



UNIVERSIDADE
FEDERAL DE
MATO GROSSO DO SUL



Gerência de Configuração de Software

Aula 06 – Modelo de Ramificações (*Branching Model*)

Prof. Dr. Awdren de Lima Fontão
awdren.fontao@ufms.br

A princípio, o branch pode ser um perigo, um ampliador de risco

Existem algumas regras que podem minimizar esse risco:

- Commits simples e “entregáveis”, orientados à uma tarefa;
- Branches de vida curta, margens mais simples;
- Estratégia combinada pela equipe;
- Devemos lembrar que muitos branches geram mais burocracia, e apresentam suas desvantagens.

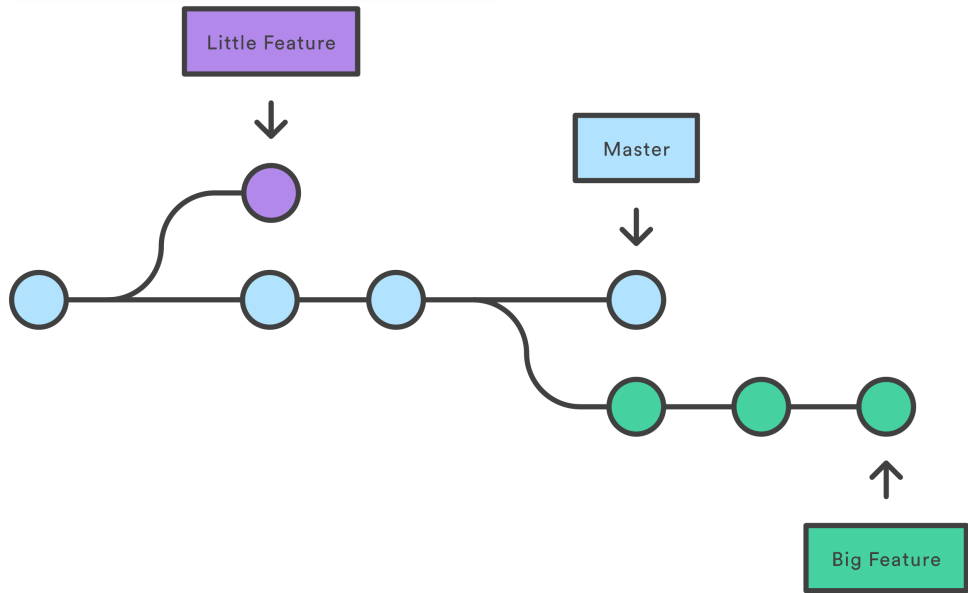
Modelos



100

Uma ramificação no *git* é um ponteiro para as alterações feitas nos arquivos do projeto.

- É útil em situações nas quais você deseja adicionar um novo recurso ou corrigir um erro;
 - Isto gera uma nova ramificação garantindo que o código instável não seja mesclado nos arquivos do projeto principal.
- Depois de concluir a atualização dos códigos da ramificação, você pode mesclar a ramificação com a principal, geralmente chamada de *master*.



Commit e Pull Request



- **Commit:** alterações enviadas ao repositório local desde que gerem uma revisão;
- **Pull Request:** solicitação de envio. Dialoga com a prática de *code review*, em que os desenvolvedores conversam e discutem sobre a estrutura do código e fazem tomadas de decisão.
 - Mas o code review independe do pull request

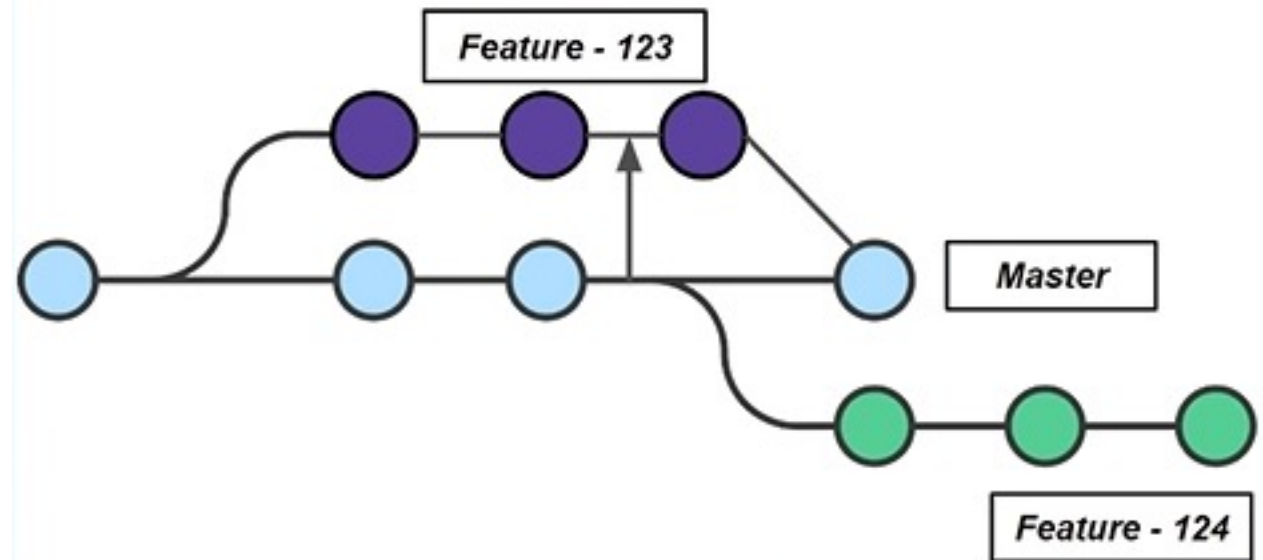
Modelos de Ramificações (*Branching Model*)

Alguns branching models famosos são:

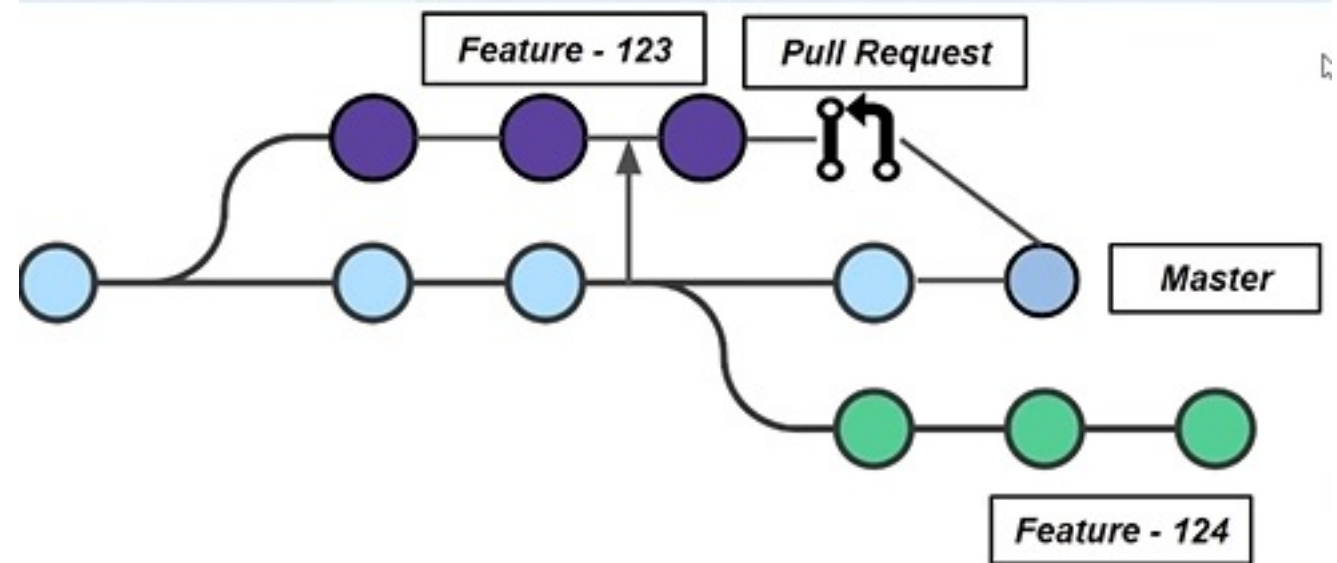
- **Temporários (branches locais):** são branches localizados apenas na máquina local e deverão se extinguir, são utilizados para organizar fluxos de trabalho e depois realizar o commit;
- **Feature Branches:** são utilizados para implementar funcionalidades ou orientar tarefas;
- **Historical Branches (master e develop):** as alterações ficam organizadas em commits baseados na cronologia no caso de um projeto de software;
- **Environment Branches (Staging e Production):** existem branches que são baseados no ambiente, isto é, em que espaço é realizado o deploy;
- **Maintenance Branches (Release e Hotfix):** temos, ainda, os branches de manutenção do sistema.


Feature Branch Workflow

- O primeiro modelo é o mais simples, menos sofisticado de todas as ramificações, mas reduz a distância de um branch para o master até o mínimo possível;
- O modelo future branch workflow faz parte da realidade de mais desenvolvedores;
- A ideia é simples: precisamos implementar uma nova funcionalidade e então criamos uma ramificação para este fim.

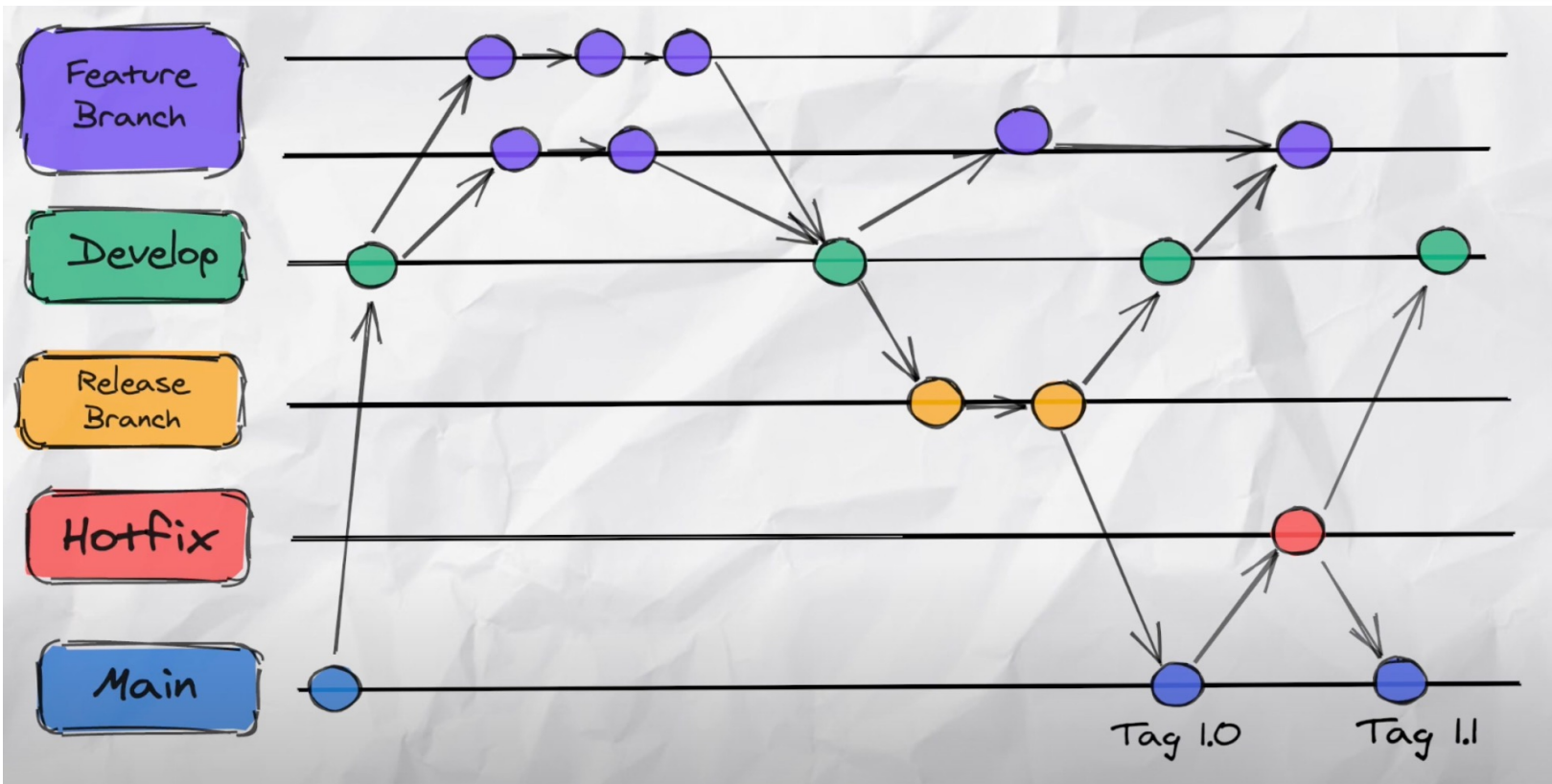


- Uma vez implementado, ele será comitado ao master e então teremos duas fontes de rebase, e que ainda deverão ser discutidas;
- Algo comum é combinar o future branch work flow com o *pull request*.



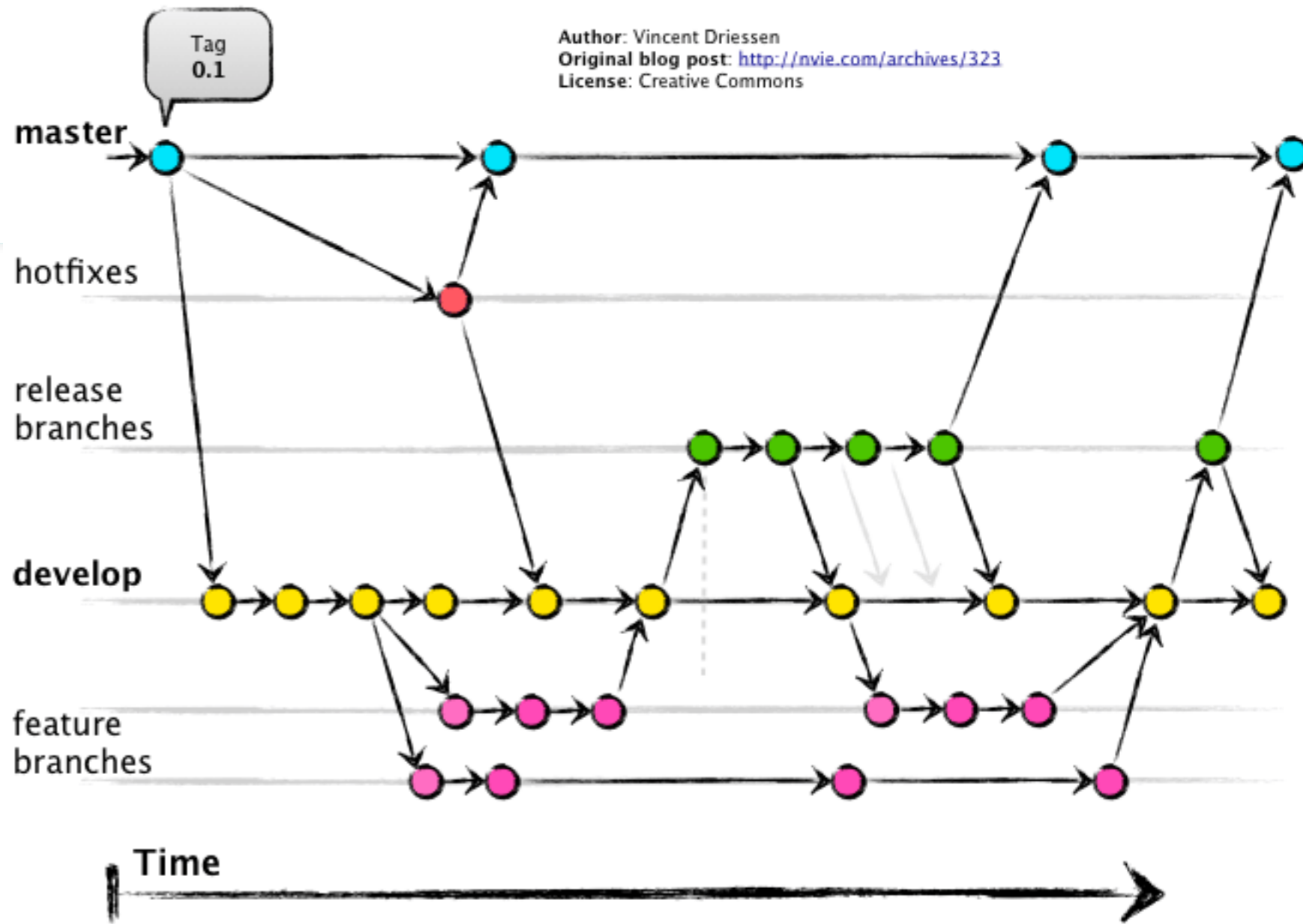
- 
- A partir de um recurso do Git, fazemos a implementação, mas não podemos - ou não queremos - fazer o merge diretamente no master;
 - Neste caso, utilizamos o pull request. Uma vez que a feature estiver finalizada, o pull request notifica todos os envolvidos da equipe, e então será avaliado o merge com o master.

Git Flow



O mais popular - e mais complexo - modelo de trabalho é o Git Flow; Obviamente, apesar da popularidade do modelo, os problemas de gerar problemas são os mesmos.

- O branch que representa o histórico de trabalho é o "develop", e os features são integrados e ao final ocorre o commit.
 - Se em algum momento desejamos criar uma nova funcionalidade, criaremos um novo branch a partir de um commit.
- Uma vez que a nova versão foi liberada e revisada, se libera uma commit no master, que represente os releases em produção.
- **Se surge algum problema em tempo de produção**, é criado uma nova ramificação e implementar o Hotfix e então lançar uma nova versão no master.
- Portanto, existem vários caminhos que o código percorre. **Não sabemos dizer qual é o trunk mais atualizado**: master, release ou develop;
 - Pois possuem uma autonomia que pode gerar confusão em algum momento, além de criar uma maior complexidade no sistema de automação.





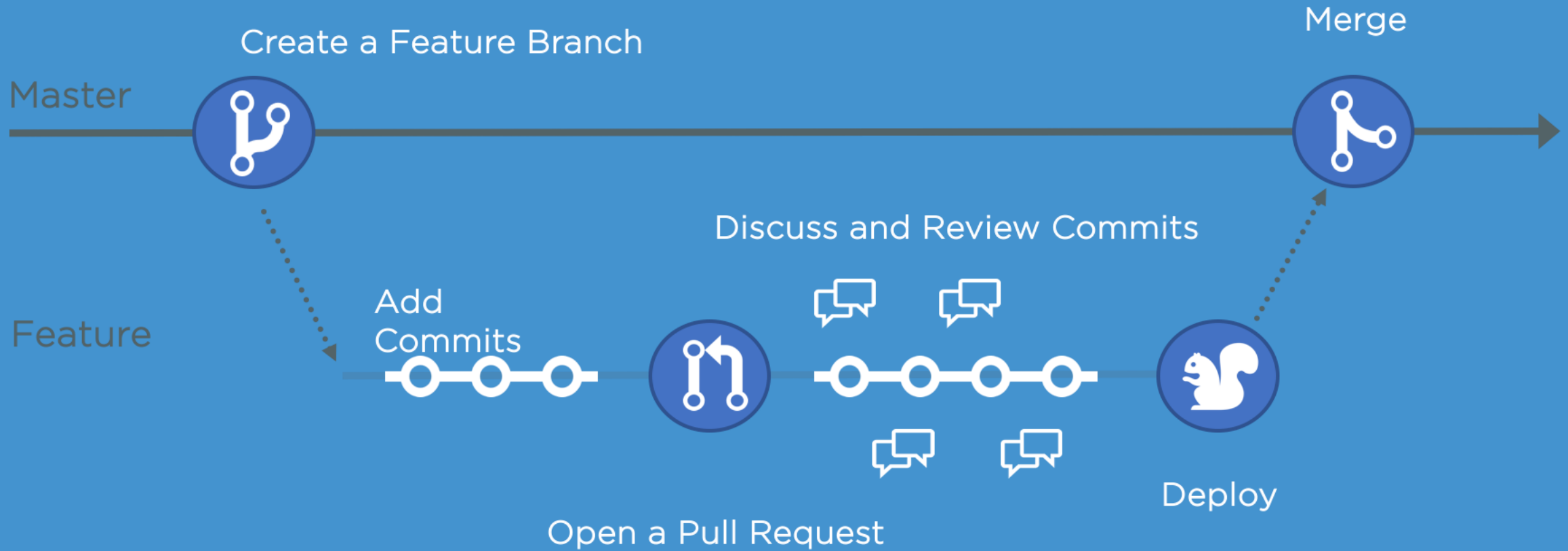
Outro ponto positivo desse fluxo é que se encaixa perfeitamente quando você trabalha em equipe e um ou mais desenvolvedores têm que colaborar para o mesmo recurso.



GitHub Flow

- O fluxo de GitHub é um fluxo de trabalho leve e baseado no branch. O fluxo do GitHub é útil para todos, não apenas para os desenvolvedores.

GitHub Flow





Trabalho em um ambiente de implantação contínua onde não há o conceito de "lançamento".

Pois toda vez que se termina de preparar um novo recurso, este é enviado imediatamente (após toda a cadeia de automação criada no ambiente).

Principais conceitos do GitHub Flow

- Qualquer coisa na ramificação master pode ser implantada;
- Para trabalhar em algo novo, crie um branch com nome descritivo fora do master (ou seja: new-oauth2-scopes);
- Comprometa-se com essa ramificação localmente e envie regularmente seu trabalho para a mesma ramificação nomeada no servidor;
- Quando você precisar de feedback ou ajuda, ou achar que a ramificação está pronta para a fusão, abra uma pull request;
- Depois que outra pessoa revisar e assinar a PR, você poderá mesclá-lo no master
- Depois de mesclado e enviado para "master", você pode e deve implantar imediatamente

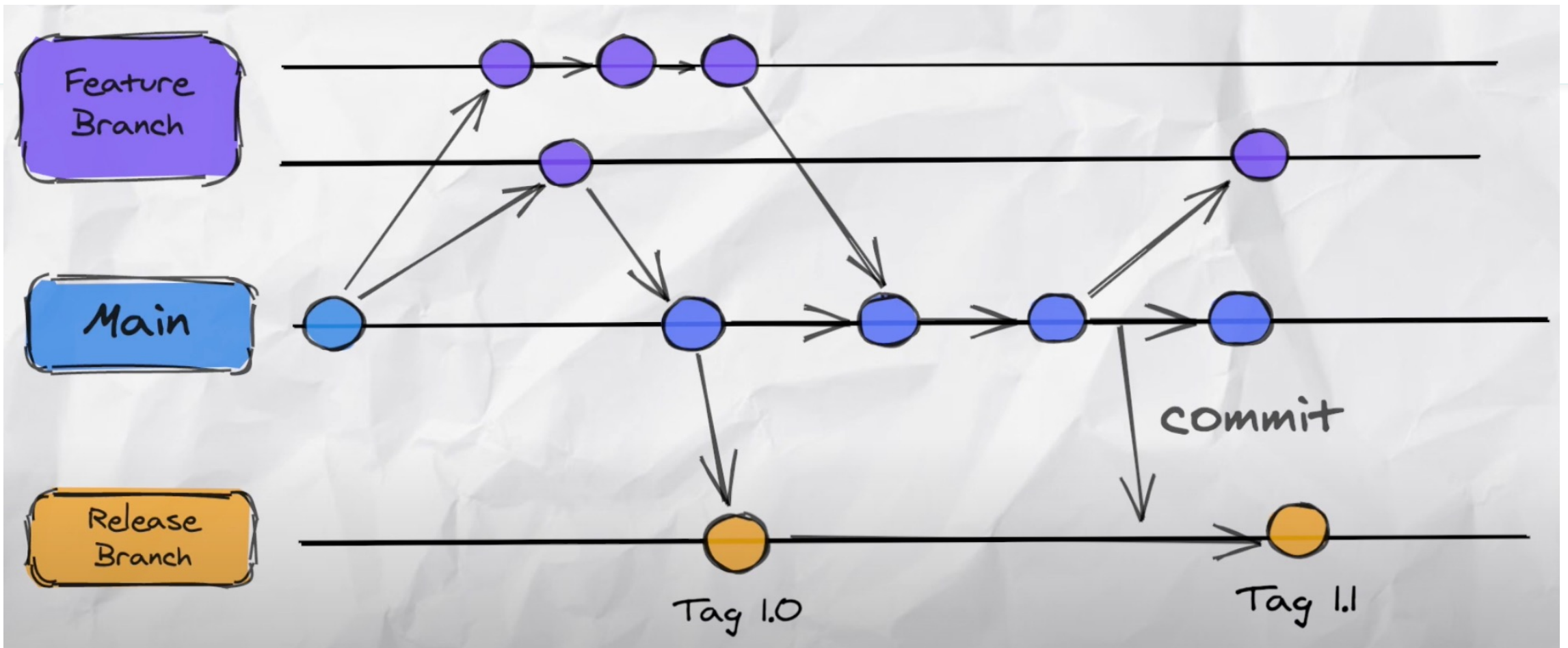
100



- 1

Trunk-based development


- Um modelo de ramificação, onde os desenvolvedores colaboram no código em uma única ramificação chamada 'tronco'*;
- Resiste a qualquer pressão para criar outras ramificações de desenvolvimento de longa duração empregando técnicas documentadas. Eles, portanto, evitam mesclar, não quebram a release...
- * main ou master, na nomenclatura Git



- O Desenvolvimento Baseado em Tronco não é um novo modelo de ramificação;
- A palavra “tronco” refere-se ao conceito de árvore em crescimento, onde o vão mais grosso e mais longo é o tronco, e não os galhos que dele irradiam e são de comprimento mais limitado.
- Tem sido um modelo de ramificação menos conhecido desde meados dos anos noventa e considerado taticamente desde os anos oitenta.
 - As maiores organizações de desenvolvimento, como o Google e o Facebook, praticam em escala

- A proposta do trunk based development é diminuir a complexidade de promover código, compartilhar código, mitigar os *merges complexos* e aumentar produtividade.
- Nesse modelo os desenvolvedores trabalham em uma única *branch*, a *branch* tronco, geralmente a *master*, a ideia é que tudo que todos estão fazendo, esteja na *master* e seja compartilhado entre todos os outros desenvolvedores.

- Os desenvolvedores abrem *features branches* que são mais *branches* de apoio do que uma *feature branch* propriamente dita.
- Essa *branch* de apoio provém da *master*, o desenvolvedor altera o código necessário da *feature* em que está trabalhando no momento;
- Uma vez que essas alterações não quebram o *build* e os testes passam, o desenvolvedor abre então um *merge request* para a *master*, onde acontece o processo de *code review*, uma vez completo e o *merge request* é aprovado;
- Então esse código entra na *master* e já pode ser baixado por todos os outros desenvolvedores.

- 
- O release desse código é feito direto da branch principal (*master* nesse caso), é criada uma tag da versão que será liberada e então o código pode ser promovido para produção.
 - **Importante: maturidade do time!!!!**

Vamos discutir...



- Você precisa escolher um modelo de ramificação para uma equipe nova;
- Há duas características que precisam ser seguidas:
 - As novas funcionalidades devem ser implementadas em uma nova *branch*
 - Cada funcionalidade deve ser revisada pela equipe antes de entrar no *mainline*