

العربية (ar) </lang/ar>	Български (bg) </lang/bg>	català (ca) </lang/ca>	čeština (cs) </lang/cs>
Dansk (da) </lang/da>	Deutsch (de) </lang/de>	Ελληνικά (el) </lang/el>	English (en) </>
español (es) </lang/es>	فارسی (fa) </lang/fa>	français (fr) </lang/fr>	עברית (he) </lang/he>
हिन्दी (hin) </lang/hin>	hrvatski (hr) </lang/hr>	magyar (hu) </lang/hu>	Հայերեն (hy) </lang/hy>
Bahasa Indonesia (id) </lang/id>	italiano (it) </lang/it>	日本語 (ja) </lang/ja>	
ქართული (ka) </lang/ka>	taqbaylit (kab) </lang/kab>	한국어 (ko) </lang/ko>	
Nederlands (nl) </lang/nl>	polski (pl) </lang/pl>	português brasileiro (pt-BR) </lang/pt-BR>	
Română (ro) </lang/ro>	русский (ru) </lang/ru>	slovensky (sk) </lang/sk>	
slovenščina (sl) </lang/sl>	srpski (sr) </lang/sr>	svenska (sv) </lang/sv>	Türkçe (tr) </lang/tr>
українська (uk) </lang/uk>	Tiếng Việt (vi) </lang/vi>	简体中文 (zh-CN) </lang/zh-CN>	
繁體中文 (zh-TW) </lang/zh-TW>			

2.0.0 </lang/pt-BR/spec/v2.0.0.html>	2.0.0-rc.1 </lang/pt-BR/spec/v2.0.0-rc.1.html>
--	--

Versionamento Semântico 2.0.0

Sumário

Dado um número de versão MAJOR.MINOR.PATCH, incremente a:

- 1. versão Maior(MAJOR): quando fizer mudanças incompatíveis na API,
- 2. versão Menor(MINOR): quando adicionar funcionalidades mantendo compatibilidade, e
- 3. versão de Correção(PATCH): quando corrigir falhas mantendo compatibilidade.

Rótulos adicionais para pré-lançamento(pre-release) e metadados de construção(build) estão disponíveis como extensão ao formato MAJOR.MINOR.PATCH.

Introdução

No mundo de gerenciamento de software existe algo terrível conhecido como inferno das dependências (“dependency hell”). Quanto mais o sistema cresce, e mais pacotes são adicionados a ele, maior será a possibilidade de, um dia, você encontrar-se neste poço de desespero.

Em sistemas com muitas dependências, lançar novos pacotes de versões pode se tornar rapidamente um pesadelo. Se as especificações das dependências são muito amarradas você corre o risco de um bloqueio de versão (A falta de capacidade de atualizar um pacote sem ter de liberar novas versões de cada pacote dependente). Se as dependências são vagamente especificadas, você irá inevitavelmente ser mordido pela ‘promiscuidade da versão’ (assumindo compatibilidade com futuras versões mais do que é razoável). O inferno das dependências é onde você está quando um bloqueio de versão e/ou promiscuidade de versão te impede de seguir em frente com seu projeto de maneira fácil e segura.

Como uma solução para este problema proponho um conjunto simples de regras e requisitos que ditam como os números das versões são atribuídos e incrementados.

Essas regras são baseadas em, mas não necessariamente limitadas às, bem difundidas práticas comumente em uso tanto em softwares fechados como open-source. Para que este sistema funcione, primeiro você precisa declarar uma API pública. Isto pode consistir de documentação ou ser determinada pelo próprio código. De qualquer maneira, é importante que esta API seja clara e precisa. Depois de identificada a API pública, você comunica as mudanças com incrementos específicos para o seu número de versão. Considere o formato de versão X.Y.Z (Maior.Menor.Correção). Correção de falhas (bug fixes) que não afetam a API, incrementa a versão de Correção, adições/alterações compatíveis com as versões anteriores da API

incrementa a versão Menor, e alterações incompatíveis com as versões anteriores da API incrementa a versão Maior.

Eu chamo esse sistema de "Versionamento Semântico". Sob este esquema, os números de versão e a forma como eles mudam transmitem o significado do código subjacente e o que foi modificado de uma versão para a próxima.

Especificação de Versionamento Semântico (SemVer)

As palavras-chaves "DEVE", "NÃO DEVE", "OBRIGATÓRIO", "DEVERÁ", "NÃO DEVERÁ", "PODEM", "NÃO PODEM", "RECOMENDADO", "PODE" e "OPCIONAL" no presente documento devem ser interpretados como descrito na [RFC 2119 <http://tools.ietf.org/html/rfc2119>](http://tools.ietf.org/html/rfc2119).

1. Software usando Versionamento Semântico DEVE declarar uma API pública. Esta API poderá ser declarada no próprio código ou existir estritamente na documentação, desde que seja precisa e compreensiva.
2. Um número de versão normal DEVE ter o formato de X.Y.Z, onde X, Y, e Z são inteiros não negativos, e NÃO DEVE conter zeros à esquerda. X é a versão Maior, Y é a versão Menor, e Z é a versão de Correção. Cada elemento DEVE aumentar numericamente. Por exemplo: 1.9.0 -> 1.10.0 -> 1.11.0.
3. Uma vez que um pacote versionado foi lançado(released), o conteúdo desta versão NÃO DEVE ser modificado. Qualquer modificação DEVE ser lançado como uma nova versão.
4. No início do desenvolvimento, a versão Maior DEVE ser zero (0.y.z). Qualquer coisa PODE mudar a qualquer momento. A API pública NÃO DEVE ser considerada estável.
5. Versão 1.0.0 define a API como pública. A maneira como o número de versão é incrementado após este lançamento é dependente da API pública e como ela muda.
6. Versão de Correção Z (x.y.z | x > 0) DEVE ser incrementado apenas se mantiver compatibilidade e introduzir correção de bugs. Uma correção de bug é definida como uma mudança interna que corrige um comportamento incorreto.
7. Versão Menor Y (x.y.z | x > 0) DEVE ser incrementada se uma funcionalidade nova e compatível for introduzida na API pública. DEVE ser incrementada se qualquer funcionalidade da API pública for definida como descontinuada. PODE ser incrementada se uma nova funcionalidade ou melhoria substancial for introduzida dentro do código privado. PODE incluir mudanças a nível de correção. A versão de Correção DEVE ser redefinida para 0(zero) quando a versão Menor for incrementada.
8. Versão Maior X (X.y.z | X > 0) DEVE ser incrementada se forem introduzidas mudanças incompatíveis na API pública. PODE incluir alterações a nível de versão Menor e de versão de Correção. Versão de Correção e Versão Menor DEVEM ser redefinidas para 0(zero) quando a versão Maior for incrementada.
9. Uma versão de Pré-Lançamento (pre-release) PODE ser identificada adicionando um hífen (dash) e uma série de identificadores separados por ponto (dot) imediatamente após a versão de Correção. Identificador DEVE incluir apenas caracteres alfanuméricos e hífen [0-9A-Za-z-]. Identificador NÃO DEVE ser vazio. Indicador numérico NÃO DEVE incluir zeros à esquerda. Versão de Pré-Lançamento tem precedência inferior à versão normal a que está associada. Uma versão de Pré-Lançamento (pre-release) indica que a versão é instável e pode não satisfazer os requisitos de compatibilidade pretendidos, como indicado por sua versão normal associada. Exemplos: 1.0.0-alpha, 1.0.0-alpha.1, 1.0.0-0.3.7, 1.0.0-x.7.z.92.
10. Metadados de construção(Build) PODE ser identificada por adicionar um sinal de adição (+) e uma série de identificadores separados por ponto imediatamente após a Correção ou Pré-Lançamento. Identificador DEVE ser composto apenas por caracteres alfanuméricos e hífen [0-9A-Za-z-]. Identificador NÃO DEVE ser vazio. Metadados de construção PODEM ser ignorados quando se determina a versão de precedência. Assim, duas versões que diferem apenas nos metadados de construção, têm a mesma precedência. Exemplos: 1.0.0-alpha+001, 1.0.0+20130313144700, 1.0.0-beta+exp.sha.5114f85.

11. A precedência refere como as versões são comparadas uma com a outra quando solicitado.
12. A precedência DEVE ser calculada separando identificadores de versão em Maior, Menor, Correção e Pré-lançamento, nesta ordem. (Metadados de construção não faz parte da precedência).
 1. A precedência é determinada pela primeira diferença quando se compara cada identificador da esquerda para direita, como se segue: Versões Maior, Menor e Correção são sempre comparadas numericamente.

Exemplo: $1.0.0 < 2.0.0 < 2.1.0 < 2.1.1$.
 2. Quando Maior, Menor e Correção são iguais, a versão de Pré-Lançamento tem precedência menor que a versão normal.

Exemplo: $1.0.0\text{-alpha} < 1.0.0$.
 3. A precedência entre duas versões de Pré-lançamento com mesma versão Maior, Menor e Correção DEVE ser determinada comparando cada identificador separado por ponto da esquerda para direita até que seja encontrada diferença da seguinte forma:
 1. Identificadores consistindo apenas dígitos são comparados numericamente.
 2. Identificadores com letras ou hífen são comparados lexicalmente na ordem de classificação ASCII.
 3. Identificadores numéricos sempre têm menor precedência do que os não numéricos.
 4. Um conjunto maior de campos de pré-lançamento tem uma precedência maior do que um conjunto menor, se todos os identificadores anteriores são iguais.
Exemplo: $1.0.0\text{-alpha} < 1.0.0\text{-alpha}.1 < 1.0.0\text{-alpha}.beta < 1.0.0\text{-beta} < 1.0.0\text{-beta}.2 < 1.0.0\text{-beta}.11 < 1.0.0\text{-rc}.1 < 1.0.0$.

Por que usar Versionamento Semântico?

Esta não é uma ideia nova ou revolucionária. De fato, você provavelmente já faz algo próximo a isso. O problema é que "próximo" não é bom o bastante. Sem a aderência a algum tipo de especificação formal, os números de versão são essencialmente inúteis para gerenciamento de dependências. Dando um nome e definições claras às ideias acima, fica fácil comunicar suas intenções aos usuários de seu software. Uma vez que estas intenções estão claras, especificações de dependências flexíveis (mas não tão flexíveis) finalmente podem ser feitas.

Um exemplo simples vai demonstrar como o Versionamento Semântico pode fazer do inferno de dependência uma coisa do passado. Considere uma biblioteca chamada "CaminhaoBombeiros". Ela requer um pacote versionado dinamicamente chamado "Escada". Quando CaminhaoBombeiros foi criado, Escada estava na versão 3.1.0. Como CaminhaoBombeiros utiliza algumas funcionalidades que foram inicialmente introduzidas na versão 3.1.0, você pode especificar, com segurança, a dependência da Escada como maior ou igual a 3.1.0 porém menor que 4.0.0. Agora, quando Escada versão 3.1.1 e 3.2.0 estiverem disponíveis, você poderá lançá-los ao seu sistema de gerenciamento de pacote e saberá que eles serão compatíveis com os softwares dependentes existentes.

Como um desenvolvedor responsável você irá, é claro, querer certificar-se que qualquer atualização no pacote funcionará como anunciado. O mundo real é um lugar bagunçado; não há nada que possamos fazer quanto a isso senão sermos vigilantes. O que você pode fazer é deixar o Versionamento Semântico lhe fornecer uma maneira sensata de lançar e atualizar pacotes sem precisar atualizar para novas versões de pacotes dependentes, salvando-lhe tempo e aborrecimento.

Se tudo isto soa desejável, tudo que você precisar fazer para começar a usar Versionamento Semântico é declarar que você o está usando e então, seguir as regras. Adicione um link para este website no seu README para que outros saibam as regras e possam beneficiar-se delas.

FAQ

Como devo lidar com revisões na fase 0.y.z de desenvolvimento inicial?

A coisa mais simples a se fazer é começar sua versão de desenvolvimento inicial em 0.1.0 e, então, incrementar a uma versão 'menor' em cada lançamento subsequente.

Como eu sei quando lançar a versão 1.0.0?

Se seu software está sendo usado em produção, ele já deve ser provavelmente 1.0.0. Se você possui uma API estável a qual usuários passaram a depender, deve ser 1.0.0. Se você está se preocupando bastante com compatibilidade com versões anteriores, já deve ser 1.0.0.

Isto não desencoraja o desenvolvimento ágil e iteração rápida?

A versão Maior zero tem o foco exatamente no desenvolvimento rápido. Se você está mudando a API todo dia, provavelmente você está na versão 0.y.z ou num branch separado de desenvolvimento, trabalhando numa próxima versão Maior.

Se mesmo a menor mudança incompatível com a API pública requer aumento da versão maior, não vou acabar na versão 42.0.0 muito rapidamente?

Esta é uma questão de desenvolvimento responsável e conhecimento antecipado. Mudanças incompatíveis não devem ser levemente introduzidas para o software que tem um monte de código dependente. O custo que deve ser incorrido para atualizar pode ser significativo. Tendo que aumentar a versão maior para lançar mudanças incompatíveis, significa que você pensará no impacto das suas mudanças, e avaliará a relação de custo/benefício envolvida.

Documentar toda a API pública dá muito trabalho!

É sua responsabilidade como desenvolvedor profissional documentar corretamente o software que será usado por outros. Gerenciar a complexidade de software é uma parte muito importante para manter o projeto eficiente, e isto é difícil de fazer se ninguém sabe como usá-lo ou que métodos são seguros de chamar. A longo prazo, Versionamento Semântico e a insistência em uma API pública bem definida podem deixar tudo e todos funcionando suavemente.

O que eu faço se, acidentalmente, liberar uma mudança incompatível com versões anteriores como uma versão menor (minor version)?

Assim que você perceber que quebrou a especificação de versionamento semântico, conserte o problema e lance uma nova versão menor, que corrige o problema e restaura a compatibilidade. Mesmo sob esta circunstância, é inaceitável modificar versões lançadas. Se for apropriado, documente a versão ofensiva e informe seus usuários do problema de forma que eles fiquem cientes da versão em questão.

O que devo fazer se eu atualizar minhas próprias dependências sem modificar a API pública?

Isso seria considerado compatível, uma vez que não afeta a API pública. Software que depende explicitamente da mesmas dependências que seu pacote, deve ter sua própria especificação de dependência e o autor notificará quaisquer conflitos. Para determinar se a mudança é a nível de correção ou modificação de nível menor dependente se você atualizou suas dependências a fim de corrigir um bug ou introduzir nova funcionalidade. Eu normalmente esperaria código adicional para última instância, caso em que é obviamente um incremento no nível menor.

E se eu alterei inadvertidamente a API pública de forma incompatível com a mudança no número de versão (ex.: o código incorretamente introduz uma grande mudança incompatível em liberação de um patch)

Use o bom senso. Se você tem um público enorme que será drasticamente impactado pela mudança de comportamento de volta para o que a API pública pretendida, então pode ser melhor realizar um lançamento de uma versão maior, mesmo que a correção pudesse ser considerada estritamente uma versão de correção. Lembre-se, Versionamento Semântico trata de transmitir o conhecimento das mudanças ocorridas na versão. Se estas mudanças são importantes para seus usuários, utilize o número da versão para informá-los.

Como devo lidar com descontinuação de funcionalidades?

Descontinuar funcionalidades é um processo comum no desenvolvimento de software e muitas vezes é necessário para haver progresso. Quando você descontinua partes de sua API pública, você deve fazer duas coisas: (1) atualizar sua documentação, para que os usuários saibam das mudanças, (2) lançar uma versão Menor anunciando a descontinuação. Antes de remover completamente a funcionalidade em uma versão Maior deve haver ao menos uma versão Menor que possui a descontinuação anunciada, fazendo com que os usuários realizem uma transição tranquila para a nova API.

O SemVer tem um limite de tamanho para string de versão?

Não, mas use o bom senso. Uma string de versão com 255 caracteres por exemplo, provavelmente é um exagero. Porém, sistemas específicos podem definir seus próprios limites para o tamanho da string.

Sobre

A Especificação da Semântica de Versionamento é autoria de [Tom Preston-Werner](http://tom.preston-werner.com) <<http://tom.preston-werner.com>>, criador do Gravatar e co-fundador do GitHub.

A tradução deste documento para Português-Brasil foi iniciada de forma colaborativa pela [Wend Tecnologia](https://github.com/wendtecnologia) <<https://github.com/wendtecnologia>> através de [Walker de Alencar Oliveira](https://github.com/walkeralecar) <<https://github.com/walkeralecar>> e teve a participação de:

- [William G. Comnisky](https://github.com/wcomnisky) <<https://github.com/wcomnisky>>
- [Rafael Sirotheau](https://github.com/rafasirotheau) <<https://github.com/rafasirotheau>>
- [Arthur Almeida](https://github.com/arthuralemeidap) <<https://github.com/arthuralemeidap>>
- [Alberto Guimarães Viana](https://github.com/albertogviana) <<https://github.com/albertogviana>>
- [Rafael Lúcio](https://github.com/poste9) <<https://github.com/poste9>>
- Josiel Rocha
- Alessandro Leite
- Vinícius Assef
- [Silas Ribas Martins](https://github.com/silasrm) <<https://github.com/silasrm>>
- [Lucas Honda Tonini](https://github.com/LucasHT22) <<https://github.com/LucasHT22>>

Toda colaboração na tradução pode ser acompanhada no link: <http://pad.okfn.org/p/Fh9hjBPVu9>

Caso queira deixar sua opinião, por favor [abra uma issue no GitHub](https://github.com/semver/semver/issues) <<https://github.com/semver/semver/issues>>.

Licença

[Creative Commons — CC BY 3.0](http://creativecommons.org/licenses/by/3.0/) <<http://creativecommons.org/licenses/by/3.0/>>