**Unity Development Automation Manual**

# Section 1: Project Vision

The ultimate goal is to create a hands-free Unity development pipeline where changes can be made, tested, and committed without requiring manual file management or tedious setup.  
  
The system is divided into two complementary workflows:  
- Workflow A (Foundation): Git + Patch Router Workflow  
 Provides stability, version control, and patch-based iteration.  
- Workflow B (Advanced): Realtime Unity Voice Workflow  
 Allows rapid, hands-free iteration through speech-to-code.  
  
Think of Workflow A as the engine block, and Workflow B as the turbocharger bolted on top.

# Section 2: Workflow A — Git + Patch Router Workflow

Purpose: Provide a stable backbone for automated iteration.

Core Components:

- Unity Auto-Commit Hook: A script that commits to Git automatically whenever you exit play mode.  
- Git Repository: Holds canonical source of truth.  
- Assistant: Reads repo + logs, generates patches.  
- Patch Router: Watches Downloads folder for new .patch files, applies them instantly to Unity.

Step-by-Step Loop:

1. Developer exits play mode.  
2. Unity commit hook automatically commits all changes.  
3. Developer tells assistant: 'Exited play mode.'  
4. Assistant fetches logs and diffs from Git.  
5. Assistant creates a .patch file representing the new changes.  
6. Developer downloads the .patch.  
7. Patch Router detects it instantly → applies into Unity project (without moving files).  
8. Developer re-enters play mode to test.

Key Benefits:

- Always safe (everything in Git).  
- Fully automated patch application.  
- Developer doesn’t need to manage files manually.

# Section 3: Workflow B — Realtime Unity Voice Workflow

Purpose: Remove all manual steps, enabling live Unity coding by voice.

Core Components:

- Whisper STT: Local speech-to-text for voice commands.  
- Python Bridge: Normalizes speech into raw C#.  
- Unity VoiceCommandReceiver.cs: Receives & recompiles live scripts.  
- Workflow A (Git + Patch Router): Still runs in the background to commit checkpoints.

Step-by-Step Loop:

1. Developer speaks a command (e.g., 'Add a script that moves the player upward when spacebar is pressed').  
2. Whisper transcribes to text.  
3. Python bridge formats as raw C# code.  
4. Unity VoiceCommandReceiver receives code → compiles & injects.  
5. Developer tests instantly in play mode.  
6. Upon exiting play mode, Workflow A auto-commits the changes to Git.

Key Benefits:

- Hands-free coding.  
- Changes are immediate and testable.  
- Git safety net ensures no spoken command is ever 'lost.'

# Section 4: How They Interlock

The two workflows nest together:  
- Workflow B feeds rapid voice-driven changes into Unity.  
- Workflow A captures and stabilizes them.  
- Assistant can always fall back to Workflow A if something breaks.  
  
Combined Flow Example:  
1. Developer uses voice → Unity script updates instantly.  
2. Developer tests → exits play mode.  
3. Workflow A commits → Assistant prepares .patch if needed.  
4. Developer downloads .patch → auto-applied by Patch Router.  
5. Development cycle continues seamlessly.

# Section 5: Visualizations

Diagram 1: Workflow A (Patch Cycle)  
- Unity Exit → Git Commit → Assistant Patch → Router → Unity Apply.  
  
Diagram 2: Workflow B (Voice Cycle)  
- Voice → Whisper → Python → Unity → Play Mode.  
  
Diagram 3: Combined Loop  
- Workflow B (Voice) nested inside Workflow A (Patch/Git).