ECRITURE","GDL_LECTURE","GDL_VUE","Parent_GDL") | ForEach-Object { $_ -in $cols
} | Where-Object {$_} | Measure-Object | Select-Object -ExpandProperty Count) {
        $needGroups = @()
        foreach ($r in $rows) {
            foreach ($g in @($r.GDL_ECRITURE,$r.GDL_LECTURE,$r.GDL_VUE,$r.Parent_GDL)) {
                if ([string]::IsNullOrWhiteSpace($g)) { continue }
                if (!(Get-ADGroup -Filter "Name -eq '$g'" -ErrorAction SilentlyContinue)) {
$needGroups += $g }
            }
        }
        Write-Host "GDL manquants : $($needGroups.Count)"
        if ($needGroups.Count -gt 0) { $needGroups | Select-Object -Unique | ForEach-Object {
"  - $_" } }
    } else {
```

```
        Write-Host "(Colonnes GDL_* absentes pour l'instant — lancer 2.1 pour les creer.)"
    }

    # 3) Audit GG references dans CSV Arbo vs AD
    $ggRefs = @()
    foreach ($r in $rows) {
        foreach ($cell in @($r.GG_ECRITURE,$r.GG_LECTURE)) {
            $cell = Clean-Cell $cell
            if (-not $cell) { continue }
            $ggRefs += ($cell.Split(",") | ForEach-Object { Clean-Cell $_ } | Where-Object {
$_ -ne "" })
        }
    }
    $ggRefs = $ggRefs | Select-Object -Unique
    $missingGG = @()
    foreach ($g in $ggRefs) {
        if (-not (Get-ADGroup -Identity $g -ErrorAction SilentlyContinue)) { $missingGG += $g
}
    }
    Write-Host "GG references mais introuvables dans AD : $($missingGG.Count)"
    if ($missingGG.Count -gt 0) {
        # suggestions
        $allKnown = Get-ADGroup -LDAPFilter "(cn=LCC_GG_*)" -SearchScope Subtree -ErrorAction
SilentlyContinue | Select-Object -ExpandProperty Name
        foreach ($m in $missingGG) {
            $best=""; $bestScore=9999
            foreach ($k in $allKnown) {
                $d = Get-EditDistance $m $k
                if ($d -lt $bestScore) { $bestScore=$d; $best=$k }
            }
            if ($best) { Write-Host ("  - {0}  (suggestion: {1})" -f $m, $best) } else {
Write-Host ("  - {0}" -f $m) }
        }
    }
    Read-Host "Fin audit - Entrer pour continuer" | Out-Null
}

# ================================================================================
#  PREPARATION CSV GG (nettoyage + tri topologique + export)
# ================================================================================
```

```
function Prepare-GG-CSV {
    if (!(Test-Path $CsvGG)) { Write-Warning "CSV GG introuvable: $CsvGG"; return }

    if (!(Test-Path $OutDir)) { New-Item -ItemType Directory -Path $OutDir -Force | Out-Null }
    $out = Join-Path $OutDir "GG_LCC_SVC.sorted.csv"

    Write-Host "Lecture et nettoyage de $CsvGG ..."
    $df = Import-Csv -Path $CsvGG -Delimiter ';' | ForEach-Object {
        [pscustomobject]@{
            GG     = (Clean-Cell $_.GG)
            Membre = (Clean-Cell $_.Membre)
        }
    }

    # Supprimer lignes totalement vides
    $df = $df | Where-Object { $_.GG -or $_.Membre }

    # Construire dependances : parent depend de chaque Membre *_GG_*
    $parents = @{}
    $refs    = @{}
    foreach ($r in $df) {
        if (-not $r.GG) { continue }
        if (-not $parents.ContainsKey($r.GG)) { $parents[$r.GG] = New-Object
System.Collections.Generic.HashSet[string] ([StringComparer]::OrdinalIgnoreCase) }
        if ($r.Membre -and $r.Membre -like "*_GG_*" -and $r.Membre -ne $r.GG) {
            if (-not $refs.ContainsKey($r.Membre)) { $refs[$r.Membre] = $true }
            [void]$parents[$r.GG].Add($r.Membre)
        }
    }

    # Ensemble des noeuds
    $all = New-Object System.Collections.Generic.HashSet[string]
([StringComparer]::OrdinalIgnoreCase)
    foreach ($k in $parents.Keys) { [void]$all.Add($k) }
    foreach ($k in $refs.Keys)    { [void]$all.Add($k) }

    # Calcul indegree pour Kahn
    $indeg = @{}
    foreach ($n in $all) { $indeg[$n] = 0 }
```

```
foreach ($p in $parents.Keys) {
    foreach ($c in $parents[$p]) {
        if (-not $indeg.ContainsKey($c)) { $indeg[$c] = 0 }
        $indeg[$p] += 0 # ensure key
        $indeg[$p] = $indeg[$p]       # parent indeg sera incremente ci-dessous
    }
}
# indegree = nb de dependances (c -> p) : parent a +1 par enfant reference
foreach ($p in $parents.Keys) {
    foreach ($c in $parents[$p]) {
        $indeg[$p] = $indeg[$p] + 1
    }
}

# File des noeuds indeg 0
$queue = New-Object System.Collections.Queue
foreach ($n in $all) { if ($indeg[$n] -eq 0) { $queue.Enqueue($n) } }

$order = New-Object System.Collections.ArrayList
while ($queue.Count -gt 0) {
    $n = $queue.Dequeue()
    [void]$order.Add($n)
    foreach ($p in $parents.Keys) {
        if ($parents[$p].Contains($n)) {
            $indeg[$p] = $indeg[$p] - 1
            [void]$parents[$p].Remove($n)
            if ($indeg[$p] -eq 0) { $queue.Enqueue($p) }
        }
    }
}

# Cycles residuels
$cyclic = @()
foreach ($n in $indeg.Keys) { if ($indeg[$n] -gt 0) { $cyclic += $n } }

# Reconstruire CSV trie : pour chaque GG dans l'ordre topo, garder ses lignes
$declares = ($df | Select-Object -ExpandProperty GG -Unique)
$blocks = New-Object System.Collections.ArrayList
foreach ($g in $order) {
    if ($declares -contains $g) {
```

```
            $blk = $df | Where-Object { $_.GG -eq $g }
            foreach ($r in $blk) { [void]$blocks.Add($r) }
        }
    }
    # Ajouter ceux eventuellement manques
    foreach ($g in $declares) {
        if (-not ($order -contains $g)) {
            $blk = $df | Where-Object { $_.GG -eq $g }
            foreach ($r in $blk) { [void]$blocks.Add($r) }
        }
    }

    # Export
    $blocks | Export-Csv -Path $out -Delimiter ';' -NoTypeInformation -Encoding UTF8
    Write-Host "CSV GG trie ecrit : $out"

    if ($cyclic.Count -gt 0) {
        Write-Warning "Attention: dependances cycliques detectees:"
        $cyclic | Select-Object -Unique | ForEach-Object { "  - $_" }
    }

    $ans = Read-Host "Remplacer le CSV GG d'origine par la version triee ? (O/N)"
    if ($ans -match '^(?i)o') {
        Copy-Item -Path $out -Destination $CsvGG -Force
        Write-Host "Remplacement effectue."
    }
    Read-Host "Fin preparation GG - Entrer pour continuer" | Out-Null
}

# ================================================================================
#  MENU PRINCIPAL
# ================================================================================
function Main-Menu {
    do {
        Draw-Matrix
        Write-Host ""
        Write-Host "=== MENU AGDLP ==="
        Write-Host "1) Tout lancer (1.1 -> 2.1 -> 3.1 -> 4.1 -> 5.1 -> 6.1)"
        Write-Host "2) 1.1 - Creer arborescence"
        Write-Host "3) 2.1 - Creer GDL + OU"
```

```
        Write-Host "4) 3.1 - Parentalite GDL"
        Write-Host "5) 4.1 - Appliquer ACL"
        Write-Host "6) 5.1 - Creer GG"
        Write-Host "7) 6.1 - Ajouter GG -> GDL"
        Write-Host "8) Audit / Dry-run"
        Write-Host "9) Preparer CSV GG (nettoyage + tri topo)"
        Write-Host "10) Quitter"

        $coreOK = Validate-Core
        $choice = Read-Host "Choix"

        if (-not $coreOK -and $choice -notin @('8','9','10')) {
            Write-Warning "Pre-requis generaux non valides. Corriger puis relancer."
            Read-Host "Entrer pour continuer" | Out-Null
            continue
        }

        switch ($choice) {
            '1' { Run-Step ARBO; Run-Step GDL; Run-Step PARENT; Run-Step ACL; Run-Step GG;
Run-Step GGtoGDL; Read-Host "Flux complet termine - Entrer pour continuer" | Out-Null }
            '2' { Run-Step ARBO;     Read-Host "Entrer pour continuer" | Out-Null }
            '3' { Run-Step GDL;      Read-Host "Entrer pour continuer" | Out-Null }
            '4' { Run-Step PARENT;   Read-Host "Entrer pour continuer" | Out-Null }
            '5' { Run-Step ACL;      Read-Host "Entrer pour continuer" | Out-Null }
            '6' { Run-Step GG;       Read-Host "Entrer pour continuer" | Out-Null }
            '7' { Run-Step GGtoGDL;  Read-Host "Entrer pour continuer" | Out-Null }
            '8' { Audit-Only }
            '9' { Prepare-GG-CSV }
            '10' { return }
            default { }
        }
    } while ($true)
}

# Lancement
Main-Menu
```

Utilisation

On peut ouvrir le script dans Powershell ISE ou Powershell basique, et au moment de l'exécution, nous avons ce retour :

```
==================== PROGRESSION AGDLP ====================
1/6  Arborescence          EN ATTENTE [--------------------------------------] 0%
2/6  GDL + OU              EN ATTENTE [--------------------------------------] 0%
3/6  Parentalite           EN ATTENTE [--------------------------------------] 0%
4/6  ACL                   EN ATTENTE [--------------------------------------] 0%
5/6  Creation GG           EN ATTENTE [--------------------------------------] 0%
6/6  GG -> GDL             EN ATTENTE [--------------------------------------] 0%
============================================================


=== MENU AGDLP ===
1) Tout lancer (1.1 -> 2.1 -> 3.1 -> 4.1 -> 5.1 -> 6.1)
2) 1.1 - Creer arborescence
3) 2.1 - Creer GDL + OU
4) 3.1 - Parentalite GDL
5) 4.1 - Appliquer ACL
6) 5.1 - Creer GG
7) 6.1 - Ajouter GG -> GDL
8) Audit / Dry-run
9) Preparer CSV GG (nettoyage + tri topo)
10) Quitter
Choix :
```

On peut alors choisir les actions à mener, une barre de progression s'affiche pour chacun des 6 scripts principaux afin de voir ou se trouve l'éventuel blocage.

Revision #4
Created 2026-02-25 15:53:10 UTC by clement-derouet
Updated 2026-02-26 11:16:59 UTC by clement-derouet