Website Versioning System - Explained Simply

Website Versioning System - Explained Simply

What Is Versioning?

Think of versioning like a library book system. Every time the website changes, we "stamp" it with a version number so we can:

- **Know what version is running** on the website
- **Track when changes were made**
- **Identify specific updates** if something needs to be fixed
- **Compare versions** between test and production sites

It's like having a receipt for every change made to the website.

Current Website Version

Live Version Information:

- **PMTNM Version**: {{api:version|version.pmtnm_version}}
- **Version Date**: {{api:version|version.pmtnm_version_date}}
- **Description**: {{api:version|version.pmtnm_version_desc}}
- **Site Version**: {{api:version|version.site_version}}
- **Git Commit**: {{api:version|version.git_commit_hash}}
- **Branch**: {{api:version|git.branch}}
- **Commit Date**: {{api:version|git.commit_date_formatted}}
- **Commit Message**: {{api:version|git.commit_message}}

Complete Version API Response:

{{api:version}}

Types of Versions

Our website uses three different types of version numbers, each serving a different purpose:

1. PMTNM Version (Manual Version)

What it is: A version number that we manually update when we make significant changes.

Format: `2025.12.17.01`

- **2025** = Year
- **12** = Month
- **17** = Day
- **01** = Build number (if multiple updates in one day)

When it changes: Only when we make important updates that we want to track officially.

Example: When we add a major new feature or fix a critical bug, we update this number.

Where to find it: In `includes/version.php` as `PMTNM_VERSION`

2. Site Version (Automatic Version)

What it is: An automatic version that changes whenever any code file is modified.

Format: `2025-12-17-a3f2b1c4`

- **2025-12-17** = Date of the most recent code change
- **a3f2b1c4** = Short hash based on when files were last modified

When it changes: Automatically whenever any PHP, CSS, or JavaScript file is updated.

How it works: The system scans all code files, finds the most recently modified one, and creates a version based on that date and a hash.

Why it's useful: You can see at a glance when the code was last changed, even if we forgot to update the manual version number.

Where to find it: Calculated automatically by `get_site_version()` function

3. Git Commit Hash (Code Repository Version)

What it is: A unique identifier for each set of changes saved in our code repository (Git).

Format:

- **Short:** `abbf329` (7 characters)
- **Full:** `abbf3292d23ec11496fc25b281d7c11f5529e94f` (40 characters)

When it changes: Every time code is saved (committed) to the repository.

How it works: Git creates a unique "fingerprint" for each set of changes. This hash is like a serial number - no two commits have the same hash.

Why it's useful:

- Can identify the exact set of changes deployed
- Can compare what's on the server vs. what's in the repository
- Can track down specific changes if needed

Where to find it: Stored in `data/.git_commit_hash.txt` after deployment

How Versioning Works

The Version Lifecycle

1. **Developer makes changes** to code files
2. **Changes are saved** to Git repository (creates commit hash)
3. **Code is deployed** to the website
4. **Version information is updated** automatically:
    - Site version updates based on file modification times
    - Git commit hash is stored on the server
    - PMTNM version is updated manually if it's a significant change

Version Tracking on the Server

When code is deployed, the system automatically:

- Stores the git commit hash in `data/.git_commit_hash.txt`
- Stores the commit message in `data/.git_commit_message.txt`
- Stores full commit info in `data/.git_commit_info.json`
- Calculates the site version based on code file dates

This means even if the server doesn't have access to the Git repository, we still know what version is running.

Accessing Version Information

Via Web Page

Visit: `/software_version.php`

This page shows:

- All version numbers in a user-friendly format
- Comparison between different version types
- Links to the API endpoint

Via API

Endpoint: `/api/version.php`

Method: GET (no authentication required)

Response: JSON format with all version information

Example Usage:

```
curl http://localhost:2025/api/version.php
```

Response includes:

- PMTNM version (manual)
- Site version (automatic)
- Git commit hash (short and full)
- Git branch name
- Commit date and message
- Environment information

In Markdown Documents

You can embed version information in any markdown document using:

- `{{api:version}}` - Full JSON response
- `{{api:version|version.pmtnm_version}}` - Specific value

This document uses these placeholders to show live version data!

Understanding Version Numbers

PMTNM Version Format

Example: `2025.12.17.01`

- **Year.Month.Day.Build**
- If multiple updates happen on the same day, increment the build number
- This is the "official" version number for releases

Site Version Format

Example: `2025-12-17-a3f2b1c4`

- **Date-Hash**
- Date is when the most recent code file was modified
- Hash is a short identifier based on the modification time
- Automatically updates when any code changes

Git Commit Hash Format

Example: `abbf329`

- 7-character short version (most common)
- 40-character full version (for precise identification)
- Hexadecimal characters (0-9, a-f)
- Unique to each set of changes

Why This Matters

For Administrators

- **Know what's deployed:** Quickly check if the latest code is on the server
- **Troubleshooting:** Identify which version has a bug
- **Audit trail:** Track when changes were made

- **Environment comparison:** Compare test vs. production versions

For Developers

- **Deployment verification:** Confirm code was deployed correctly
- **Version tracking:** See what code is running where
- **Debugging:** Match server version to repository commits
- **Change history:** Track when specific features were deployed

For Users

- **Transparency:** See that the site is actively maintained
- **Confidence:** Know that updates are tracked and managed
- **Support:** Provide version info when reporting issues

Version Comparison

You can compare versions between:

- **Local development** vs. **Test server** vs. **Production server**
- **Different time periods** to see what changed
- **Different branches** (test vs. main)

The version API makes it easy to programmatically check and compare versions across environments.

Frequently Asked Questions

Q: Why do we need three different version numbers?

A: Each serves a different purpose:

- **PMTNM Version:** Official releases and major milestones
- **Site Version:** Automatic tracking of any code changes
- **Git Commit Hash:** Precise identification of exact code state

Q: Which version should I reference when reporting issues?

A: All three! But the most useful are:

1. **PMTNM Version** - Easy to communicate
2. **Site Version** - Shows when code last changed
3. **Git Commit Hash** - For developers to find exact code

Q: How often does the site version change?

A: Every time any code file (PHP, CSS, JS) is modified and deployed. It's automatic and always up-to-date.

Q: Can I see the version history?

A: Yes! The git commit hash links to specific commits in the repository. You can also check the deployment logs to see version changes over time.

Q: What if the version API shows an error?

A: This usually means:

- The API endpoint is not accessible
- The version files haven't been created yet (first deployment)
- There's a server configuration issue

Check the error message for specific details.

Technical Details

Version Files Location

- `includes/version.php` - Version constants and functions
- `data/.git_commit_hash.txt` - Stored commit hash
- `data/.git_commit_message.txt` - Stored commit message
- `data/.git_commit_info.json` - Full commit information
- `data/.site_version_cache.json` - Cached site version

Version Calculation

Site Version:

1. Scans all code files (PHP, CSS, JS) in key directories
2. Finds the most recent modification time

3. Creates a hash from that timestamp
4. Formats as: `YYYY-MM-DD-hash`
5. Caches result for 1 hour

Git Commit Hash:

1. First tries to read from stored file (`data/.git_commit_hash.txt`)
2. If not found, tries to read from `.git` directory
3. If not found, tries `git` command
4. Falls back to null if Git is unavailable

Related Documentation

- Deployment Workflow Guide - How versions are updated during deployment
- Version API Documentation - Technical API details
- Version Hash Explanation - How version hashes work

Support

For questions about the versioning system, please use the contact form to reach the administrators.

This document automatically displays the current version information. Last updated: {{api:version|version.pmtnm_version_date}}