Repositório - Melhores Práticas

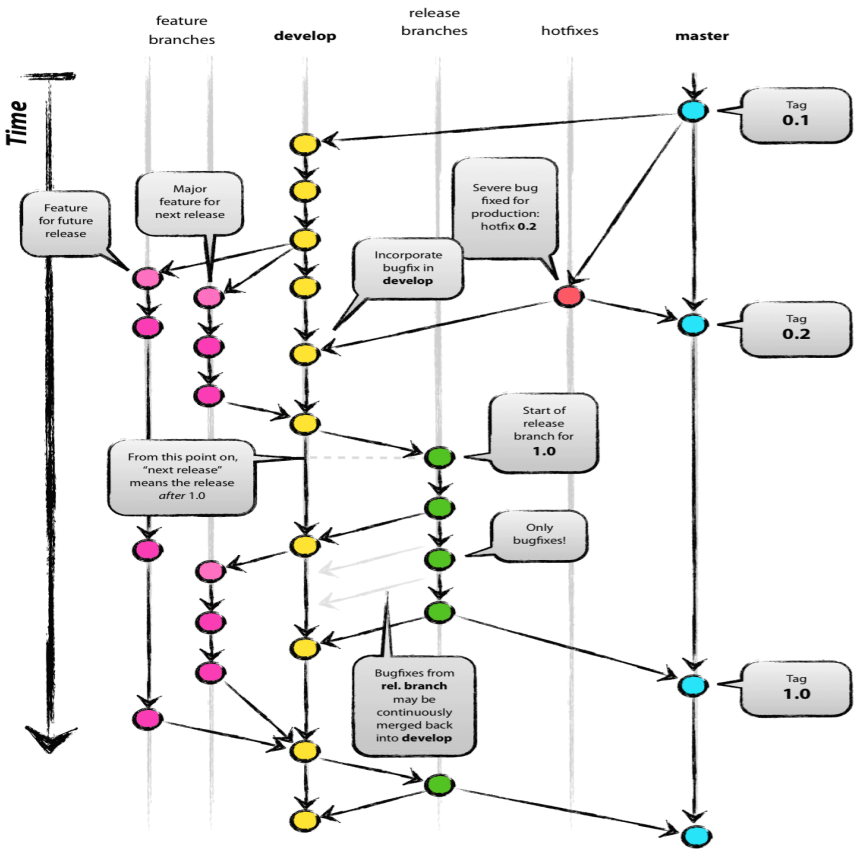
1.0	2020-03-03	Criação do Documento	Mauricio

Tipos de *Branchs*

1. Principais (não expiram nunca)
    a. Master
    b. Develop

2. Suporte (só vivem enquanto se dá suporte a determinada versão)
    a. Releases

3. Temporárias (tempo de vida finito, apenas enquanto seu objetivo não é atingido)
    a. Features
    b. Hot fixes



fonte: [2] – *Branch model*

Ação	Comando
Criar uma nova branch	`git checkout -b myfeature develop` `*cria uma nova branch "myfeature" a partir de develop`

Incorporando as mudanças na branch develop	`git checkout develop //*(muda para a branch 'develop')`
	`git merge --no-ff myfeature //*(faz o merge como um novo commit)`
	`git branch -d myfeature //*(remove a branch "myfeature")`
	`git push origin develop //*(sincroniza a branch 'develop')`

Tornando a release a versão oficial	`git checkout master //* (Switched to branch 'master')`
	`git merge --no-ff release-1.2 //* (Merge made by recursive.)`
	`git tag -a 1.2`
	`git checkout develop // *(fazendo a mesma coisa na branch develop)`
	`git merge --no-ff release-1.2`
	`git branch -d release-1.2 //* (remove a branch 1.2 )`

Versionamento

Padrão de mercado: Major.Minor.Patch.Build

Neste momento, nós não controlaremos o Build, portanto, nossos produtos terão apenas os 3 primeiros componentes: *Major.Minor.Patch*;

- **MAJOR**: versão quando você faz alterações incompatíveis da API. Exemplo: a versão 2 não tem a obrigação de ser compatível com a versão 1;
- **MINOR** versão quando você adiciona funcionalidade de uma maneira compatível com versões anteriores;
- **PATCH** versão quando você faz correções de erros. (Não adiciona funcionalidade nova)

Esta não é uma ideia nova ou revolucionária. Na verdade, você provavelmente já faz algo próximo a isso. O problema é que "fechar" não é bom o suficiente. Sem conformidade com algum tipo de especificação formal, os números de versão são essencialmente inúteis para o gerenciamento de dependências. Ao dar um nome e uma definição clara às idéias acima, fica fácil comunicar suas intenções aos usuários do seu software. Uma vez que essas intenções sejam claras, especificações de dependência flexíveis (mas não muito flexíveis) podem finalmente ser feitas.

Um exemplo simples demonstrará como o versionamento semântico pode tornar a dependência um inferno do passado. Considere uma biblioteca chamada "Firetruck". Requer um pacote com versão semântica chamado "Ladder". No momento em que o Firetruck é criado, o Ladder está na versão 3.1.0. Como o Firetruck usa algumas funcionalidades que foram introduzidas pela primeira vez no 3.1.0, você pode especificar com segurança a dependência do Ladder como

maior ou igual a 3.1.0, mas menor que 4.0.0. Agora, quando as versões 3.1.1 e 3.2.0 do Ladder estiverem disponíveis, você poderá liberá-las no sistema de gerenciamento de pacotes e saber que elas serão compatíveis com o software dependente existente.

Como desenvolvedor responsável, é claro que você deseja verificar se qualquer atualização de pacote funciona como anunciado. O mundo real é um lugar confuso; não há nada que possamos fazer sobre isso, mas fique atento. O que você pode fazer é permitir que o Semantic Versioning forneça uma maneira sensata de liberar e atualizar pacotes sem precisar rolar novas versões de pacotes dependentes, economizando tempo e trabalho.

Se tudo isso parece desejável, tudo o que você precisa fazer para começar a usar o Semantic Versioning é declarar que está fazendo isso e seguir as regras. Link para este site a partir do seu LEIA-ME para que outras pessoas conheçam as regras e possam se beneficiar delas.

Referências

1. Semantic Versioning 2.0.0 - https://semver.org/
2. A successful Git branching model - https://nvie.com/posts/a-successful-git-branching-model/