**Semantic Versioning Proposal**

**Abstract** Semantic Versioning (SemVer) is a systematic versioning scheme designed to convey meaningful information about software changes. It provides clear guidelines on incrementing version numbers based on the nature of changes (major, minor, or patch). This proposal introduces an automated Semantic Versioning approach using GitHub Actions to ensure consistent and reliable versioning in software development workflows.

**Problem Statement**

* Managing versioning dynamically within the repository can introduce inconsistencies and deployment issues.Manually managing versioning can lead to inconsistencies and deployment issues.
* Lack of a structured versioning mechanism can make tracking changes and backward compatibility difficult.
* Ensuring only specific branches merge into production or staging while maintaining release candidate versions needs automation.

**Solution** This proposal implements an automated Semantic Versioning workflow using GitHub Actions:

1. **Branch Control:** Merging restrictions are enforced to ensure a structured and stable release flow:

* **Production (prod) Branch:** Only staging or hotfix branches (hot/\*) are allowed to merge into prod. This ensures that production releases are controlled and go through a proper validation process.
* **Staging (staging) Branch:** Only develop or bugfix branches (bug/\*) can be merged into staging. This rule ensures that all features and bug fixes are reviewed and tested before being promoted to production.

This approach prevents accidental merges, maintains code stability, and enforces a systematic versioning and deployment workflow. 2. **Version Determination:** Extracts the latest tag and increments it based on PR title and body. If the PR title starts with (MAJOR), the major version will be incremented. If it starts with (MINOR), the minor version will be incremented. Otherwise, the patch version will be incremented by default. 3. **Tagging & Release:** Automatically assigns a new version number, pushes a Git tag, and creates a release with a changelog. 4. **Pre-release Handling:** Generates release candidates (RC) before final production releases. 5. **Changelog Generation:** Automatically compiles a changelog from merged PRs.

**Examples** Using an initial version of v1.0.0, the following table outlines version increments for different scenarios:

|  |  |  |  |
| --- | --- | --- | --- |
| **Source Branch** | **Target Branch** | **Change Type** | **Generated Version** |
| develop | staging | Patch Fix | v1.0.1-rc.1 |
| develop | staging | Patch Fix (multiple) | v1.0.1-rc.2, v1.0.1-rc.3, ... |
| develop | staging | Minor Feature | v1.0.1-rc.1 |
| develop | staging | Minor Feature (multiple) | v1.0.1-rc.2, v1.0.1-rc.3, ... |
| develop | staging | Major Change | v1.0.1-rc.1 |
| develop | staging | Major Change (multiple) | v1.0.1-rc.2, v1.0.1-rc.3, ... |
| staging | prod | Patch Fix Release | v1.0.1 |
| staging | prod | Minor Feature Release | v1.1.0 |
| staging | prod | Major Change Release | v2.0.0 |

**Conclusion** Automating Semantic Versioning simplifies release management, ensuring consistency and reducing manual errors. This approach enhances workflow efficiency and helps maintain clear versioning standards in software development.