Version Control as Filesystem

September 2025

5 min read • 1,235 words

Themes: Technology Human Centered

Most people think of version control as a tool that sits on top of a filesystem—a way to track changes to files that already exist. But this misses the deeper revolution: version control systems like Git don't just track filesystems; they are filesystems. They represent a fundamental reimagining of what files and directories can be when time becomes a first-class dimension.

Traditional filesystems present an illusion of the eternal present. A file exists or it doesn't. Its contents are what they are right now. The past is archaeology—maybe you saved backups, maybe you didn't. The future doesn't exist until you write it.

But version control systems reveal files as they truly are: streams of change flowing through time.

The Temporal Dimension

Traditional filesystems optimize for the present moment. They're efficient at storing and retrieving the current state of files, but they treat history as an expensive luxury. Your file is what it is right now—anything else requires explicit backup strategies that feel bolted-on.

```
@dataclass
class TraditionalFile:
    """Files as snapshots in time."""
    contents: bytes
    modified_time: datetime

    def write(self, new_contents):
        # The past is destroyed
        self.contents = new_contents
        # Previous versions are gone forever

@dataclass
class VersionControlledFile:
    """Files as streams of change through time."""
    history: List[Commit]

    def read(self, point_in_time: Optional[str] = None):
        """Read file contents at any point in time."""
        return self.get_content_at(point_in_time or "HEAD")

    def write(self, contents: bytes, message: str):
        """Create new point in timeline, preserving all history."""
        self.history.append(Commit(contents, message))
        # Previous versions remain accessible
```

Version control systems solve the problem of time by making it explicit. Every change creates a new node in a graph of possibilities rather than destroying what came before.

Branching Realities

The most profound insight is that version control systems are parallel universe generators. When you create a branch, you're not just organizing work—you're creating an alternate timeline where different decisions were made.

```
class BranchingFilesystem:
    """A filesystem where reality can fork."""

    def create_branch(self, name: str):
        """Fork reality - create alternate timeline."""
        # Same starting point, infinite divergent possibilities
        return f"Created branch '{name}' - reality forked"

    def checkout(self, branch_name: str):
        """Switch to different version of reality."""
        # The filesystem contents change based on which reality you're in
        return f"Switched to reality '{branch_name}'"

    def merge(self, source_branch: str, target_branch: str):
        """Reconcile parallel timelines."""
        # Find the point where realities diverged
        # Attempt to unify both timelines
        return "Parallel realities successfully unified"
```

Traditional filesystems force you to make destructive choices. You can only be in one state at a time. Want to try a different approach? You have to abandon your current work or manually manage copies.

But branching filesystems let you explore multiple possibilities simultaneously. Each branch is a complete, coherent reality where a different set of decisions was made. You can switch between them instantly, compare them directly, and even merge insights from different timelines.

The Graph of All Possibilities

Version control systems reveal that filesystems are actually graphs—networks of relationships between different states of information over time.

In a traditional filesystem, you navigate through space—directories and files arranged in a hierarchy. In a version control filesystem, you navigate through time and possibility. Every commit represents a node in the graph of all possible states your files could be in.

This reveals something profound: filesystems aren't really about storing data. They're about modeling the evolution of information over time.

Distributed Reality

Git's distributed nature adds another layer to this reimagining. Traditional filesystems are centralized—there's one canonical version of reality, usually on one machine. But distributed version control systems create a mesh network of realities that can diverge, evolve independently, and synchronize when needed.

This distributed model means that no single machine holds the "true" version of the filesystem. Instead, truth emerges through consensus and synchronization. Multiple developers can work on the same codebase simultaneously, each maintaining their own complete copy of the entire history, merging changes when they align.

Content-Addressed Storage

Perhaps the most elegant insight in Git's design is content-addressed storage. Instead of files being identified by their names and locations, they're identified by the hash of their contents.

This approach eliminates the traditional relationship between files and paths. Two files with identical contents have the same hash regardless of their names or locations. Renames become trivial—you're not moving data, just updating pointers. Duplicated content is automatically shared.

It's a profound shift from location-based identity ("the file at `/home/user/document.txt`") to content-based identity ("the content with hash `a1b2c3d4...`").

The Philosophy of Immutability

Version control systems embody a philosophy of immutability. Once a commit is created, it never changes. History is append-only. This might seem inefficient compared to systems that modify files in place, but it enables powerful capabilities.

This immutable approach means that "undo" is not a special operation—it's just navigation. Every state that ever existed still exists. Bugs can be traced back to their exact introduction. Features can be rolled back without losing work.

Version Control as Truth

Ultimately, version control systems represent a different philosophy about truth and change. Traditional filesystems present the illusion that files are things—static objects that exist in specific places. Version control systems reveal that files are processes—streams of changes flowing through time.

```
def traditional_mindset():
    """Files as nouns - things that exist."""
    file = File("/path/to/document.txt")

    if file.exists():
        content = file.read()
        # This is what the file IS
    else:
        # The file doesn't exist
        pass

def version_control_mindset():
    """Files as verbs - processes of change."""
    file_history = Repository().get_file_history("/path/to/document.txt")

    # This is what the file has BECOME
    current_state = file_history.at_commit("HEAD")

    # This is how it GOT that way
    change_process = file_history.diff_from("initial_commit", "HEAD")

    # This is where it MIGHT go
    possible_futures = file_history.get_active_branches()
```

This shift from "files as things" to "files as processes" has implications beyond just software development. It suggests a way of thinking about information, knowledge, and even identity that embraces change and history rather than fighting against them.

Version control systems don't just track changes to code. They model how knowledge evolves, how decisions get made, how collaboration actually works. They're filesystems designed for reality—messy, branching, collaborative, and constantly changing.

Git for Humans: The Unrealized Potential

Despite these powerful concepts, Git remains trapped in developer culture. The interface is arcane, the mental models are programmer-centric, and the tooling assumes you think in commits and DAGs. But the underlying filesystem paradigm could revolutionize how anyone works with evolving information.

Imagine version control designed for writers, researchers, and creative professionals:

```
class HumanVersionControl:
    """Version control that speaks human language."""

    def save_draft(self, message: str = "Working on ideas"):
        """Writers don't commit - they save drafts."""
        return self.create_checkpoint(message)

    def try_different_approach(self, experiment_name: str):
        """Writers don't branch - they try different approaches."""
        return self.create_alternate_reality(experiment_name)

    def go_back_to(self, description: str):
        """Writers don't checkout - they go back to earlier versions."""
        matching_drafts = self.search_history(description)
        return self.restore_version(matching_drafts[0])

    def combine_best_parts(self, draft1: str, draft2: str):
        """Writers don't merge - they combine the best parts."""
        return self.assisted_integration(draft1, draft2)
```

The technology exists. The paradigm is proven. What's missing is the human-centered interface that makes temporal filesystems accessible to everyone who creates, not just those who code.

Someone could build "Git for Humans" - version control that helps writers track their thinking, researchers manage evolving hypotheses, and creators explore possibilities without fear of losing their work. The filesystem of the future isn't just for programmers.

"Time is the fire in which we burn, but version control is the phoenix that rises from the ashes of every destroyed file."

"In Git, nothing is lost, everything is transformed, and all possibilities coexist until the moment of checkout."