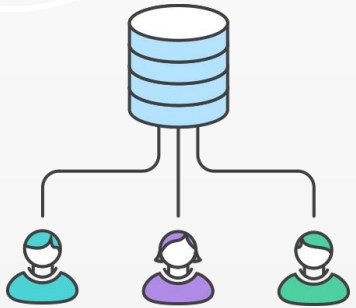
TRUNK BASED DEVELOPMENT

MODELOS PARA BRANCHES DE CÓDIGO FONTE



Edilson S.Morais • Devops

Sumário

1. Trunk Based Development
    a. O Que é Trunk Based Development
    b. Estilo Committing to the trunk
    c. Estilo Short Lived Feature Branches
    d. Estilo release for Trunk
    e. Estilo branch for Release
    f. Revisando Cenários com TBD
    g. Considerações Sobre Trunk Based Development
2. Material de Referência

TRUNK BASED DEVELOPMENT

O que é o **Trunk Based Development**?

1. Um modelo para controle de branches, em que os devs colaboram no código apenas em uma branch (**Trunk**), e resistem a criação de **branches de vida longa**.
2. A branch main é **sempre estável**, e está sempre pronto para ir para produção. Funcionalidades em desenvolvimento ficam geridas por ***feature toggles***
3. Possibilita **CI/CD** de verdade, os indivíduos de um time fazem commits na trunk várias vezes ao dia, ou pelo menos um vez a cada 24h, isso ajuda a tornar o CI/CD uma realidade.
4. De modo geral, toda a equipe de desenvolvimento deve fazer o que puder para dividir as **histórias/tasks** em menores .
5. **Menos Conflitos**: as entregas são pequenas (**commits atômicos**) - pequenas mudanças incrementais resultam em conflitos menos dolorosos.
6. As revisões de código são menores e mais eficazes.

Estilos de desenvolvimento em trunk based

A linha divisória entre os estilos de desenvolvimento em Trunk-Based está relacionada ao **número de desenvolvedores do time e taxa de commits**.

Estilos:

1. **Committing to the trunk**.

   Indicado para time com equipe pequena, com commiters ativos entre 1 a 100.

2. **Short lived feature branch**.

   Indicado para time com equipe maior, com commiters ativos entre 2 a 1000.

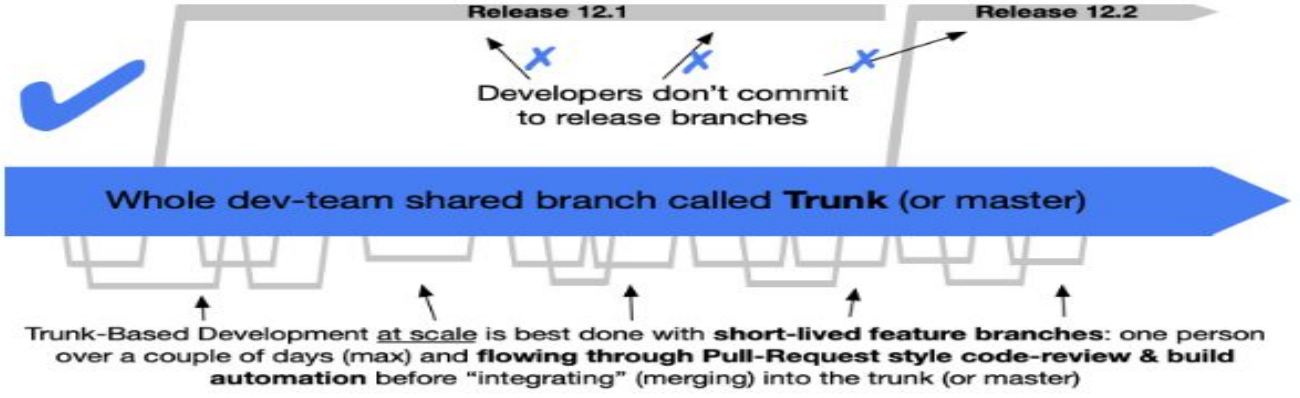
Independente do modelo, as equipes realizam o **CI completo** (compile, unit tests, integration tests) em suas **estações de trabalho** antes do committing/pushing.

1

Trunk-Based Development For Smaller Teams:

2

Scaled Trunk-Based Development:

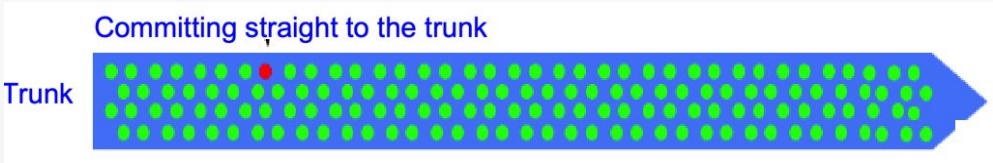


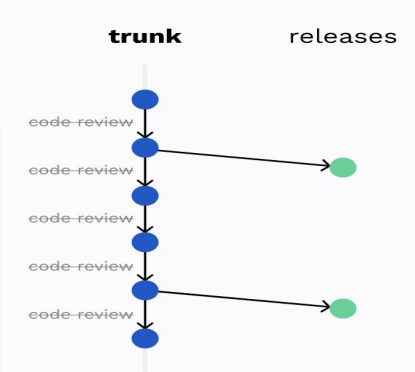
VAMOS FALAR
DO MODELO
COMMITTING
TO THE TRUNK

Model **Committing to the trunk**

1. **Commits diretamente no tronco**.

- Indicado para time com equipe pequena, com commiters ativos entre 1 a 100.
- Altamente recomendado **pair-programming**.
- Cada membro sabe o que o outro está fazendo.
- **Code reviewer** feito pelos companheiros.
- Builds rápidos e raramente quebram .



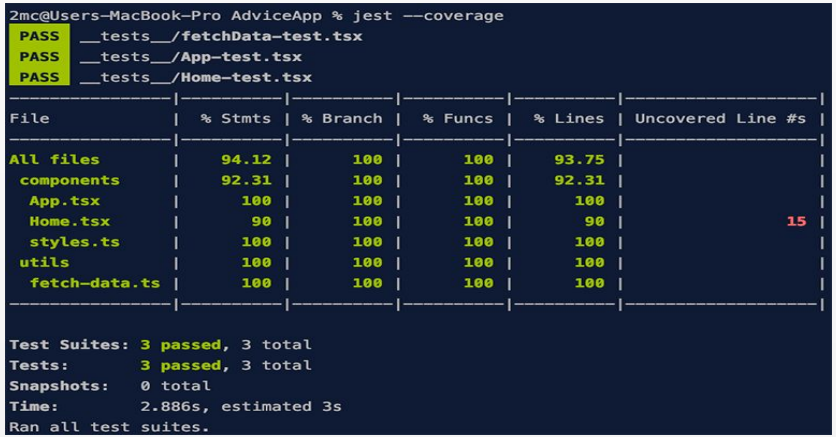


Continuando … Modelo Committing to the trunk.

2. Commit/push direto no tronco tem seus **desafios**:

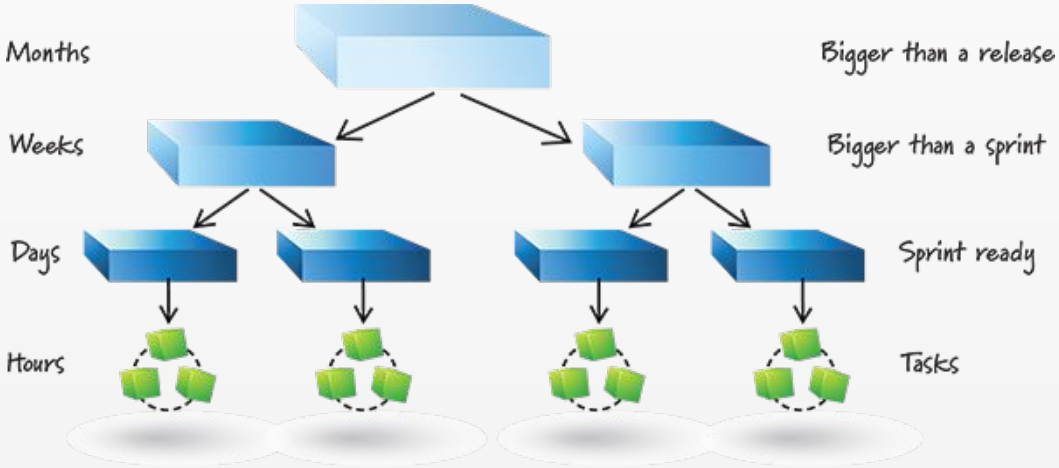
- Alguém pode enviar um código que **quebra o build**.

A **mitigação deste risco** é todo mundo executar em suas workstations o mesmo **build/tests** que a esteira de CI faria, antes do commit/push, e apenas fazendo push para o trunk se isso for aprovado.



Continuando … Modelo Committing to the trunk.

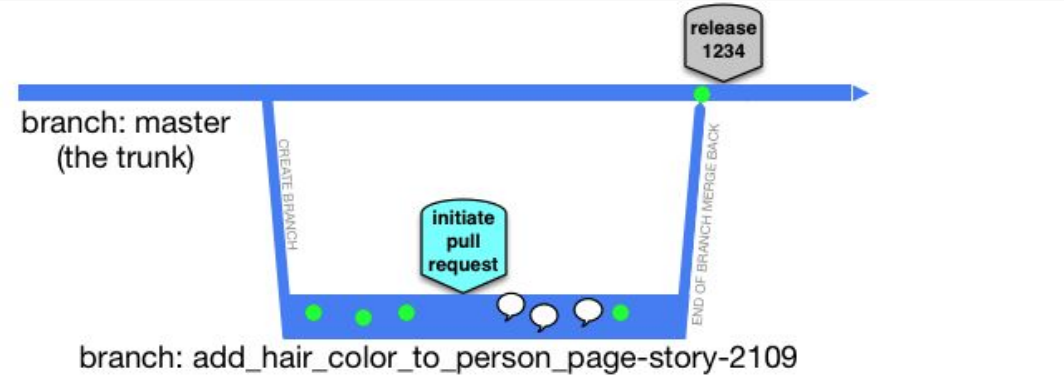
3. Há uma maior **probabilidade** de que as **histórias/tasks** sejam divididas em **menores**, e que esse estilo se torne um fluxo de pequenos commits no tronco, com cada um deles sendo um passo incremental à frente e perfeitamente capaz de entrar em operação enquanto a story/card maior permanece incompleto.



VAMOS FALAR
DO MODELO
SHORT LIVED
FEATURE BRANCH

Model **Short lived feature branches**

1. **Feature branch** tem uma regra fundamental que é a duração de no máximo 2 dias.
   - Há o risco de se tornar uma **feature branch de longa duração** (**anti-padrão TBD**)
   - Seja um clone, ou um fork, esses branches voltaram como PR para main/trunk.

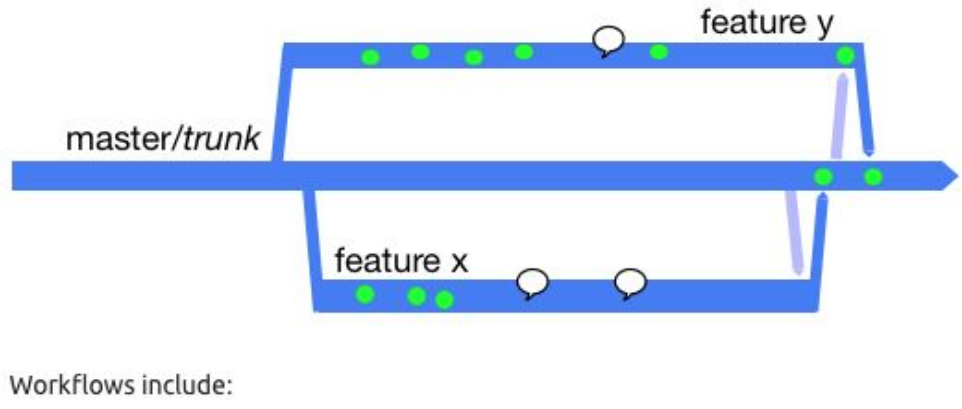


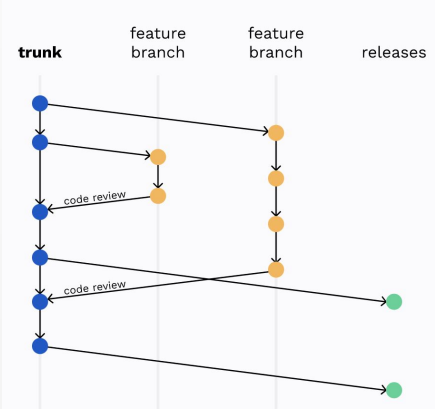
Sometimes the community calls these 'task' or 'topic' branches, instead of long-lived feature branch.

Continuando … Model Short lived feature branches

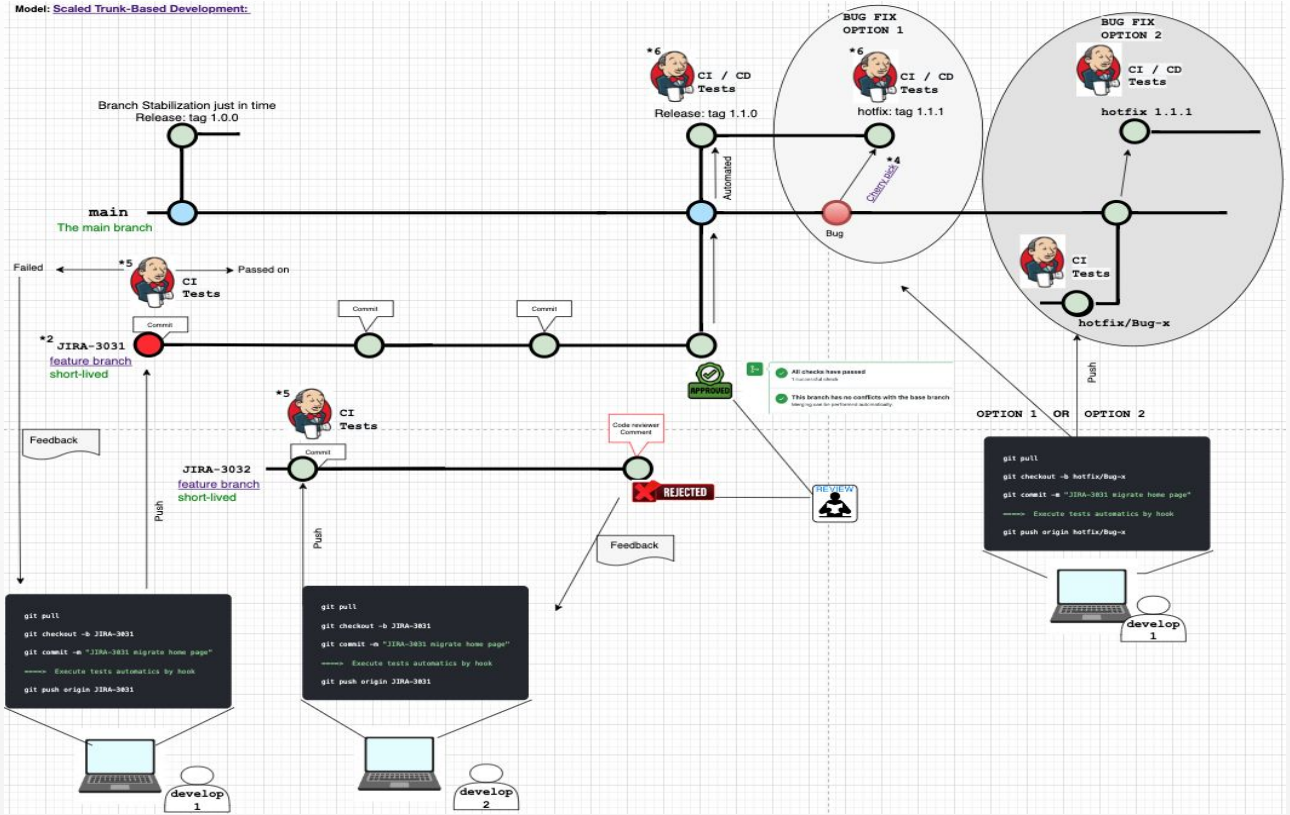
2. **Mais produtivo** para time com **16 ou mais** pessoas e com commiters ativos entre 2 a 1000.

- Desenvolvedores poderão trabalhar em features branches X e Y **independentes e simultâneamente**.
- Na feature branch é **executada a esteira de CI** e também o **code reviewer** antes dos commits chegarem na main.





Continuando … Diagrama (exemplo)

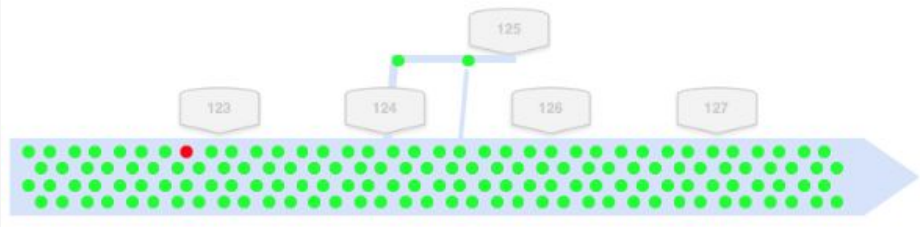


VAMOS FALAR
DO MODELO
Release for Trunk

Model **Release for trunk**

1. Gerar a release no tronco é indicado para equipes com uma **cadência de release** muito alta.

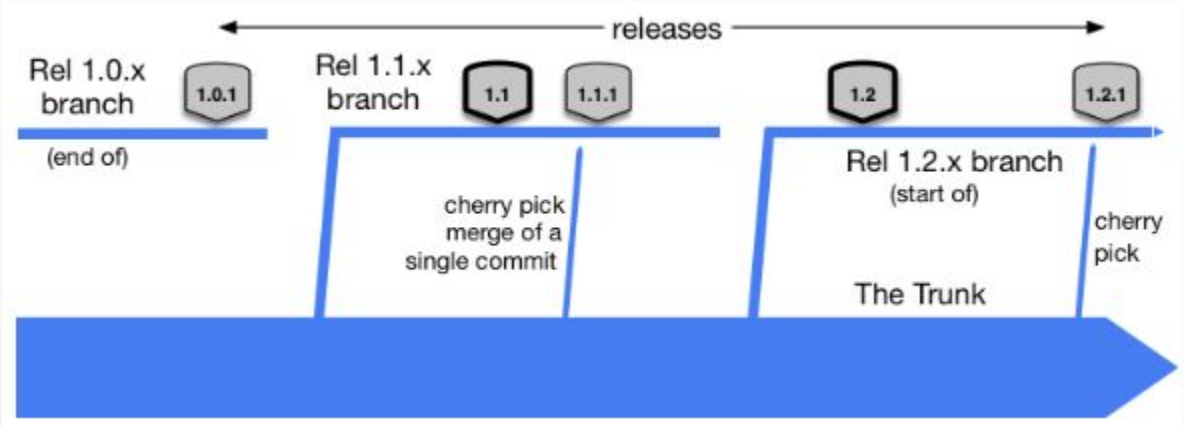
2. A **correção de bug** neste modelo pode ser **direto no tronco**, ou se for gerado mais de uma release por dia, ainda pode criar uma branch release o qual receberá da main o commit com a correção do bug-fix da release. (regra principal é que a main esteja sempre pronta)



VAMOS FALAR
DO MODELO
Branch for Release

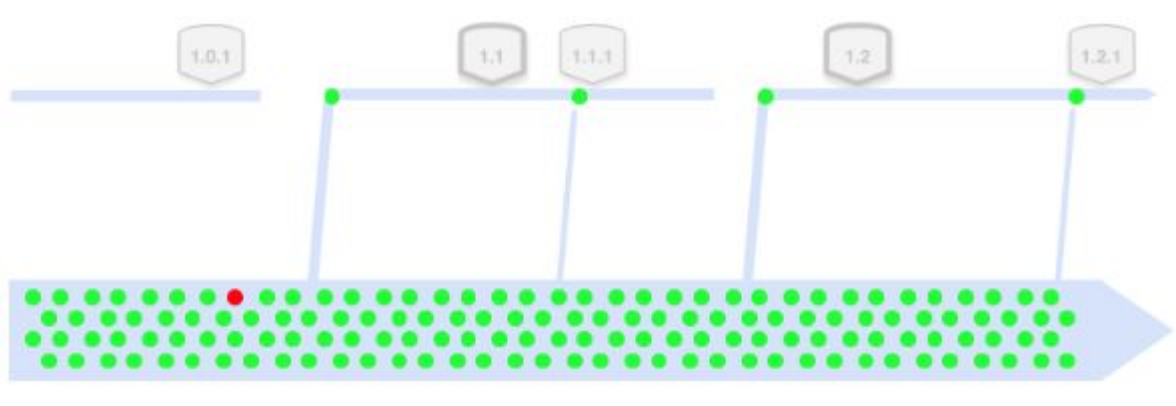
Model **Branch for Release**

1. Indicado para equipes que enviam **release em produção mensalmente**, neste modelo é comum o uso de uma **branch release just in time**, tornando um lugar estável diante de diversos commits, e também facilita a correção dos bugs nesta release.



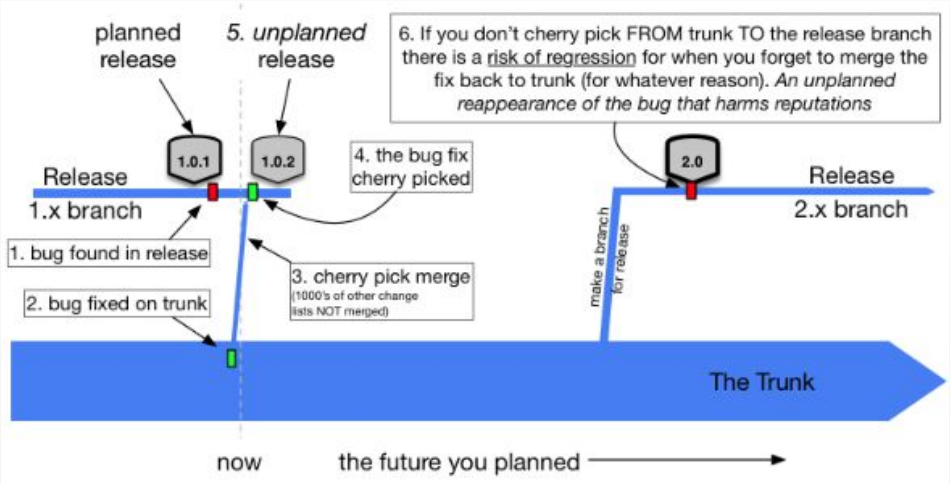
Continuando . . . Branch for release

2. Os desenvolvedores continuam enviando os commits no tronco (**pontos verdes**), em alta velocidade e não diminuem o ritmo nem congelam, e a branch da release não recebe estes commits.



Continuando … Branch for release

2. Uma boa prática no TBD é reproduzir o **bug no tronco**, corrigi-lo com um teste, validando-o na esteira CI, em seguida envia para branch de release (na prática a main deve sempre estar pronta para produção, permitindo o merge para branch release, porém se preferir pode ser enviado apenas o commit).



Revisando cenários com TBD

Revisando Cenários com **Trunk Based Development**

- ✓ **Cenário 1: Merges Menos Dolorosos**: Como as entregas são mais granulares, a quantidade de código que entra para a *branch* principal a cada *commit* é menor, o que torna a **resolução de conflitos** (eles continuam acontecendo, aceite!) **mais fácil e menos dolorosa**. Apenas se recorde de não usar seu repositório local como um estoque de código!
- ✓ **Cenário 2: Code Review de Verdade**: Seguindo a ideia que foi apresentada acima, como a entrada de código é constante e em menores "porções", o processo de revisão deixa de ter caráter exaustivo e sua execução se torna mais eficiente. Mas a mesma regra do **Cenário 1** deve ser observada aqui!

Revisando Cenários com **Trunk Based Development**

- ✓ **Cenário 3: Code Refactoring Constante**: O processo correto de trabalho quando se usa o TBD exige a construção de uma infraestrutura de suporte sólida (como uma configuração adequada do **Sonar**) e o uso de práticas "saudáveis" para escrever código (**TDD**, por exemplo). Isso oferece garantias para que o desenvolvedor pratique a refatoração do código entregue, sem o fantasma da quebra da *build*.
- ✓ **Cenário 4: Entrega Contínua de Verdade**: As entregas são realizadas na *branch* principal, o que realmente caracteriza entrega contínua. O controle das entregas agora se subordina ao uso de **Feature Toggles**, e não mais a *merges* de *branches* de que possuem a funcionalidade.

Revisando Cenários com **Trunk Based Development**

- ✓ **Cenário 5: Cadência de Trabalho Diminui Dependências**: com um único ramo no remoto, é fácil visualizar o que está sendo feito por todo o grupo de trabalho. Dessa forma, dificilmente um trabalho que esteja sendo executado por um desenvolvedor seja replicado por outro. Mas novamente - como nos **Cenários 1 e 2**, a disciplina no fluxo de desenvolvimento é essencial, e o grande mandamento aqui é:

> “***Não usarás sua máquina local para a estocagem de código***”.

Considerações sobre o **Trunk Based Development**?

1. TBD **não é** sobre **evitar ramificações**: é sobre não limitar o seu desenvolvimento a uma relação 1-1 (um ramo para cada funcionalidade). Como desenvolvedor, **você ainda poderá criar quantos *branches* desejar em seu ambiente local**, apenas saiba como gerenciar isso!

2. É necessário ter **maturidade em DevOps** para ter êxito pleno em sua aplicação

Qual caminho seguir?

Para seguirmos com o Trunk Based development

1. Todos os **commits** devem ter o **número da task** (jira number) correto
2. Toda funcionalidade nova deve estar segura por uma **Feature toggle**. Funcionalidades ainda em desenvolvimento não devem quebrar a aplicação, portanto deve estar com a toggle desligada.
3. Fazer o git pull antes de iniciar o desenvolvimento e antes de submeter as mudanças (git push)
4. Fazer o code review das tasks antes de começar o desenvolvimento de outra task.

MATERIAL DE REFERÊNCIA

Material de Referência (links)

1. **Trunk-Based Development**: Excelente artigo produzido pelo Google Cloud.
2. **Trunk Based Development: Five-minute overview**: Artigo publicado na página do projeto que fornece uma visão geral do que é, como funciona e o motivo para adotar o TBD.
3. **Comparing Workflows**: Um excelente artigo publicado pela Atlassian comparando vários fluxos de trabalho com Git.
4. **Por que trunk-based é o melhor modelo de desenvolvimento para DevOps**: Artigo (em português) que faz uma análise interessante em porque adotar o TBD como flow para o Git.
5. **Feature Toggles (aka Feature Flags)**: um artigo que aborda o conceito de Feature Toggles, seus usos, técnicas de implementação e configuração, e como trabalhar com este complexo recurso.

Thank you!