**TRUNK-BASED**

# Introdução

Trunk based development é um modelo de controle de branch, em que os desenvolvedores colaboram no código em uma única branch chamada ‘trunk/main’, este modelo é resistente a qualquer pressão para criar outras branchs de desenvolvimento de vida longa, e emprega técnicas documentadas em [trunkbaseddevelopment](https://trunkbaseddevelopment.com/) . Eles, portanto, evitam o "inferno dos merges", não quebram o build e vivem felizes para sempre.

[**Auxilio para tomada de decisão**](https://trunkbaseddevelopment.com/deciding-factors/)**:**

O desenvolvimento Trunk-Based precisa que você tenha pequenas histórias/tarefas. O ideal é você começar a trabalhar em uma mudança, deve ser apenas uma questão de horas antes de completá-la e enviá-la para a revisão de código.

De um modo geral, toda a equipe de desenvolvimento deve fazer o que puder para dividir histórias/tarefas em histórias/tarefas menores. No Agile, existe um mnemônico [INVEST](https://en.wikipedia.org/wiki/INVEST_%28mnemonic%29) que auxilia na divisão de histórias.

Um dos principais benefícios da abordagem Trunk-Based é que ela simplifica a implantação de eventos e mantém o código atualizado, porque há um número menor de linhas de desenvolvimento e implantações menores e mais frequentes.

Existem muitos fatores decisivos antes de uma equipe de desenvolvimento se decidir pelo desenvolvimento Trunk-Based, mas aqui está uma breve visão geral das práticas, se o fizerem:

# **1 - Estilos de desenvolvimento em trunk based**

Existem basicamente quatro estilos, separados em 2 grupos para desenvolvimento baseado em trunk como uma atividade diária do desenvolvedor. Dependendo do número de desenvolvedores na equipe, da cadência de release e da taxa desejada de commits (supondo que os tamanhos de história suportem isso), você tem trade-off para cada um dos quatro:

1. Envio de commits para trunk

[Committing straight to the trunk](https://trunkbaseddevelopment.com/committing-straight-to-the-trunk/) (figura 2)

[Short-Lived Feature Branches](https://trunkbaseddevelopment.com/short-lived-feature-branches/) (figura 3)

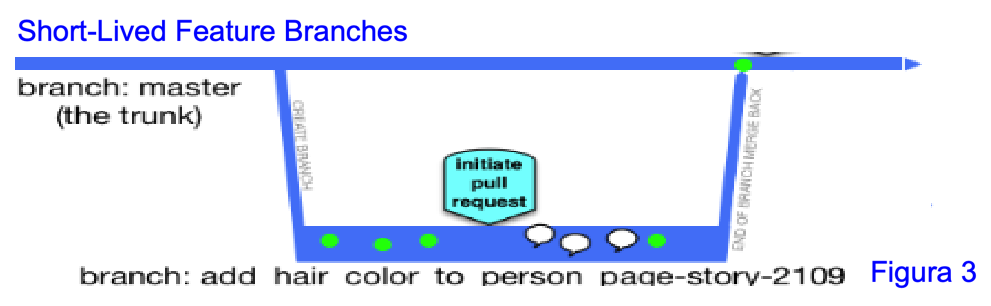
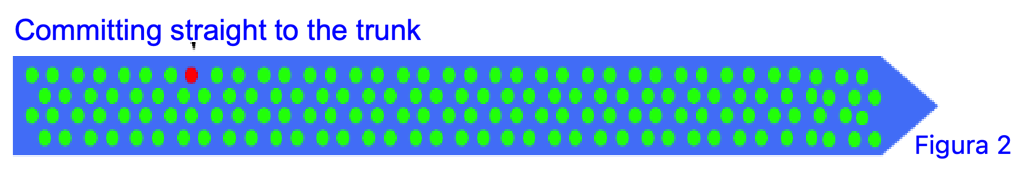
1. Geração de release/tag

[Release for Trunk](https://trunkbaseddevelopment.com/release-from-trunk/) (figura 4)

[Branch For Release](https://trunkbaseddevelopment.com/branch-for-release/)(figura 5)

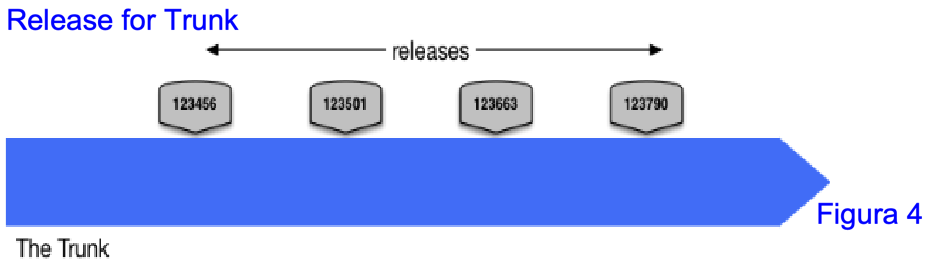
### **1.1 - Envio de commits para trunk**

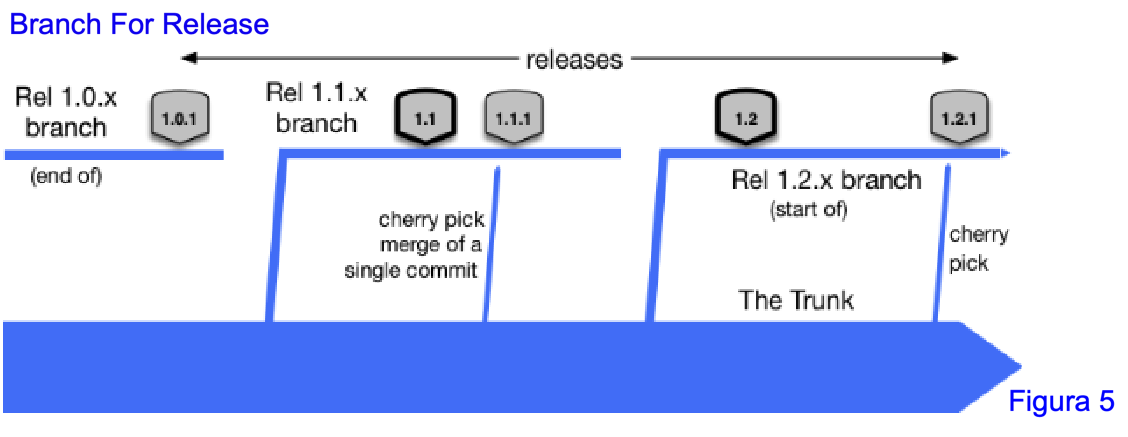
Os estilos **A ( figura 2) e ( figura 3)** descrito acima no tópico 1, fazem referência à prática de envio de commit/push, se diretamente na **Trunk** ou em uma **Feature Branch** de curta duração, sobre esse assunto é comentado melhor no [tópico 2](#_k431ejgbz9ix)



### **1.2 - Geração de release/tag**

Os estilos **B ( figura 4)** e **( figura 5)** descrito no tópico 1, fazem referência à geração da release/tag, se será gerado na **Trunk** ou em uma **Release Branch** para estabilização, sobre esse assunto é comentado melhor no [tópico 3](#_68nu8iyieo4l)

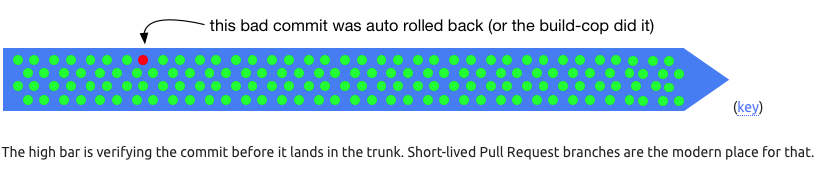




# 2 - Práticas de envio de commits para Trunk.

## [**2.1 - Committing straight to the trunk**](https://trunkbaseddevelopment.com/committing-straight-to-the-trunk/)

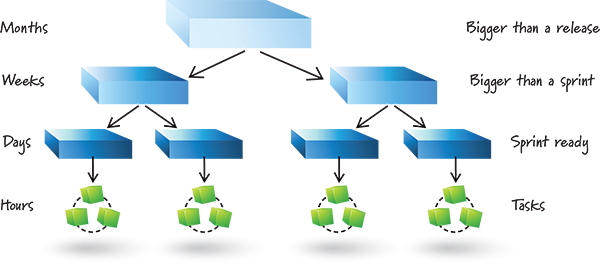
Algumas equipes optarão por enviar commit/push direto para o tronco. Provavelmente porque eles são uma [equipe pequena](https://trunkbaseddevelopment.com/committing-straight-to-the-trunk/), [com committers ativo entre 1 e 100](https://trunkbaseddevelopment.com/styles/) com cada membro da equipe sabendo o que os outros estão fazendo. **Sua construção é provavelmente rápida e relativamente exaustiva**, e eles raramente experimentam uma quebra de build. Se a compilação for interrompida devido uma quebra (pós-integração no trunk/main), é provável que eles 'revertam' o commit imediatamente, possivelmente bloqueando o tronco por um curto período de tempo enquanto isso é executado, para evitar que alguém ​​faça um git-pull e traz aquele commit ruim para sua workstation.



**2.1.1 - Benefícios**

É mais fácil verificar objetivamente a correção de seus commits você mesmo (idealmente com um parceiro de programação em pares) e então fazer commit/push nos momentos de conveniência.

De fato, há uma probabilidade maior de que esse estilo ou trabalho 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.



**2.1.2 - Desafios**

* Commit (e push) direto ao tronco é um dos **desafios**. Principalmente, porque alguém pode executar commit/push do código que quebra a compilação, e o(s) servidor(es) configurado(s) para proteger a Integração Contínua não percebem isso por algum tempo após o commit estar disponível para os colegas de equipe realizar pull/sync com sua workstation de desenvolvimento para outro trabalho.

A mitigação de risco é **todo mundo** executando o build completo (o mesmo build que o demon do CI faria) antes do commit/push e apenas fazendo push para o trunk se isso for aprovado. **Esta é uma atividade essencial de integração**

* Outro **desafio** é relacionado é quanto tempo a “compilação” leva para ser executada pelo serviço de CI, em comparação com a frequência com que o tronco é atualizado com commits, Isso é importante porque, se a compilação falhar (CI), os próximos falharam também.

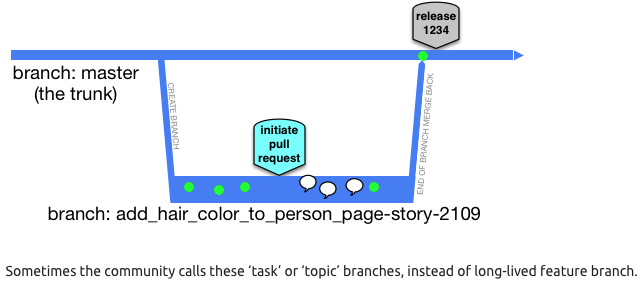
Exemplo: Se sua compilação for de 10 segundos (do início ao fim) e houver um commit a cada cinco minutos, terá tempo para agir, no entanto se sua compilação é de cinco minutos e os commits de sua equipe chegam a cada dez segundos, então você está no "inferno" e deve tentar outra coisa.

## [**2.2 - Short-Lived Feature Branches**](https://trunkbaseddevelopment.com/short-lived-feature-branches/)

Seja como branch diretamente da main ou em um fork. Essas branches estão destinadas a voltar como “pull requests” para o main/trunk.

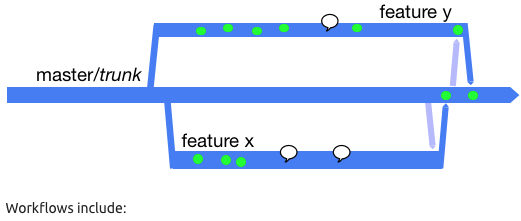
Com [16 ou mais pessoas](https://trunkbaseddevelopment.com/short-lived-feature-branches/) e com [commiters ativo entre 2 a 1000](https://trunkbaseddevelopment.com/styles/#short-lived-feature-branches), a equipe é mais produtiva com short-lived feature branches e os daemons de CI correspondentes sendo executado antes dos commits chegarem ao tronco.

Uma regra fundamental é a duração da branch, ela deve durar apenas alguns dias. Mais de dois dias e há o risco da branch se tornar uma feature branch de longa duração ( anti-padrão do trunk-based development).



**2.2.1 - Benefícios**

* **Dois desenvolvedores podem trabalhar simultaneamente usando uma feature branches.** Digamos que duas features estejam sendo trabalhados simultaneamente: os features X e Y. Ambos levarão um dia para serem concluídos e dois desenvolvedores estão trabalhando independentemente neles.



* As equipes podem se formar em torno do fluxo de trabalho de solicitação de pull request e ainda fazer o desenvolvimento baseado em tronco.
* Na feature branch é executada a esteira de CI **( tests / builds)**, e também o **code reviewer** antes dos commits chegarem na main.

**2.2.2 - Desafios**

* Há o risco de se tornar uma **feature branch de longa duração** ( anti-padrão TBD).
* Não só [existe o risco de mais confirmações](https://trunkbaseddevelopment.com/styles/#short-lived-feature-branches) do que você faria se pudesse enviar diretamente, como também há o risco de levar mais tempo. Se o tamanho médio de sua história for de um dia.

# **3 - Práticas de geração de release.**

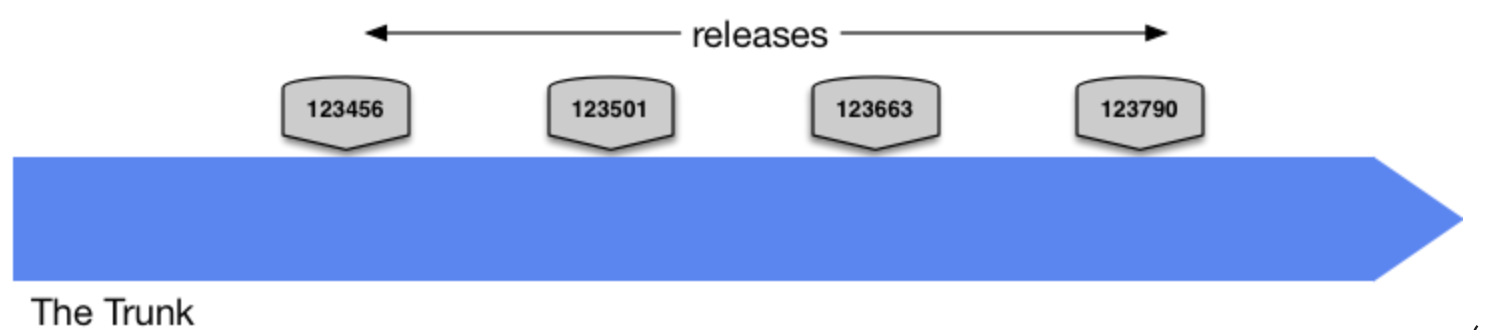
Em Trunk-Based development a branch trunk sempre estará pronto para lançamento de uma release.

Uma regra chave de facilitação é que as equipes de desenvolvimento Trunk-Based liberam exclusivamente diretamente do tronco ou fazem uma branch do tronco especificamente para o release atual, e dependendo da cadência de lançamento de release irá trabalhar ou a geração da release no estilo [Release for Trunk](https://trunkbaseddevelopment.com/release-from-trunk/) ou [Branch release](https://trunkbaseddevelopment.com/branch-for-release/).

**Committing straight to the trunk**

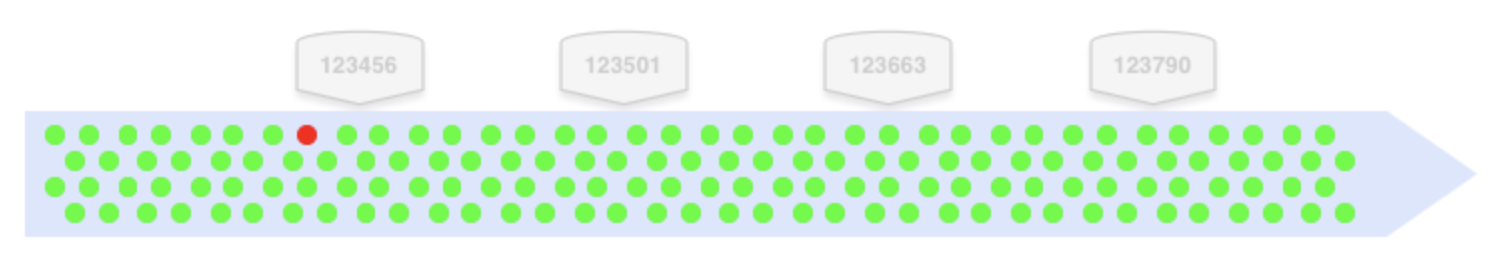
## [**3.1 - Release for Trunk**](https://trunkbaseddevelopment.com/release-from-trunk/)

Este modelo é indicado para equipe com cadência de lançamento de release muito alta, a tag da release é gerada diretamente no tronco.

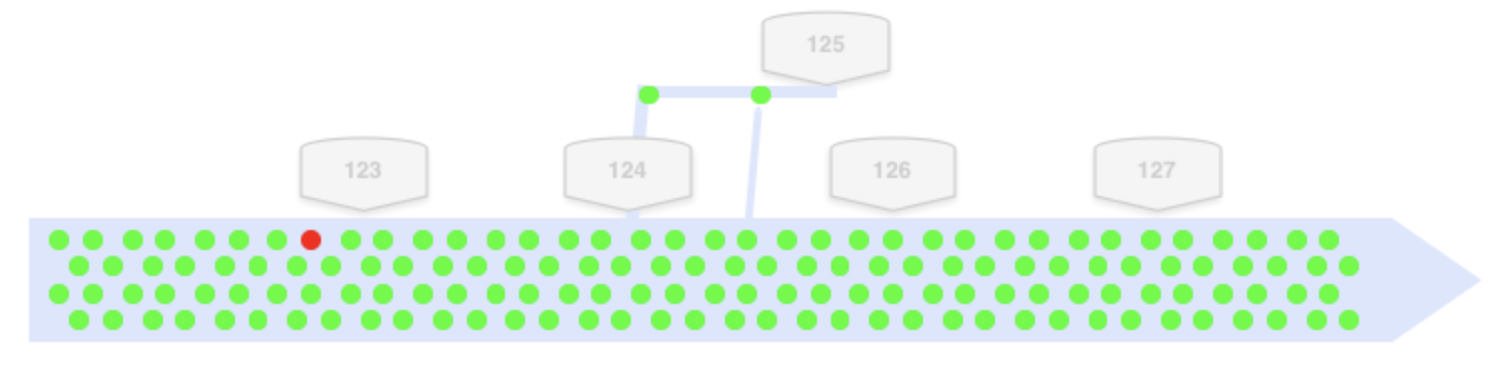
****

É mais provável que essas equipes não usem um esquema de numeração de lançamento decimal de Dewey e, em vez disso, tenham algo referente ao número ou data e hora do commit. Eles provavelmente também optaram por avançar e corrigir o bug no tronco como se fosse uma feature, embora o mais rápido possível.

Veja como são os commits estilizados:



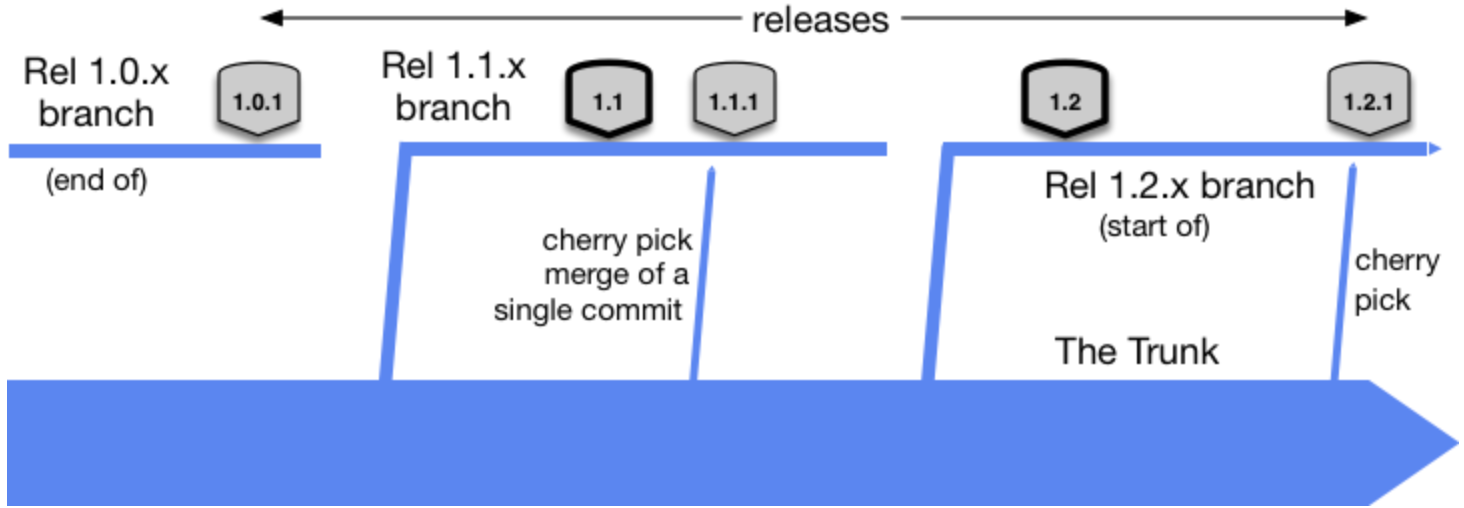
Equipes com **um lançamento por dia** (ou menos) ainda podem fazer uma branch, para escolher a correção de bug e liberar de:

****

## [**3.2 - Branch For Release**](https://trunkbaseddevelopment.com/branch-for-release/)

Se uma equipe está enviando release de produção mensalmente, eles também terão que enviar bug-fix release entre as releases planejados. Para facilitar isso, é comum que as equipes no desenvolvimento em Trunk-based façam uma branch de release just in time - digamos, alguns dias antes da release. Isso se torna um lugar estável, já que os desenvolvedores ainda estão transmitindo seus commits no tronco a toda velocidade.

*A política incompatível (ref Wingerd & Seiwald acima), que o ramo de release “não deve receber trabalho de desenvolvimento contínuo”.*

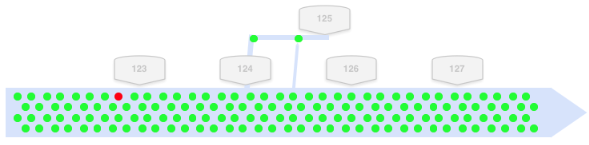


*Trunk, duas ramificações e meia de releases, cinco releases (dois planejados, três não planejados) e duas correções de bugs escolhidos a dedo*

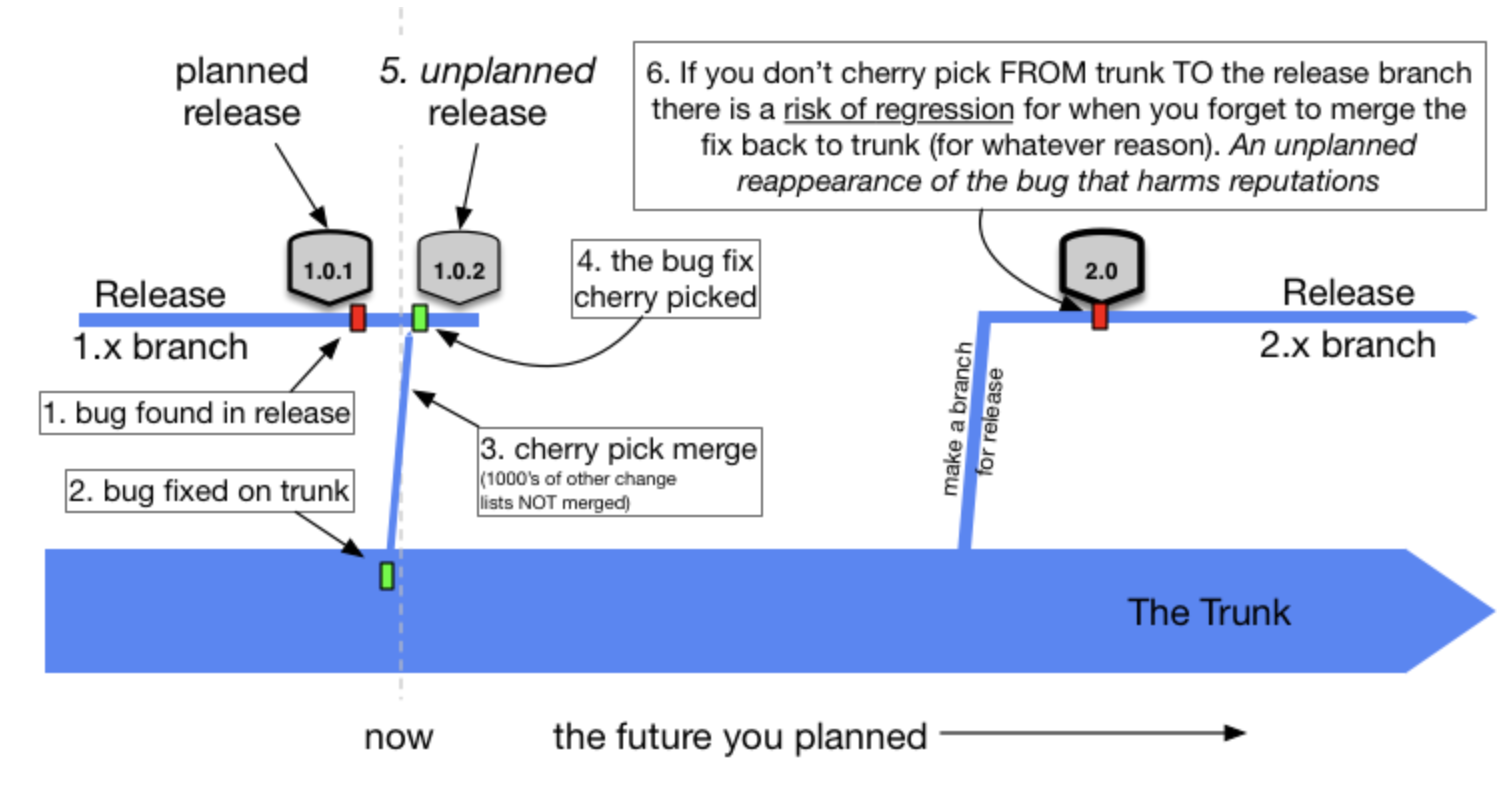
# **4 - Práticas de correção de Bug**

A [melhor prática](https://trunkbaseddevelopment.com/branch-for-release/#fix-production-bugs-on-trunk) para equipes de desenvolvimento em Trunk-based é reproduzir o bug **no tronco, corrigi-lo lá com um teste**, acompanhar a execução da esteira de CI, depois enviar para branch de release e aguardar a execução da esteira de CI também, no entanto pode ser adaptada a preferência da organização.

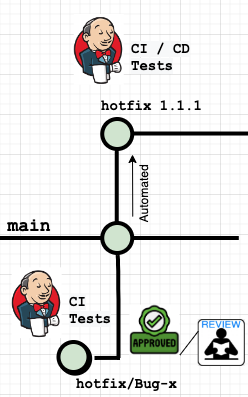
No caso, se for gerado [mais de uma release por dia](https://trunkbaseddevelopment.com/release-from-trunk/), é mais fácil seguir como exemplo mencionado acima, criando uma release branch, o qual receberá da main o commit com a correção do bug-fix via **Cherry-pick** ou um **merge da main para a release branch**, visto que a regra principal é que a main esteja sempre pronta.



Você **não deve corrigir bugs na branch de release** na expectativa de colocá-los de volta no tronco. Por quê? Bem, caso você esqueça de fazer isso no calor do momento. Esquecer significa uma regressão na produção algumas semanas depois. Isso pode acontecer se as coisas estiverem sendo consertadas durante a noite por um desenvolvedor cansado que quer voltar para a cama.



Se estiver utilizando o estilo de envio de commit para tronco via feature branch de curta duração, poderá **criar uma branch hotfix/\* que se comporta exatamente igual a feature branch,** sendo de também de curta duração, onde nela você poderá aplicar o uso semântico de geração das tags.

****

# **5 - Code Review**

Independentemente do modelo de branch, o desejo é que seja feito o [Code Reviews](https://trunkbaseddevelopment.com/continuous-review/) do PR com bastante rapidez.

Algumas equipes contaram com o fato de que o código foi 'pair programmed' como uma revisão automática, neste caso são os próprios colegas que fazem o code review. A ideia é que os desenvolvedores pares se comprometam a fazer revisões de código de forma objetiva e rápida.

Outras equipes seguirão um design convencional em que o commit é empacotado para revisão antes de chegar no tronco.



*os balões de fala são comentários de revisão de código estilizados*

As branchs de revisão de código ( feature branch ) podem (e devem) ser excluídas após a conclusão da revisão de código e ter [vida curta](https://trunkbaseddevelopment.com/short-lived-feature-branches/).

# 6 - Segurança

[Os Continuous Integration](https://trunkbaseddevelopment.com/continuous-integration/)(CI) são configurados para observar o tronco (e as [short-lived feature branches](https://trunkbaseddevelopment.com/short-lived-feature-branches/) usadas na revisão) e informar o mais rápido e completamente possível loudly/visibly ( em voz alta/visivelmente) à equipe que o tronco está quebrado. Algumas equipes bloquearão o tronco e farão rollback das alterações. Outros permitirão que o servidor CI faça isso automaticamente.

# 7 - Compromissos do time de desenvolvedores.

Como afirmado, os desenvolvedores estão prometendo ser rigorosos e não quebrar a build. Eles também precisarão considerar o impacto de seus commits potencialmente maiores, especialmente onde as renomeações ou movimentos foram por atacado, e adotar técnicas para permitir que essas alterações sejam mais facilmente consumidas pelos colegas de equipe.

# 8 - [Como implantar Trunk-based](https://cloud.google.com/architecture/devops/devops-tech-trunk-based-development?hl=pt-br)

O desenvolvimento baseado em troncos é uma prática necessária em casos de [integração contínua](https://cloud.google.com/architecture/devops/devops-tech-continuous-integration?hl=pt-br).

Na integração contínua, pequenos lotes de códigos são integrados frequentemente para eliminar integrações longas e fases de estabilização. Assim, os desenvolvedores garantem que todos saibam o que está sendo feito, enquanto a integração elimina grandes implantações que poderiam gerar muito trabalho para outros desenvolvedores e testadores.

No paradigma de CI, os desenvolvedores são responsáveis por manter o processo de build verde, ou seja, funcional. Isso significa que, se o processo de CI falhar, os desenvolvedores precisam parar o que estão fazendo para corrigir o problema imediatamente ou reverter a mudança caso a correção não possa ser feita em poucos minutos.

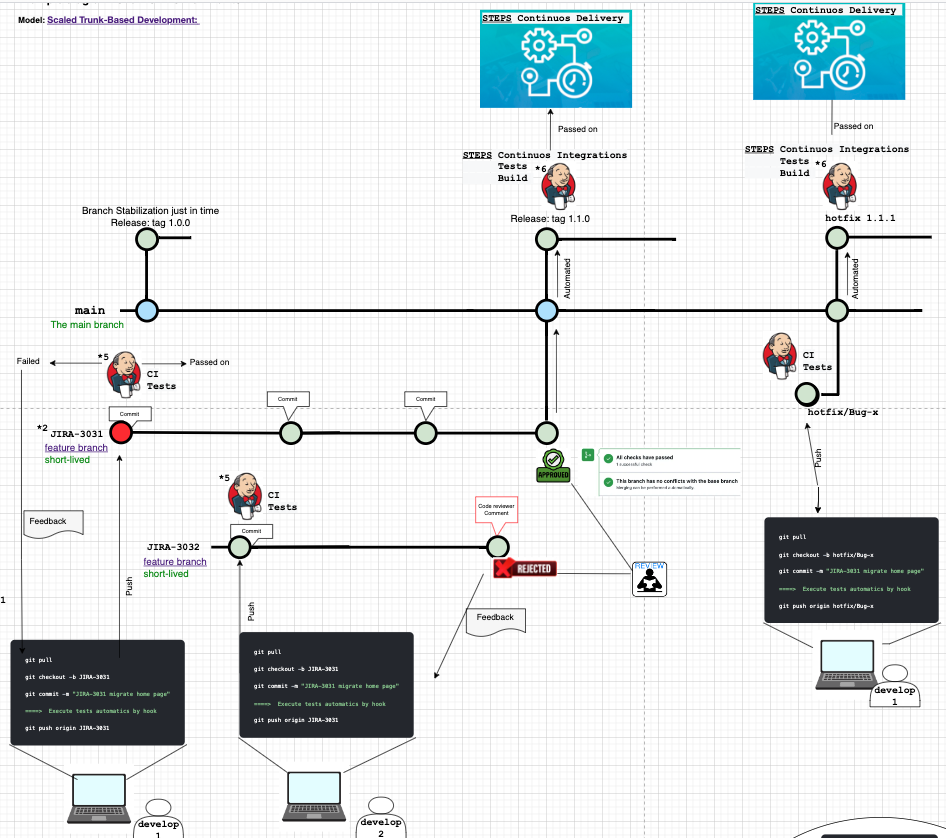
Por outro lado, o desenvolvimento baseado em troncos exige que os desenvolvedores saibam como [dividir o trabalho em lotes pequenos](https://cloud.google.com/architecture/devops/devops-process-working-in-small-batches?hl=pt-br). Isso é uma mudança significativa para os desenvolvedores não acostumados a trabalhar assim.

A análise dos dados do DevOps Research and Assessment (DORA) de [2016](https://services.google.com/fh/files/misc/state-of-devops-2016.pdf?hl=pt-br#page=31) (PDF) e [2017](https://services.google.com/fh/files/misc/state-of-devops-2017.pdf?hl=pt-br#page=40) (PDF) mostra que as equipes atingem níveis mais altos de entrega de software e desempenho operacional (estabilidade, disponibilidade e velocidade de entrega) quando seguem estas práticas:

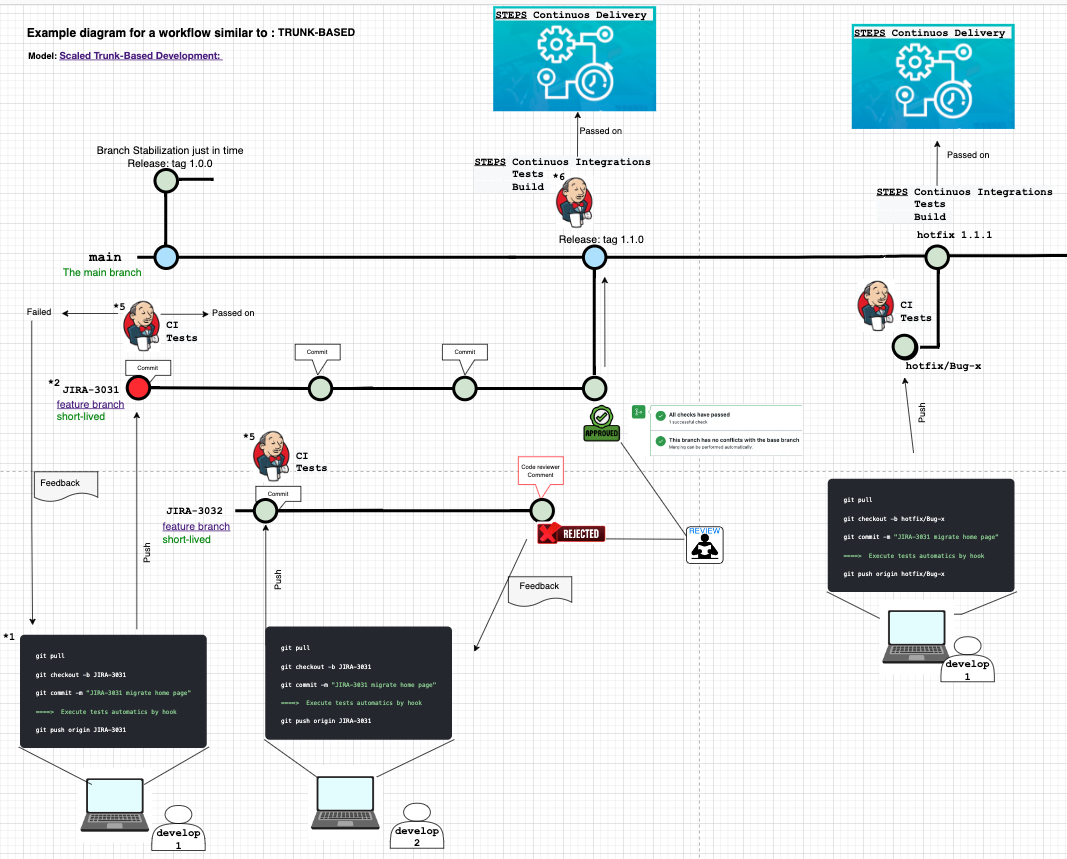
* Ter três ou menos ramificações ativas no repositório de código do aplicativo.
* Implantar branches no tronco pelo menos uma vez por dia.
* Evitar congelamentos de códigos e fases de integração.

# Diagrama do modelo Short Lived Feature Branch

No exemplo está utilizando a geração de release/tag na **branch release**.

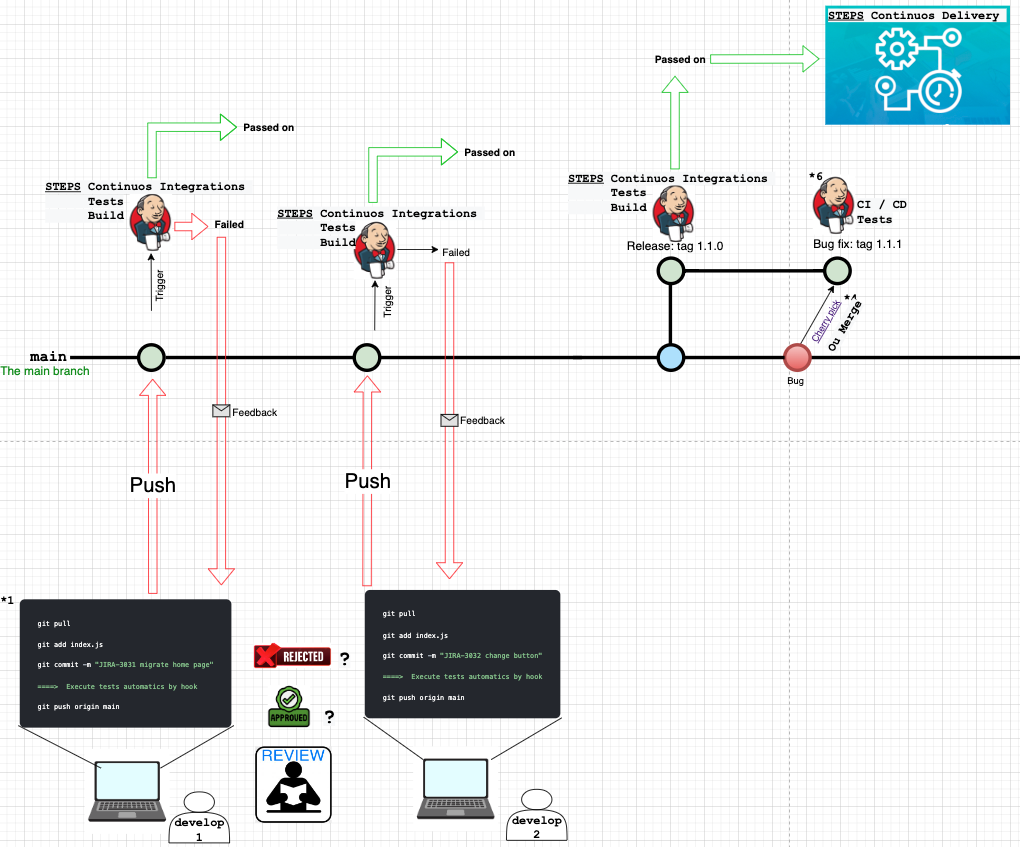


No exemplo abaixo está utilizando a geração de tag **direto no tronco**.

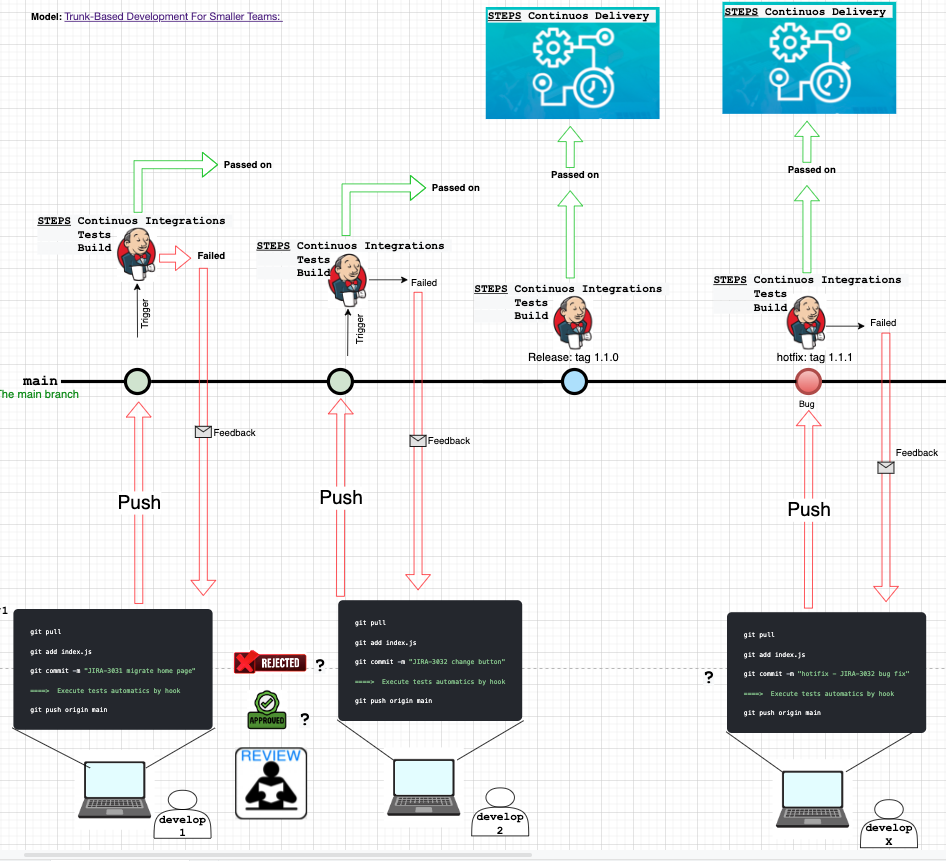


# Diagrama do modelo Committing to the trunk

No exemplo está utilizando a geração de **release/tag na branch release**.



No exemplo abaixo está utilizando a geração de **tag direto no tronco**.



# Referências:

<https://trunkbaseddevelopment.com/>

<https://cloud.google.com/architecture/devops/devops-tech-trunk-based-development?hl=pt-br>