**Unity Automation — Handover Manual (Assistant‑Oriented, Merged v4)**

Purpose: give a NEW ChatGPT assistant everything needed to resume work immediately on the Unity Automation project without any extra context from the user. This document merges the original v1 handover (vision + Workflow A/B) with the latest progress from this thread (router v1.7.10, lint/fix, compile-wait, colorized logging, reconcile, and quick-start test blocks).

# Project Vision

We are building a two‑loop system where an assistant can design, modify, and test Unity projects end‑to‑end with minimal friction and maximum resilience.

## Workflow A — Git + Patch Router (producer of patches)

Purpose: observe the Unity project repository, accept patches produced by assistants, and apply them automatically so the Unity project is kept up‑to‑date. Workflow A runs outside Unity and is resilient to Unity crashes: it stores, verifies, and applies patches using git and a PowerShell inbox‑router.

### Key components

• Git repository (project root).

• inbox-router.ps1 (PowerShell script) which watches Downloads and applies patches automatically or in‑place depending on mode.

• Logs: patch-router.log (primary). Paths are referenced below.

### Core git operations used by the router

git apply --check --3way --whitespace=nowarn <patch>

git apply --index --3way --whitespace=nowarn <patch>

git add Assets -A

git commit -m "inbox: apply <patch>"

git push

## Workflow B — Realtime Voice Assistant inside Unity (consumer and testbed)

Purpose: this is the in‑Unity voice assistant. While running in Play Mode the user speaks to the assistant and instructs it to design or modify gameplay, scenes, or systems (for example: “Create a 3D Snake game”). The assistant performs edits (via scripted operations or by generating patches applied by Workflow A), the user tests in Play Mode, and then exits Play Mode to commit/log results.

### Key guarantees & behavior

• The assistant running inside Unity (Workflow B) is stateful during Play Mode, and can run code that manipulates the scene/project.

• When the user exits Play Mode, Workflow B should produce: (a) a concise, timestamped log of actions taken; (b) any generated or modified files staged in the Unity project (or exported as a patch).

• Workflow A consumes these logs/patches and applies them so the next loop starts from the updated project.

## A ↔ B Contract (high‑level loop)

• You run Unity and enter Play Mode; the in‑editor assistant listens and edits.

• When Play Mode ends, B emits logs and/or a patch.

• You download the patch. A detects a ready .patch in Downloads, applies it, commits, (best‑effort) syncs, then waits for a successful recompile signal from Unity.

• Once Unity recompiles successfully, A logs success and you can re‑enter Play Mode.

# Current State / Environment

• OS: Windows; baseline PowerShell: 5.1

• Local repo root: C:\Users\ander\My project

• Router path: C:\Users\ander\inbox-router.ps1

• Log file: %USERPROFILE%\patch-router.log

• Unity compile signal file: Assets\\InboxPatches\\CompileDone.stamp (written by Editor-only static constructor)

• Latest router: v1.7.10 (pre‑lint + post‑verify/fix + reconcile + detached‑HEAD guard + compile‑wait)

• Last observed smoke tests: 2501–2503 (UNITY wait + success).

# User Preferences & Conventions (must‑follow)

• Provide single, self‑contained copy/paste blocks that: stop router, clear log, (re)install script, start router in a new window, and tail logs here.

• Always include server start + live log tail in the copy/paste.

• Always ask the user for the LAST patch number they downloaded and start from the next number.

• Use fresh, increasing patch IDs; avoid collisions.

• Do not ask the user to manually edit files. The router auto‑repairs missing #endif for #if UNITY\_EDITOR blocks and reconstructs new .cs files from patch lines.

• If a patch contains no .cs changes, log UNITY skip and still complete.

• Treat pull/push issues as warnings (local success still valid).

• Prefer direct, ready‑to‑download patch links; if not possible, include a helper that generates the .patch locally with correct CRLF ASCII.

# How It Works (Router & Unity)

## Router (Workflow A)

Watches Downloads → parses patch → lints C# additions (ensures #if UNITY\_EDITOR is balanced and braces match) → applies or reconciles if files already exist → post‑verifies and fixes newly created .cs (appending a missing #endif if needed) → commits → best‑effort pull/push → waits for Unity compile (stamp file) → logs result.

## Unity (Workflow B)

An Editor‑only InitializeOnLoad static constructor writes CompileDone.stamp on successful domain reload. Any successful reload updates the stamp and unblocks the router.

# Router v1.7.10 — Full Script (paste this into inbox-router.ps1)

# inbox-router.ps1 v1.7.10

# - PS 5.1-safe

# - Normalize detached HEAD -> main

# - Pre-apply linter (UNITY\_EDITOR/#endif + braces)

# - Post-apply verify/fix for NEW .cs files; appends missing '#endif' if needed

# - Reconcile new-file patches when files already exist (idempotent)

# - Wait for Unity compile (no timeout) via Assets\InboxPatches\CompileDone.stamp

param()

Set-StrictMode -Version Latest

$ErrorActionPreference = "Stop"

# Settings

$RepoRoot = "C:\Users\ander\My project"

$Downloads = "$env:USERPROFILE\Downloads"

$LogPath = "$env:USERPROFILE\patch-router.log"

$DebounceMs = 1500

$SettleMs = 1200

$PollMs = 500

function Write-Log([string]$msg) {

try {

$ts = (Get-Date).ToUniversalTime().ToString('yyyy-MM-ddTHH:mm:ssZ')

Add-Content -LiteralPath $LogPath -Value "[$ts] $msg"

} catch {}

}

# Single-instance guard

try {

$thisPath = $MyInvocation.MyCommand.Path

$others = Get-CimInstance Win32\_Process |

Where-Object { $\_.ProcessId -ne $PID -and $\_.CommandLine -match [regex]::Escape($thisPath) }

if ($others) { Write-Log ("ABORT already-running pids=" + (($others | Select-Object -Expand ProcessId) -join ',')); return }

} catch {}

Write-Log ("BOOT who={0}\{1} pid={2} pwsh={3} cwd={4}" -f $env:COMPUTERNAME,$env:USERNAME,$PID,$PSVersionTable.PSVersion,(Get-Location))

Write-Log ("SCRIPT version=v1.7.10 path={0}" -f $MyInvocation.MyCommand.Path)

# Validate repo + downloads

if (-not (Test-Path -LiteralPath $RepoRoot)) { Write-Log ("ERROR missing repo path: {0}" -f $RepoRoot); return }

if (-not (Test-Path -LiteralPath $Downloads)) { Write-Log ("ERROR missing Downloads path: {0}" -f $Downloads); return }

try {

Push-Location -LiteralPath $RepoRoot

& git rev-parse --is-inside-work-tree \*> $null

if ($LASTEXITCODE -ne 0) { Write-Log ("ERROR not a git repo: {0}" -f $RepoRoot); return }

} finally { Pop-Location }

function Invoke-Git([string[]]$gitArgs) {

try {

Push-Location -LiteralPath $RepoRoot

& git @gitArgs 2>&1 | ForEach-Object { Write-Log ("GIT " + $\_) }

return $LASTEXITCODE

} catch { Write-Log ("GIT exception: " + $\_.Exception.Message); return 1 }

finally { Pop-Location }

}

function Get-CurrentBranch() {

try {

Push-Location -LiteralPath $RepoRoot

$out = & git symbolic-ref --quiet --short HEAD 2>&1

if ($LASTEXITCODE -eq 0) { return ($out | Select-Object -First 1).Trim() }

else { return $null }

} catch { return $null }

finally { Pop-Location }

}

function Ensure-BranchMain() {

$cur = Get-CurrentBranch

$detached = (-not $cur)

if (-not $detached -and $cur -eq 'main') { return $true }

$curDisplay = '<detached>'; if ($cur) { $curDisplay = $cur }

Write-Log ("ENSURE main: cur='{0}' detached={1}" -f $curDisplay, $detached)

[void](Invoke-Git @('fetch','--prune','origin'))

$haveMain = ((Invoke-Git @('rev-parse','--verify','main')) -eq 0)

if (-not $haveMain) { [void](Invoke-Git @('branch','-f','main','HEAD')) }

$sw = Invoke-Git @('switch','-f','main')

if ($sw -ne 0) {

$co = Invoke-Git @('checkout','-f','main')

if ($co -ne 0) { Write-Log "ENSURE main: failed to switch/checkout main"; return $false }

}

if (Invoke-Git @('-c','rebase.autoStash=true','pull','--rebase','origin','main')) {

Write-Log "ENSURE main: pull --rebase failed"; return $false

}

Write-Log "ENSURE main: now on 'main' and up-to-date"

return $true

}

function Parse-PatchInfo([string]$patchPath) {

$text = Get-Content -LiteralPath $patchPath -Raw -Encoding UTF8

$lines = $text -split "`r?`n"

$info = @{}; $file=$null; $inHunk=$false; $isNewForCurrent=$false

foreach ($ln in $lines) {

if ($ln -like 'diff --git \*') { $file=$null; $inHunk=$false; $isNewForCurrent=$false; continue }

if ($ln -like 'new file mode \*') { $isNewForCurrent=$true; continue }

if ($ln -like '+++ b/\*') {

$candidate = $ln.Substring(6); $file = $candidate

if (-not $info.ContainsKey($file)) {

$isCs = ($file -like '\*.cs')

$info[$file] = @{ IsCs = $isCs; IsNew = $isNewForCurrent; AddedLines = @() }

} else { if ($isNewForCurrent) { $info[$file]['IsNew'] = $true } }

continue

}

if ($ln -like '@@\*@@\*') { $inHunk = $true; continue }

if ($inHunk -and $file -ne $null -and $ln.StartsWith('+') -and -not $ln.StartsWith('+++')) {

$info[$file]['AddedLines'] += $ln.Substring(1)

}

}

return $info

}

function Lint-CsAdded([string]$file,[string[]]$addedLines) {

if (-not $addedLines -or $addedLines.Count -eq 0) { return $true }

$txt = [string]::Join("`n", $addedLines)

$ifs = [regex]::Matches($txt, '^\s\*#if\s+UNITY\_EDITOR\b', 'Multiline').Count

$endifs = [regex]::Matches($txt, '^\s\*#endif\s\*$', 'Multiline').Count

if ($ifs -gt 0 -and $endifs -eq 0) { Write-Log ("LINT fail: {0} has #if UNITY\_EDITOR without #endif" -f $file); return $false }

$nonEmpty = $addedLines | ForEach-Object { $\_.Trim() } | Where-Object { $\_ -ne '' }

if ($nonEmpty.Count -gt 0) {

$first = $nonEmpty[0]; $last = $nonEmpty[$nonEmpty.Count-1]

if ($first -match '^\s\*#if\s+UNITY\_EDITOR\b' -and $last -notmatch '^\s\*#endif\s\*$') {

Write-Log ("LINT fail: {0} begins with #if UNITY\_EDITOR but last added line is not #endif" -f $file); return $false }

}

$opens = ([regex]::Matches($txt, '\{')).Count; $closes = ([regex]::Matches($txt, '\}')).Count

if ($opens -ne $closes) { Write-Log ("LINT fail: {0} unbalanced braces ({1} vs {2})" -f $file, $opens, $closes); return $false }

Write-Log ("LINT ok: {0} (#if={1}, #endif={2}, braces ok)" -f $file, $ifs, $endifs)

return $true

}

function Lint-Patch([hashtable]$info) {

foreach ($k in $info.Keys) {

$v = $info[$k]

if ($v['IsCs']) { if (-not (Lint-CsAdded $k $v['AddedLines'])) { return $false } }

}

return $true

}

function Verify-And-Fix-NewCs([hashtable]$info) {

foreach ($k in $info.Keys) {

$v = $info[$k]

if (-not $v['IsCs']) { continue }

if (-not $v['IsNew']) { continue }

$rel = $k -replace '/', '\'

$full = Join-Path $RepoRoot $rel

$expected = [string]::Join("`r`n", $v['AddedLines']) + "`r`n"

$needEndIf = $false

foreach ($line in $v['AddedLines']) { if ($line -match '^\s\*#if\s+UNITY\_EDITOR\b') { $needEndIf = $true; break } }

$shouldHaveEndIf = $false

foreach ($line in $v['AddedLines']) { if ($line -match '^\s\*#endif\s\*$') { $shouldHaveEndIf = $true; break } }

$repair = $false

$actual = ''

if (Test-Path -LiteralPath $full) { try { $actual = Get-Content -LiteralPath $full -Raw -Encoding UTF8 } catch {} }

$actualLines = @()

if ($actual -ne '') { $actualLines = $actual -split "`r?`n" }

if ($needEndIf -and -not $shouldHaveEndIf) {

$repair = $true

Write-Log ("VERIFY WARN: patch for {0} lacks #endif in added lines" -f $k)

} else {

if ($actualLines.Count -lt $v['AddedLines'].Count) { $repair = $true }

if ($needEndIf) {

$hasEndIfOnDisk = $false

foreach ($line in $actualLines) { if ($line -match '^\s\*#endif\s\*$') { $hasEndIfOnDisk = $true; break } }

if (-not $hasEndIfOnDisk) { $repair = $true }

}

}

if ($repair) {

try {

$dir = Split-Path -Parent $full

if (-not (Test-Path -LiteralPath $dir)) { New-Item -ItemType Directory -Path $dir -Force \*> $null }

$needAppend = $needEndIf -and (-not $shouldHaveEndIf)

if ($needAppend) { $expected = $expected + "#endif`r`n"; Write-Log ("FIXUP appended 1 '#endif' to {0}" -f $full) }

else { Write-Log ("FIXUP wrote expected content for {0} (from patch added lines)" -f $k) }

Set-Content -LiteralPath $full -Value $expected -Encoding UTF8

} catch {

Write-Log ("FIXUP failed to write ${k}: " + $\_.Exception.Message)

return $false

}

}

try {

$post = Get-Content -LiteralPath $full -Raw -Encoding UTF8

$ifs = ([regex]::Matches($post, '^\s\*#if\s+UNITY\_EDITOR\b', 'Multiline')).Count

$endifs = ([regex]::Matches($post, '^\s\*#endif\s\*$', 'Multiline')).Count

Write-Log ("POSTCHECK {0}: #if={1} #endif={2}" -f $full, $ifs, $endifs)

} catch {}

}

return $true

}

function Reconcile-NewFiles([hashtable]$info,[string]$patchName) {

$anyWrites = $false

$allNew = $true

foreach ($k in $info.Keys) { if (-not $info[$k]['IsNew']) { $allNew = $false; break } }

if (-not $allNew) { return $false }

foreach ($k in $info.Keys) {

$rel = ($k -replace '/', '\')

$full = Join-Path $RepoRoot $rel

$expected = [string]::Join("`r`n", $info[$k]['AddedLines']) + "`r`n"

if (-not (Test-Path -LiteralPath $full)) {

try {

$dir = Split-Path -Parent $full

if (-not (Test-Path -LiteralPath $dir)) { New-Item -ItemType Directory -Path $dir -Force \*> $null }

Set-Content -LiteralPath $full -Value $expected -Encoding UTF8

Write-Log ("RECONCILE wrote missing file {0}" -f $k)

$anyWrites = $true

} catch {

Write-Log ("RECONCILE failed to write ${k}: " + $\_.Exception.Message)

return $false

}

} else {

$actual = ''

try { $actual = Get-Content -LiteralPath $full -Raw -Encoding UTF8 } catch {}

if ($actual -ne $expected) {

try {

Set-Content -LiteralPath $full -Value $expected -Encoding UTF8

Write-Log ("RECONCILE replaced existing {0} with expected content" -f $k)

$anyWrites = $true

} catch {

Write-Log ("RECONCILE failed to replace ${k}: " + $\_.Exception.Message)

return $false

}

} else {

Write-Log ("RECONCILE skip: {0} already matches expected content" -f $k)

}

}

}

if ($anyWrites) {

if (Invoke-Git @('add','-A')) { Write-Log ("RECONCILE stage-failed: {0}" -f $patchName); return $false }

$msg = "inbox: reconcile " + $patchName

if (Invoke-Git @('commit','-m',$msg)) { Write-Log ("RECONCILE commit-failed: {0}" -f $patchName); return $false }

if (Invoke-Git @('-c','rebase.autoStash=true','pull','--rebase','origin','main')) { Write-Log ("RECONCILE pull-failed: {0}" -f $patchName); return $false }

if (Invoke-Git @('push','-u','origin','HEAD:main')) { Write-Log ("RECONCILE push-failed: {0}" -f $patchName); return $false }

} else {

Write-Log ("RECONCILE result: all files already present and identical for {0}" -f $patchName)

}

if (-not (Verify-And-Fix-NewCs $info)) { Write-Log ("RECONCILE verify-failed: {0}" -f $patchName); return $false }

return $true

}

function Wait-ForUnityCompile([datetime]$sinceUtc) {

$stamp = Join-Path $RepoRoot 'Assets\InboxPatches\CompileDone.stamp'

Write-Log ("UNITY wait: request={0:o} (waiting until CompileDone.stamp >= request)" -f $sinceUtc)

while ($true) {

try {

if (Test-Path -LiteralPath $stamp) {

$w = (Get-Item -LiteralPath $stamp).LastWriteTimeUtc

if ($w -ge $sinceUtc) { Write-Log ("UNITY compiled: {0:o} (>= request)" -f $w); break }

}

} catch {}

Start-Sleep -Milliseconds 500

}

}

function Apply-Patch([string]$fullPath) {

$name = Split-Path -Leaf $fullPath

Write-Log ("APPLY start: {0}" -f $name)

try {

if (-not (Test-Path -LiteralPath $fullPath)) { Write-Log ("APPLY missing: {0}" -f $name); return }

if (-not (Ensure-BranchMain)) { Write-Log "APPLY aborted: could not ensure 'main'"; return }

$pi = Parse-PatchInfo $fullPath

if (-not (Lint-Patch $pi)) { Write-Log ("APPLY lint-failed: {0}" -f $name); return }

$check = Invoke-Git @('-c','core.safecrlf=false','apply','--check','--whitespace=nowarn',$fullPath)

if ($check -ne 0) {

if (Reconcile-NewFiles $pi $name) {

$since = (Get-Date).ToUniversalTime()

Wait-ForUnityCompile $since

Write-Log ("APPLY success (reconciled, local-only possible): {0}" -f $name)

return

}

$rev = Invoke-Git @('-c','core.safecrlf=false','apply','-R','--check','--whitespace=nowarn',$fullPath)

if ($rev -eq 0) { Write-Log ("APPLY already-applied: {0}" -f $name); return }

Write-Log ("APPLY check-failed: {0}" -f $name); return

}

if (Invoke-Git @('-c','core.safecrlf=false','apply','--whitespace=nowarn',$fullPath)) { Write-Log ("APPLY failed: {0}" -f $name); return }

if (-not (Verify-And-Fix-NewCs $pi)) { Write-Log ("APPLY verify-failed: {0}" -f $name); return }

if (Invoke-Git @('add','-A')) { Write-Log ("APPLY stage-failed: {0}" -f $name); return }

$msg = "inbox: apply " + $name

if (Invoke-Git @('commit','-m',$msg)) { Write-Log ("APPLY commit-failed: {0}" -f $name); return }

if (Invoke-Git @('-c','rebase.autoStash=true','pull','--rebase','origin','main')) { Write-Log ("APPLY pull-warning: {0}" -f $name) }

if (Invoke-Git @('push','-u','origin','HEAD:main')) { Write-Log ("APPLY push-warning: {0}" -f $name) }

$since = (Get-Date).ToUniversalTime()

$touchedCs = $false; foreach($k in $pi.Keys){ if($pi[$k]['IsCs']){ $touchedCs = $true; break } }

if ($touchedCs) { Wait-ForUnityCompile $since } else { Write-Log "UNITY skip: no .cs changes in this patch" }

Write-Log ("APPLY success: {0}" -f $name)

}

catch { Write-Log ("APPLY exception: " + $\_.Exception.Message) }

}

$processed = @{} # path -> ticks

Write-Log ("WATCH polling: {0} (\*.patch) debounce={1} settle={2} poll={3}" -f $Downloads,$DebounceMs,$SettleMs,$PollMs)

while ($true) {

try {

$files = Get-ChildItem -LiteralPath $Downloads -Filter '\*.patch' -File -ErrorAction SilentlyContinue

foreach ($f in $files) {

$path = $f.FullName

$ticks = $f.LastWriteTimeUtc.Ticks

if ($processed.ContainsKey($path) -and $processed[$path] -eq $ticks) { continue }

$ageMs = [int]((New-TimeSpan -Start $f.LastWriteTimeUtc -End (Get-Date).ToUniversalTime()).TotalMilliseconds)

if ($ageMs -lt $DebounceMs) { continue }

try {

$a = Get-Item -LiteralPath $path -ErrorAction SilentlyContinue

Start-Sleep -Milliseconds $SettleMs

$b = Get-Item -LiteralPath $path -ErrorAction SilentlyContinue

if (-not $a -or -not $b -or ($a.Length -ne $b.Length -or $a.LastWriteTimeUtc -ne $b.LastWriteTimeUtc)) { continue }

} catch { continue }

Write-Log ("READY detected: '{0}' size={1} lastWriteUtc={2:o}" -f $path,$f.Length,$f.LastWriteTimeUtc)

Apply-Patch $path

$processed[$path] = $ticks

}

} catch { Write-Log ("LOOP exception: " + $\_.Exception.Message) }

Start-Sleep -Milliseconds $PollMs

}

# Install/Restart + Smoke Test Block (copy/paste)

# === Router v1.7.10 install + restart + smoke test (PS 5.1) ===

$RepoRoot = 'C:\Users\ander\My project'

$RouterPath = 'C:\Users\ander\inbox-router.ps1'

$LogPath = "$env:USERPROFILE\patch-router.log"

$psExe = "$env:SystemRoot\System32\WindowsPowerShell\v1.0\powershell.exe"

$dl = "$env:USERPROFILE\Downloads"

# Stop router + clear log

Get-CimInstance Win32\_Process |

Where-Object { $\_.CommandLine -match [regex]::Escape($RouterPath) } |

ForEach-Object { try { Stop-Process -Id $\_.ProcessId -Force } catch {} }

Remove-Item -LiteralPath $LogPath -ErrorAction SilentlyContinue

# Write router (paste script from this doc)

$body = @'

REPLACE\_WITH\_SCRIPT\_FROM\_DOC

'@

Set-Content -LiteralPath $RouterPath -Value $body -Encoding UTF8

# Start router new window + tail

Start-Process -FilePath $psExe -ArgumentList @('-NoProfile','-ExecutionPolicy','Bypass','-NoExit','-File', $RouterPath) -WindowStyle Normal

for ($i=0; $i -lt 40 -and -not (Test-Path -LiteralPath $LogPath); $i++) { Start-Sleep -Milliseconds 250 }

if (Test-Path -LiteralPath $LogPath) { Get-Content -LiteralPath $LogPath -Tail 20 }

# Helper to make unified-diff new-file patches

function New-AddFilePatch([int]$Num,[string]$RelPath,[string[]]$ContentLines) {

$nn = '{0:D4}' -f $Num

$patchName = "patch\_${nn}.patch"

$target = Join-Path $dl $patchName

$lineCount = $ContentLines.Length

$hunkHeader = "@@ -0,0 +$lineCount @@"

$plusLines = $ContentLines | ForEach-Object { '+' + $\_ }

$body = @("diff --git a/$RelPath b/$RelPath","new file mode 100644","--- /dev/null","+++ b/$RelPath",$hunkHeader) + $plusLines

Set-Content -LiteralPath $target -Value ($body -join "`r`n") -Encoding ascii

Write-Host "Created $target"

}

# Three patches (use fresh, unused numbers)

New-AddFilePatch -Num 2701 -RelPath 'Assets/Editor/InboxCompileSignal\_2701.cs' -ContentLines @(

'#if UNITY\_EDITOR',

'using UnityEditor; using UnityEngine;',

'[InitializeOnLoad] public static class InboxCompileSignal\_2701 {',

' static InboxCompileSignal\_2701(){',

' try{ var stamp = System.IO.Path.Combine(Application.dataPath,"InboxPatches","CompileDone.stamp");',

' System.IO.Directory.CreateDirectory(System.IO.Path.GetDirectoryName(stamp));',

' System.IO.File.WriteAllText(stamp, System.DateTime.UtcNow.ToString("o"));',

' Debug.Log("[Inbox] 2701 wrote CompileDone.stamp"); }',

' catch(System.Exception ex){ Debug.LogWarning("[Inbox] 2701 failed: "+ex.Message);} } }',

'#endif'

)

New-AddFilePatch -Num 2702 -RelPath 'Assets/Editor/InboxAutoRefresh\_2702.cs' -ContentLines @(

'#if UNITY\_EDITOR',

'using UnityEditor; using UnityEngine;',

'[InitializeOnLoad] public static class InboxAutoRefresh\_2702 {',

' static InboxAutoRefresh\_2702(){ EditorApplication.update += DoOnce; }',

' static void DoOnce(){ EditorApplication.update -= DoOnce;',

' AssetDatabase.Refresh(ImportAssetOptions.Default); Debug.Log("[Inbox] Auto-refresh 2702 complete."); } }',

'#endif'

)

New-AddFilePatch -Num 2703 -RelPath 'Assets/InboxPatches/PATCH2703\_MARKER.txt' -ContentLines @(

'Patch 2703 marker.',

'Smoke test bundle for router v1.7.10.'

)

Write-Host "`nSmoke tests 2701–2703 created. Watch for UNITY wait/compiled then APPLY success." -ForegroundColor Cyan

if (Test-Path -LiteralPath $LogPath) { Get-Content -LiteralPath $LogPath -Wait -Tail 80 }

# Optional: Color Tail Viewer (copy/paste to colorize live log)

function Tail-ColorLog {

param([string]$Path = "$env:USERPROFILE\patch-router.log")

if(-not (Test-Path $Path)){ Write-Host "No log at $Path" -ForegroundColor Yellow; return }

Get-Content -Path $Path -Wait -Tail 1 | ForEach-Object {

$line = $\_

if($line -match '\bERROR|\bexception|\bfailed|\bfail|check-failed'){ $c='Red' }

elseif($line -match '\bWARN|pull-warning|push-warning|VERIFY|RECONCILE|UNITY wait'){ $c='Yellow' }

elseif($line -match '\bAPPLY success|\bUNITY compiled|\bLINT ok|READY detected'){ $c='Green' }

else { $c='Gray' }

Write-Host $line -ForegroundColor $c

}

}

# Tail-ColorLog

# Unity Editor Reference Snippets (Workflow B)

## Editor compile signal sample

#if UNITY\_EDITOR

using UnityEditor;

using UnityEngine;

[InitializeOnLoad]

public static class InboxCompileSignal\_Sample {

static InboxCompileSignal\_Sample() {

try {

var stamp = System.IO.Path.Combine(Application.dataPath, "InboxPatches", "CompileDone.stamp");

System.IO.Directory.CreateDirectory(System.IO.Path.GetDirectoryName(stamp));

System.IO.File.WriteAllText(stamp, System.DateTime.UtcNow.ToString("o"));

Debug.Log("[Inbox] Wrote CompileDone.stamp");

} catch (System.Exception ex) {

Debug.LogWarning("[Inbox] Failed to write CompileDone.stamp: " + ex.Message);

}

}

}

#endif

## Editor auto‑refresh helper

#if UNITY\_EDITOR

using UnityEditor;

using UnityEngine;

[InitializeOnLoad]

public static class InboxAutoRefresh\_Sample {

static InboxAutoRefresh\_Sample() { EditorApplication.update += DoOnce; }

private static void DoOnce() {

EditorApplication.update -= DoOnce;

AssetDatabase.Refresh(ImportAssetOptions.Default);

Debug.Log("[Inbox] Auto-refresh complete.");

}

}

#endif

# Failure Modes & Built‑in Fixes

• Missing #endif after #if UNITY\_EDITOR: router appends one and logs FIXUP + POSTCHECK.

• Truncated/partial new .cs: router reconstructs expected content from patch lines.

• Duplicate new‑file patches: router reconciles by writing expected content; commits as 'inbox: reconcile <patch>'.

• Detached HEAD/worktree loops: Ensure-BranchMain re‑attaches to main and pulls --rebase before apply.

• Remote pull/push issues: logged as warnings; do not block local success.

• Marker‑only patches: router logs UNITY skip and still declares success.

# Handover Rules (for the next assistant)

• Ask first: “What is the last patch number you downloaded?” Then start from the next number.

• Never assume a patch applied unless the log shows UNITY compiled followed by APPLY success.

• Always include start‑router and tail‑log in copy/paste blocks.

• If compile fails (no UNITY compiled), inspect Unity Console; router is waiting. Fix via a follow‑up patch.

# Troubleshooting & Tips

• If a patch isn't detected: confirm Downloads path, watch for READY in patch-router.log, ensure router window is running.

• If Unity crashes: A still has repo state; use git history to inspect or roll back.

• When switching to a new chat: upload this DOCX and ask for last downloaded patch number.

# Document Version

This file: Unity\_Automation\_Handover\_Full\_v4.docx — generated from merged content of v1 and v3, adapted with live fixes from this thread (router v1.7.10).