**Ý nghĩa của số-phiên-bản 2.0.0**

**http://semver.org/**

**Tổng quan**

Số-phiên-bản dạng MAJOR.MINOR.PATCH, tăng theo quy luật:

1. Tăng MAJOR khi bạn có những thay đổi không tương thích với API cũ.
2. Tăng MINOR khi bạn thêm những chức năng mà vẫn duy trì sự tương-thích-ngược, và
3. Tăng PATCH khi bạn sửa những lỗi mà vẫn đảm bảo sự tương-thích-ngược.

Những “nhãn” cho bản tiền-xuất-bản và mô-tả bản-dựng[[1]](#footnote-1) cũng có thể thêm vào cú pháp này.

**Giới Thiệu**

Trong thế giới của việc quản lý phần mềm tồn tại một nơi đáng sợ gọi là “dependency hell – địa ngục của sự phụ thuộc”. Hệ thống của bạn phát triển càng nhanh và càng nhiều thứ bạn tích hợp vào phần mềm, thì càng giống như bạn thấy mình – ngày nào đó, ở trong một hố sâu tuyệt vọng.

Ở những hệ thống với nhiều sự phụ thuộc, phát-hành một phiên-bản chứa đựng nhiều tính năng mới có thể nhanh chóng dẫn bạn đến cơn ác mộng. Nếu các đặc-tả về các mối phụ thuộc này quá gò bó, thì bạn đang trong vùng nguy hiểm của việc khóa phiên-bản (chỉ việc không thể nâng cấp một chức-năng mà không phải phát-hành phiên-bản mới cho các gói nó phụ-thuộc). Nếu các mối phụ thuộc được định nghĩa quá lỏng-lẻo, khá rõ ràng bạn sẽ phải bận tâm nhiều về các phiên-bản (phải “dự đoán” về tính tương thích của các phiên-bản trong tương lai nhiều hơn mức độ hợp lý của nó). Địa ngục của việc phụ thuộc nằm nằm ở nơi việc khóa phiên-bản và/hoặc sự bận-tâm về các phiên-bản ngăn cản việc bạn đưa dự án phát triển dễ dàng và an toàn.

Để đưa ra một giải pháp cho vấn đề này, tôi đưa ra một tập quy-tắc và yêu-cầu đơn giản về cách đánh số và tăng số-phiên-bản. Những quy-tắc này dựa trên những phương-cách đã sử dụng phổ biến trong cả phần mềm nguồn-đóng hay nguồn-mở, nhưng không nhất thiết chỉ giới hạn trong các phương-cách đó. Để cỗ máy này hoạt động, trước hết bạn cần định nghĩa ra một API công-cộng. Thứ có thể chứa những tài liệu hoặc gián tiếp qua bản-thân mã-nguồn. Dù thế nào, API này phải rõ ràng và chính xác. Khi bạn đã định nghĩa ra một API công cộng, bạn chuyển tải sự thay đổi của nó thông qua việc tăng số phiên bản. Trong một phiên bản dạng X.Y.Z (major.minor.patch). Việc sửa những lỗi không ảnh hưởng đến API, tăng số patch. Thêm tính năng/thay đổi mà tương-thích-ngược với API, tăng số minor. Và khi thêm tính năng/thay đổi mà không tương-thích-ngược với API, tăng số major.

Tôi gọi hệ-thống như vậy là “ý nghĩa của số-phiên-bản”, theo mô tả này, số-phiên-bản và cách thay đổi của nó chỉ ra tính chất trong việc thay-đổi từ một phiên-bản sang phiên-bản tiếp theo của mã-nguồn bên dưới.

**ĐẶC TẢ Ý-NGHĨA SỐ-PHIÊN-BẢN (Sematic versioning spectification - SEMVER)**

Các từ **NÊN**, **KHÔNG-NÊN**, **BẮT-BUỘC**, **SẼ**, **KHÔNG-ĐƯỢC**, **SẼ-KHÔNG**, **NÊN**, **KHUYÊN-DÙNG**, **KHÔNG-NÊN**, **KHÔNG-KHUYẾN-KHÍCH**, **CÓ-THỂ**, **TÙY**, **TÙY-CHỌN** được sử dụng như mô tả tại [[RFC-2119](https://tools.ietf.org/html/rfc2119)] và bản dịch [[RFC-2119-vn](https://github.com/tapvanvn/rfc_for_vn/blob/master/RFC-2119.docx)].

1. Phần mềm sử dụng semver **PHẢI** khai-báo một API công-cộng. API này có thể được mô tả bởi bản thân mã nguồn hoặc quy định bởi tài liệu. Dù cách nào thì nó nên rõ ràng, chính xác và đầy đủ.
2. Một số-phiên-bản thông thường **PHẢI** có dạng X.Y.Z trong đó X,Y và Z là các số nguyên dương, và **KHÔNG-ĐƯỢC** chứa các số 0 đằng trước. X là phiên bản major (chính), Y là phiên bản minor (nhánh) và Z là phiên bản patch (vá). Mỗi thành phần **PHẢI** tăng tuần tự. Ví dụ 1.9.0 -> 1.10.0 -> 1.11.0.
3. Khi một gói đã đánh số phiên-bản được phát hành, nội dung của phiên bản đó **KHÔNG-ĐƯỢC** thay đổi. Bấy kì thay đổi nào cũng **PHẢI** được phát hành ở một phiên-bản mới.
4. Phiên bản Major số 0 (0.y.z) dành cho thiết lập quy trình phát triển. Bất cứ thứ gì cũng có thể thay đổi bất kì lúc nào. Không nên đánh giá API là đã ổn định.
5. Phiên bản 1.0.0 định nghĩa API công-khai. Theo cách các thay đổi của số-phiên-bản về sau dựa trên API này và sự thay đổi của nó.
6. Phiên bản patch Z (x.y.Z | x > 0) **PHẢI** được tăng nếu chỉ có những lỗi được sửa không làm API trở nên không tương thích. Một lỗi được sửa, được định nghĩa là một thay đổi nội tại để sửa chữa một hành vi không hoạt động đúng.
7. Phiên bản minor Y (x.Y.z | x > 0) **PHẢI** được tăng nếu có chức năng mới tương-thích-ngược được giới thiệu trong API công-khai. Nó PHẢI được tăng nếu có bất kì API công cộng nào bị đánh dấu là “sắp-loại-bỏ”. Nó **CÓ-THỂ** được tăng nếu một chức năng mới có giá trị hoặc một cải tiến nội tại được giới thiệu. Nó **CÓ-THỂ** bao gồm những thay đổi ở cấp độ của bản patch. Số-phiên-bản patch **PHẢI** đánh về 0 khi tăng số phiên bản minor.
8. Phiên bản major X (X.y.z | X > 0) PHẢI được tăng nếu bất kỳ thay đổi không tương-thích-ngược nào trong API công-cộng được giới thiệu. Nó **CÓ-THỂ** bao gồm các thay đổi ở cấp độ của bản patch hay minor. Số-phiên-bản của patch và minor **PHẢI** được đánh về 0 khi tăng số phiên bản major.
9. Một bản tiền-xuất-bản **CÓ-THỂ** được đánh dấu bằng cách thêm một dấu “-“ và một chuỗi định danh ngăn cách bởi dấu chấm ngay sau số-phiên-bản patch. Chuỗi định-danh này **PHẢI** được kết hợp bởi các kí tự số và chữ trong bảng mã ASCII và dấu “-“ [0-9A-Za-z-]. Chuỗi định-danh **KHÔNG-ĐƯỢC** để trống. Số trong chuỗi định danh **KHÔNG-ĐƯỢC** bắt đầu bởi 0. Các phiên bản tiền-xuất-bản có độ ưu-tiên thấp hơn các phiên bản thông thường. Một phiên bản tiền-xuất-bản ngầm định rằng phiên bản đó không ổn định và không được đảm bảo về các ràng buộc về tính tương thích như các phiên bản thông thường. Ví dụ: 1.0.0-alpha, 1.0.0-alpha.1, 1.0.0-0.3.7, 1.0.0-x.7.z.92.
10. Thuộc tính của bản-dựng **CÓ-THỂ** được đánh dấu bằng cách thêm vào một dấu “+” và theo sau là chuỗi định-danh ngăn cách bởi dấu chấm theo sau ngay số-phiên-bản patch hoặc tiền-xuất-bản. Chuỗi định-danh **PHẢI** được kết hợp bởi các kí tự, số và dấu “-“ theo bảng mã ASCII [0-9A-Za-z-]. Chuỗi định-danh **KHÔNG-ĐƯỢC** để trống. Thuộc tính của bản dựng **NÊN** được bỏ qua khi xem xét độ ưu-tiên của phiên bản. Thực tế hai phiên bản chỉ khác nhau phần thuộc-tính có cùng độ ưu-tiên. Ví dụ: 1.0.0-alpha+001, 1.0.0+20130313144700, 1.0.0-beta+exp.sha.5114f85.
11. Độ ưu-tiên xác định cách so sánh hai phiên bản với nhau khi sắp xếp. Độ ưu-tiên PHẢI được tính toán bằng cách tách các phần của phiên bản ra thành major, minor, patch và định-danh bản tiền-xuất-bản theo thứ tự (thuộc tính của bản-dựng không ảnh hưởng tới độ ưu-tiên). Độ ưu-tiên được xác định bằng khác biệt đầu tiên khi so sánh mỗi thành phần này từ trái sang phải, như sau: Major, minor, và patch luôn so sánh bằng cách so sánh số. Ví dụ: 1.0.0 < 2.0.0 < 2.1.0 < 2.1.1. Khi Major, minor và patch bằng nhau, một phiên bản tiền-xuất-bản sẽ có độ ưu-tiên thấp hơn phiên bản thông thường. Ví dụ: 1.0.0-alpha < 1.0.0. Độ ưu-tiên của hai bản tiền-phát-hành với cùng phiên bản major, minor, patch PHẢI được chỉ ra bằng cách so sánh mỗi thành phần tách biệt bởi dấu chầm trong chuỗi định danh từ trái sang phải cho tới khi phát hiện sự khác nhau, như sau: chuỗi định danh chỉ chứa số sẽ được so sánh bằng cách so sánh số, với chuỗi định danh chứa kí tự và dấu “-“ sẽ được so sánh theo thứ tự ASCII. Định danh bằng số luôn có độ ưu-tiên thấp hơn định danh có chứa kí tự. Định danh có nhiều trường hơn có độ ưu-tiên cao hơn, nếu tất cả các mục so sánh đều bằng nhau. Ví dụ: 1.0.0-alpha < 1.0.0-alpha.1 < 1.0.0-alpha.beta < 1.0.0-beta < 1.0.0-beta-2 < 1.0.0-beta.11 < 1.0.0-rc.1 < 1.0.0.

About : The Semantic Versioning specification is authored by Tom Preston-Werner, inventor of Gravatars and cofounder of GitHub.

1. Không ưng ý từ này lắm nhưng chưa tìm ra tự phù hợp hơn [↑](#footnote-ref-1)