# Versionamento Semântico 2.0.0

## Por que usar Versionamento Semântico?

Esta não é uma ideia nova ou revolucionária. De fato, você provavelmente já fez 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 essas 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-las ao seu sistema de gerenciamento de pacote e saberá que elas 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.

https://semver.org/lang/pt-BR/

# O que significa 2.0.0 no contexto do SemVer?

No SemVer, as versões são representadas por três números separados por pontos, no formato **MAJOR.MINOR.PATCH**:

1. **MAJOR** (2 no caso do "2.0.0"): É incrementado quando você faz mudanças incompatíveis na API, ou seja, mudanças que quebram a compatibilidade com versões anteriores.
2. **MINOR** (0 no caso do "2.0.0"): É incrementado quando você adiciona funcionalidades de maneira retrocompatível. Isso significa que o novo recurso não quebra as funcionalidades existentes.
3. **PATCH** (0 no caso do "2.0.0"): É incrementado quando você faz correções de bugs de maneira retrocompatível.

## Interpretação do número de versão "2.0.0"

- **2**: Houve uma mudança significativa na API, tornando a nova versão incompatível com a anterior (1.x.x).
- **0**: Não foram adicionadas novas funcionalidades desde a última versão principal, indicando que esta é a primeira versão da nova linha 2.x.x.
- **0**: Não foram aplicadas correções de bugs na versão 2.0.0, ou seja, é a primeira liberação da nova versão major.

## Exemplo Prático

Se a versão anterior fosse 1.5.3, uma atualização para 2.0.0 indicaria que houve mudanças fundamentais no software, que podem incluir novas funcionalidades ou modificações que não são compatíveis com a versão 1.x.x, exigindo que os desenvolvedores que utilizam essa biblioteca ou software ajustem seu código para funcionar corretamente com a nova versão.