AssumeUTXO Snapshot Distribution with Self-Repairing Integrity

Sync a Bitcoin Full Node in Minutes, Not Days

Oliver Odusanya

Independent Principal Investigator

`syncbitcoin.dev`

February 2026

Snapshot Height: 937,817

Abstract

Initial Block Download (IBD) for a Bitcoin full node takes 3–7 days on typical hardware — a significant barrier to node operation and decentralization. Bitcoin Core's `assumeUTXO` feature (merged in v28) allows bootstrapping from a trusted UTXO snapshot, but no public distribution mechanism provides the necessary integrity guarantees for a multi-gigabyte financial dataset.

We present a distribution package for block height **937,817** that combines **zstd compression** (8.8 GB $\rightarrow$ 7.9 GB), **dual cryptographic checksums** (SHA-256 + SHA-512), and **PAR2 forward error correction** (5% redundancy, capable of repairing $\sim$400 MB of corruption without re-downloading). This is, to our knowledge, the first Bitcoin UTXO snapshot distributed with forward error correction.

The package is fully self-contained and includes a **cross-platform installer orchestrator** — `INSTALL.sh` (macOS/Linux), `INSTALL.ps1` (Windows PowerShell), and `INSTALL.bat` (Windows launcher) — that guides the user through verification, decompression, configuration, and loading in a single terminal session. Platform-specific double-click launchers (`Install.command` for macOS, `Install.desktop` for Linux) provide a zero-command entry point. Alternatively, this PDF can be uploaded to AI coding assistants (Claude Code, Codex, etc.) to automate the entire process.

Introduction

The Sync Problem

Running a Bitcoin full node is the gold standard of trustless verification. However, syncing a new node from genesis requires downloading, validating, and indexing over 600,000 blocks — a process known as Initial Block Download (IBD). On consumer hardware, IBD takes:

- **3–5 days** on modern NVMe SSDs with broadband internet
- **5–10+ days** on spinning disks or limited bandwidth
- **Several hours** even on server-grade hardware

This is a real barrier to decentralization. Many users who intend to run a full node abandon the process when they realize the time commitment.

What is AssumeUTXO?

`assumeUTXO` is a feature merged into Bitcoin Core starting with v28.0, based on work proposed by James O'Beirne. It allows a node to load a serialized snapshot of the UTXO set at a specific block height and immediately begin validating new blocks from that point.

The key properties:

1. The node is **immediately usable** after loading the snapshot (within minutes, not days).
2. **Background validation** proceeds automatically: the node downloads and validates all historical blocks, eventually proving the snapshot was correct.
3. The security model is **no weaker** than existing `assumevalid` — the node will detect any discrepancy and reject the snapshot.

Why Existing Snapshots Fail

Unofficial UTXO snapshots have been shared via torrents and direct downloads, but they lack critical properties:

- **No error correction**: A single bit flip in an 8 GB file renders it useless. Users must re-download the entire file.
- **No standardized checksums**: Some distributions provide SHA-256 only, others provide nothing.
- **Torrent seeders die**: Files become unavailable when seeders go offline. No redundancy in the distribution layer.
- **No automation**: Loading a snapshot requires multiple manual steps, code changes, and recompilation.

Snapshot Specification

This package distributes the UTXO set at the following block:

Field	**Value**
Block height	937,817
`nchaintx`	1,314,106,044
`coins_written`	164,547,146
Source	Bitcoin Core v29.0.0 (fully synced mainnet; compatible with v30.2+)
Created	February 22, 2026

Block hash:

```
000000000000000000009025a9fdaa6b039c0df253b46331bdc12e7165c6a819
```

`hash_serialized_3`:

```
1e958331afe8162df7d8779e4a96e06bd3c1517c168f271c44e327fc040fda95
```

Integrity Architecture

Our distribution uses a three-layer integrity model. Each layer is independent and provides defense-in-depth.

Layer 1: Compression

The raw UTXO snapshot (8.8 GB) is compressed with **zstd** (Zstandard), Facebook's modern compression algorithm. zstd provides:

- 10% size reduction (8.8 GB → 7.9 GB)
- Decompression speed of ∼1.5 GB/s on modern hardware
- Built-in frame checksums for corruption detection during decompression

Layer 2: Dual Cryptographic Checksums

Both the compressed and raw files are checksummed with two independent hash algorithms:

File	SHA-256	SHA-512
`utxo-937817.dat`	`5ff2b5d5...996d50`	`87002076...dbc3b`
`utxo-937817.dat.zst`	`b6636f1a...e952f`	`cf5f6726...ce0f`

Using two independent algorithms provides:

- Defense against algorithm-specific attacks
- Cross-verification (if SHA-256 passes but SHA-512 fails, something is very wrong)

Layer 3: PAR2 Forward Error Correction

This is the key innovation. **PAR2** (Parity Archive Volume 2) adds Reed-Solomon error correction codes that can *repair* file corruption without re-downloading.

Parameter	Value
Redundancy	5%
Recovery capacity	∼400 MB of corruption
PAR2 index	39 KB
Recovery blocks	403 MB
Total overhead	∼5% of compressed file size

With PAR2, if the file is partially corrupted during download or storage:

1. `par2 verify` detects the exact corrupted regions
2. `par2 repair` reconstructs the original data from Reed-Solomon parity blocks
3. No re-download needed for corruption up to ∼400 MB

Comparison with Existing Approaches

Feature	Torrent	Direct DL	This Package
Compression	Sometimes	Sometimes	**Always (zstd)**
SHA-256 checksum	Rarely	Sometimes	**Yes**
SHA-512 checksum	No	No	**Yes**
Error correction	No	No	**PAR2 (5%)**
Self-repairing	No	No	**Yes**
Automated loading	No	No	**One-command script**
Permanent hosting	No (seeders die)	Maybe	**Internet Archive**

Installer Orchestrator

The installer is a cross-platform orchestrator that manages the entire snapshot lifecycle — from prerequisite detection through to a running Bitcoin node — in a single guided terminal session.

Platform Coverage

Platform	Installer	Launcher	How to Run
macOS	`INSTALL.sh`	`Install.command`	Terminal: `./INSTALL.sh` or double-click `Install.command`
Linux	`INSTALL.sh`	`Install.desktop`	Terminal: `./INSTALL.sh` or double-click `Install.desktop`
Windows	`INSTALL.ps1`	`INSTALL.bat`	PowerShell: `.\INSTALL.ps1` or double-click `INSTALL.bat`

The `INSTALL.bat` launcher auto-detects the best available shell on Windows (WSL, Git Bash, or native PowerShell) and delegates accordingly. The `.command` (macOS) and `.desktop` (Linux) launchers open a terminal window automatically — the user never needs to type a command.

Orchestrator Pipeline

Both `INSTALL.sh` and `INSTALL.ps1` implement the same seven-stage pipeline:

1. **Platform detection** — Identifies the OS, shell, and available tools. On macOS, detects Homebrew and whether `zstd`, `par2`, and `bitcoin-cli` are installed. On Linux, detects the package manager (`apt`, `dnf`, `pacman`). On Windows, detects `choco`/`scoop` for dependency installation.

2. **Prerequisite installation** — Offers to install missing dependencies automatically (e.g. `brew install zstd par2` on macOS, `sudo apt install zstd par2` on Linux, `choco install zstandard` on Windows).

3. **PAR2 verification & self-repair** — Runs `par2 verify` on the compressed snapshot. If corruption is detected, automatically runs `par2 repair` to reconstruct damaged blocks using Reed-Solomon parity data.

4. **Checksum verification & decompression** — Verifies SHA-256 and SHA-512 checksums of the compressed file, then decompresses with `zstd -d`. Verifies checksums of the decompressed file.

5. **Chainparams auto-patch** — Searches common paths for the Bitcoin Core source tree (`~/bitcoin`, installer directory, etc.). If the source is not found, offers to **clone it automatically** via `git clone -depth 1` (∼200 MB shallow clone). Once located, checks whether the block 937,817 entry already exists (idempotent). If not, inserts the entry into the `m_assumeutxo_data` map, creates a `.bak` backup, shows the diff, and offers to rebuild (`cmake -build build`). Auto-detects the `blockhash` format (`uint256{}` vs. `consteval_ctor`) to match the user's Bitcoin Core version. Falls back to a syntax-highlighted display if the user declines.

6. **Node mode selection** — Prompts the user to choose between **Pruned** (∼10–15 GB disk, recommended) or **Full Archival** (∼600+ GB disk). Both modes fully validate the blockchain with identical security.

7. **Snapshot loading** — Offers to load the snapshot into Bitcoin Core via `bitcoin-cli loadtxoutset`.

Terminal Experience

The installer uses ANSI escape codes for a polished terminal experience on all three platforms:

- **Animated splash screen** — Progressive reveal of the GhostMint wordmark in orange/gold block characters
- **Color-coded output** — Green for success, red for errors, orange for progress, dim gray for informational text
- **Real-time progress bars** — Percentage-based bars for decompression, verification, and loading stages
- **Boxed summaries** — Unicode box-drawing characters for step headers and completion summaries

Windows PowerShell supports ANSI escape codes natively since Windows 10 1511. The `INSTALL.ps1` script uses identical color codes and box-drawing characters as the bash version, ensuring visual parity across platforms.

Unattended Mode

For scripted deployments and CI/CD pipelines:

```
# macOS / Linux
./INSTALL.sh --non-interactive

# Windows PowerShell
.\INSTALL.ps1 -NonInteractive
```

Unattended mode skips all prompts, uses sensible defaults (pruned mode, default data directory), and exits with code 0 on success or non-zero on failure.

Usage Instructions

Prerequisites

Tool	Required?	macOS	Linux	Windows
`zstd`	Yes	`brew install zstd`	`apt install zstd`	`choco install zstandard`
`par2`	Recommended	`brew install par2`	`apt install par2`	`choco install par2cmdline`
`shasum`	Yes	Pre-installed	Pre-installed	Pre-installed (PS)
Bitcoin Core	For loading	bitcoincore.org/en/download		

Disk space: ∼16.7 GB during decompression (7.9 GB compressed + 8.8 GB decompressed). After cleanup: 8.8 GB.

Method 1: One-Click Installer (Recommended)

Download all files from the Internet Archive into a single directory, then:

```
# macOS / Linux
chmod +x INSTALL.sh && ./INSTALL.sh

# Windows (PowerShell)
.\INSTALL.ps1

# Or double-click the platform launcher:
#   macOS:   Install.command
#   Linux:   Install.desktop
#   Windows: INSTALL.bat
```

The installer orchestrates the full pipeline: prerequisite detection, PAR2 verification, checksum verification, decompression, automatic chainparams patching (with optional rebuild), node mode selection, and snapshot loading.

Method 2: Verify-and-Load Script

For users who prefer the lightweight script without the guided UI:

```
chmod +x verify-and-load.sh
./verify-and-load.sh
```

The script handles everything:

1. Checks prerequisites (`zstd`, `par2`, `shasum`)
2. Runs PAR2 verification; repairs corruption if detected
3. Verifies SHA-256 checksum of the compressed file

4. Decompresses with `zstd`
5. Verifies SHA-256 checksum of the decompressed file
6. Displays the required `chainparams.cpp` patch
7. Offers to load the snapshot into a running Bitcoin Core instance

Script options:

```
./verify-and-load.sh --no-load          # Verify + decompress only
./verify-and-load.sh --keep-compressed  # Keep .zst after decompression
./verify-and-load.sh --datadir /path    # Specify Bitcoin data directory
```

Method 3: Manual Step-by-Step

```
# 1. Verify PAR2 integrity (optional but recommended)
par2 verify utxo-937817.dat.zst.par2

# 2. Verify SHA-256 checksum
shasum -a 256 -c SHA256SUMS

# 3. Decompress
zstd -d utxo-937817.dat.zst

# 4. Verify decompressed file
shasum -a 256 utxo-937817.dat
# Should output: 5ff2b5d51604439f3e3cd9c24f9fd27be7297279563a5889033661099d996d50

# 5. Apply the chainparams patch (see Section 4.4)

# 6. Load into Bitcoin Core
bitcoin-cli -rpcclienttimeout=0 loadtxoutset /path/to/utxo-937817.dat
```

Method 3: AI-Assisted

Upload this PDF to **Claude Code**, **Codex**, or any AI coding assistant and say:

> *"Follow the instructions in this PDF to set up a Bitcoin node with the UTXO snapshot. Download the files, verify them, patch chainparams, build Bitcoin Core, and load the snapshot."*

The AI will have all the information it needs: download URLs, checksums, exact code patches, and verification commands.

Chainparams Patch

Bitcoin Core must be told about the snapshot height. **The installer handles this automatically**: it searches for `chainparams.cpp` in common locations (`~/bitcoin`, `/usr/local/src/bitcoin`, the installer directory). If the source is not found, it offers to **clone Bitcoin Core** via `git clone`

`-depth 1` (∼200 MB). It then detects the `blockhash` constructor format used by the installed version, inserts the entry with a `.bak` backup, and offers to rebuild. The patch is idempotent—if the entry already exists, it is skipped.

To apply manually, add this entry to `src/kernel/chainparams.cpp` inside the `CMainParams` constructor, in the `m_assumeutxo_data` map (after the last existing entry):

```
{
    .height = 937’817,
␣␣␣␣.hash_serialized␣=␣AssumeutxoHash{uint256{
␣␣␣␣␣␣␣␣"1e958331afe8162df7d8779e4a96e06bd3c1517c168f271c44e327fc040fda95"
␣␣␣␣}},
␣␣␣␣.m_chain_tx_count␣=␣1314106044,
␣␣␣␣.blockhash␣=␣uint256{
␣␣␣␣␣␣␣␣"00000000000000000000009025a9fdaa6b039c0df253b46331bdc12e7165c6a819"
␣␣␣␣},
}
```

Note: Some Bitcoin Core versions use `consteval_ctor(uint256{...})` instead of plain `uint256{...}`. The installer auto-detects the correct format.

Then rebuild Bitcoin Core:

```
cmake --build build -j$(nproc)
```

Verification & Trust Model

Independent Verification

Anyone with a fully synced Bitcoin Core node at or past block 937,817 can independently verify the snapshot:

```
bitcoin-cli gettxoutsetinfo muhash \
  "00000000000000000000009025a9fdaa6b039c0df253b46331bdc12e7165c6a819"
```

The output should include:

```
"hash_serialized_3": "1e958331afe8162df7d8779e4a96e06bd3c1517c168f271c44e327fc040fda95"
"txouts": 164547146
```

This proves the snapshot matches the Bitcoin blockchain’s state at that exact block, computed independently by your own node.

The AssumeUTXO Security Model

After loading the snapshot, Bitcoin Core operates in two phases:

1. **Immediate**: The node accepts the UTXO set and begins validating new blocks from height 937,817+. It can send/receive transactions and serve blocks to peers.
2. **Background**: The node downloads and fully validates all blocks from genesis to 937,817. If any discrepancy is found, the snapshot is rejected and the node rolls back.

This is functionally equivalent to the trust model of `assumevalid` (which has been in Bitcoin Core since v0.14): both optimize initial sync by deferring full validation, with the guarantee that full validation completes eventually.

Appendix

Complete File Manifest

File	**Description**	**Size**
Snapshot data		
`utxo-937817.dat.zst`	Compressed UTXO snapshot	7.9 GB
`utxo-937817.dat.zst.par2`	PAR2 index file	39 KB
`utxo-937817.dat.zst.vol000+100.par2`	PAR2 recovery blocks	403 MB
`SHA256SUMS`	SHA-256 checksums	168 B
`SHA512SUMS`	SHA-512 checksums	296 B
Installer orchestrator		
`INSTALL.sh`	One-click installer (macOS / Linux)	33 KB
`INSTALL.ps1`	One-click installer (Windows PowerShell)	33 KB
`INSTALL.bat`	Windows launcher (WSL / Git Bash fallback)	2.7 KB
`Install.command`	macOS double-click launcher	115 B
`Install.desktop`	Linux desktop launcher	246 B
Scripts & documentation		
`verify-and-load.sh`	Lightweight verify + load script	12 KB
`utxo-937817-README.txt`	Documentation	5 KB

Full Checksums

SHA-256:

```
5ff2b5d51604439f3e3cd9c24f9fd27be7297279563a5889033661099d996d50  utxo-937817.dat
b6636f1a3cdabfe2f2fa67a2cee07906714a880c680091b916acd40855ae952f  utxo-937817.dat.zst
```

SHA-512:

```
87002076f5dd80564657a1122c3de16e79353da54b7d1455605ccbaa4b049b01
53dd01686bd396b7b5d05d6fa3b4bb4c02c370bb19a9dea78917dad48a7dbc3b
  utxo-937817.dat

cf5f672600b89b4f430969da05af22fc70c510ebd84459e310008a9ac2aed691
1100a0ca9eba33d9081ae1400255757abc08fc88ff2cea34ace58e46f648ce0f
  utxo-937817.dat.zst
```

PAR2 Technical Details

PAR2 (Parity Archive Volume 2) implements Reed-Solomon error correction over GF(2^{16}). The compressed snapshot is divided into fixed-size "slices," and 100 recovery blocks are generated. The recovery blocks form a systematic code: given n data slices and k recovery slices, any n of the $n + k$ total slices suffice to reconstruct the original data.

In practical terms:

- Up to 5% of the file can be corrupted and fully repaired
- Corruption is detected at the slice level (not just file level)
- Repair is local: only corrupted slices are recomputed
- `par2 verify` runs in seconds; `par2 repair` runs in under a minute

Download Links

All files are permanently hosted on the Internet Archive:

```
https://archive.org/details/utxo-937817

Direct downloads:
https://archive.org/download/utxo-937817/utxo-937817.dat.zst
https://archive.org/download/utxo-937817/utxo-937817.dat.zst.par2
https://archive.org/download/utxo-937817/utxo-937817.dat.zst.vol000+100.par2
https://archive.org/download/utxo-937817/SHA256SUMS
https://archive.org/download/utxo-937817/SHA512SUMS
https://archive.org/download/utxo-937817/verify-and-load.sh
https://archive.org/download/utxo-937817/INSTALL.sh
https://archive.org/download/utxo-937817/INSTALL.ps1
https://archive.org/download/utxo-937817/INSTALL.bat
https://archive.org/download/utxo-937817/Install.command
https://archive.org/download/utxo-937817/Install.desktop
```

Website: `https://syncbitcoin.dev`