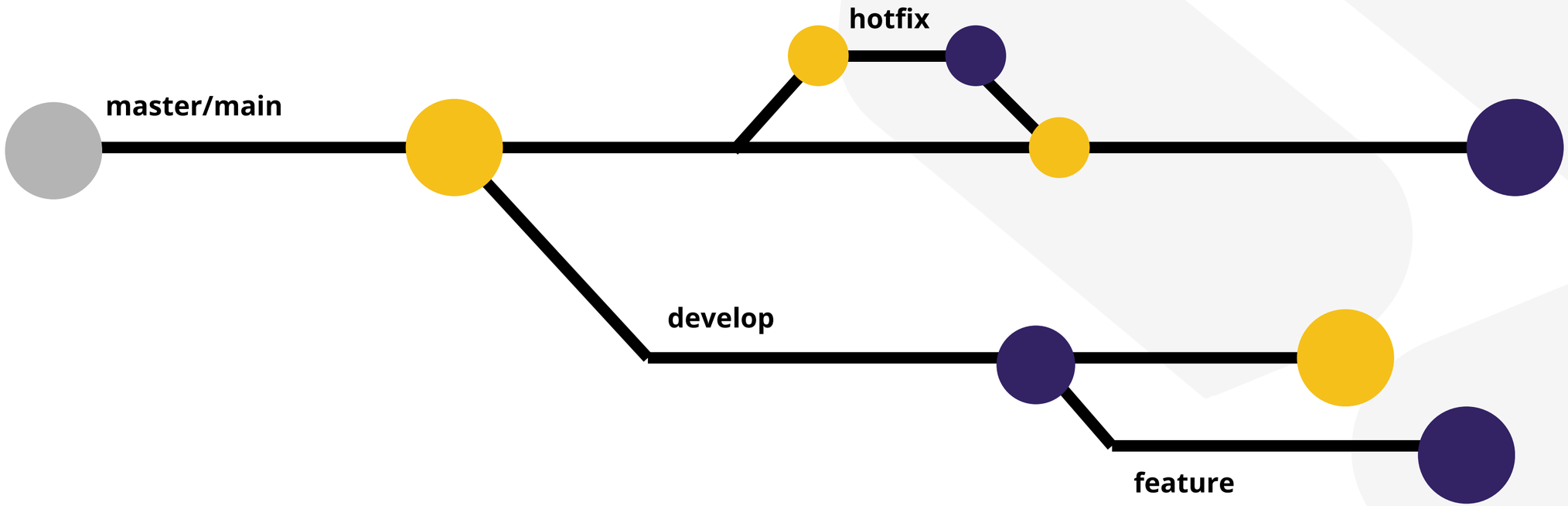
WebTech network

BRANCHES e GIT FLOW

Fluxo de trabalho, boas praticas de um desenvolvimento colaborativo

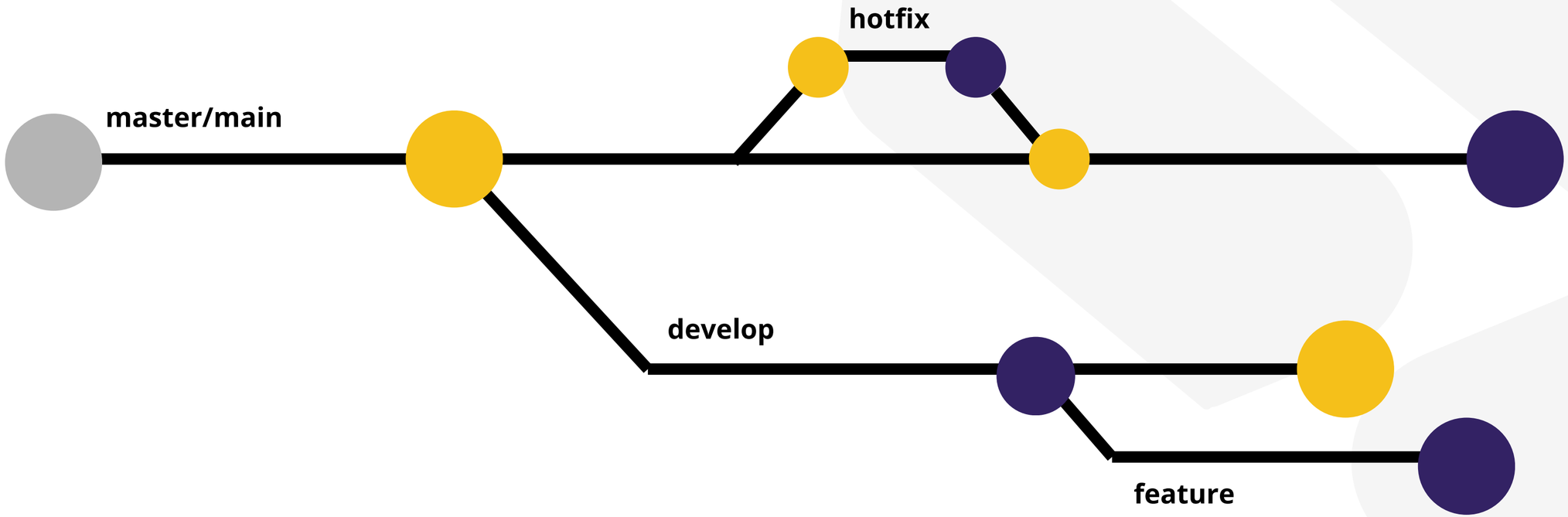
Porque isso é importante?

Trabalhar com versionamento é essencial quando falamos de trabalho em equipe, existem diversas abordagens que buscam organizar isso, uma delas é o GitFlow, mas antes disso precisamos entender alguns conceitos.

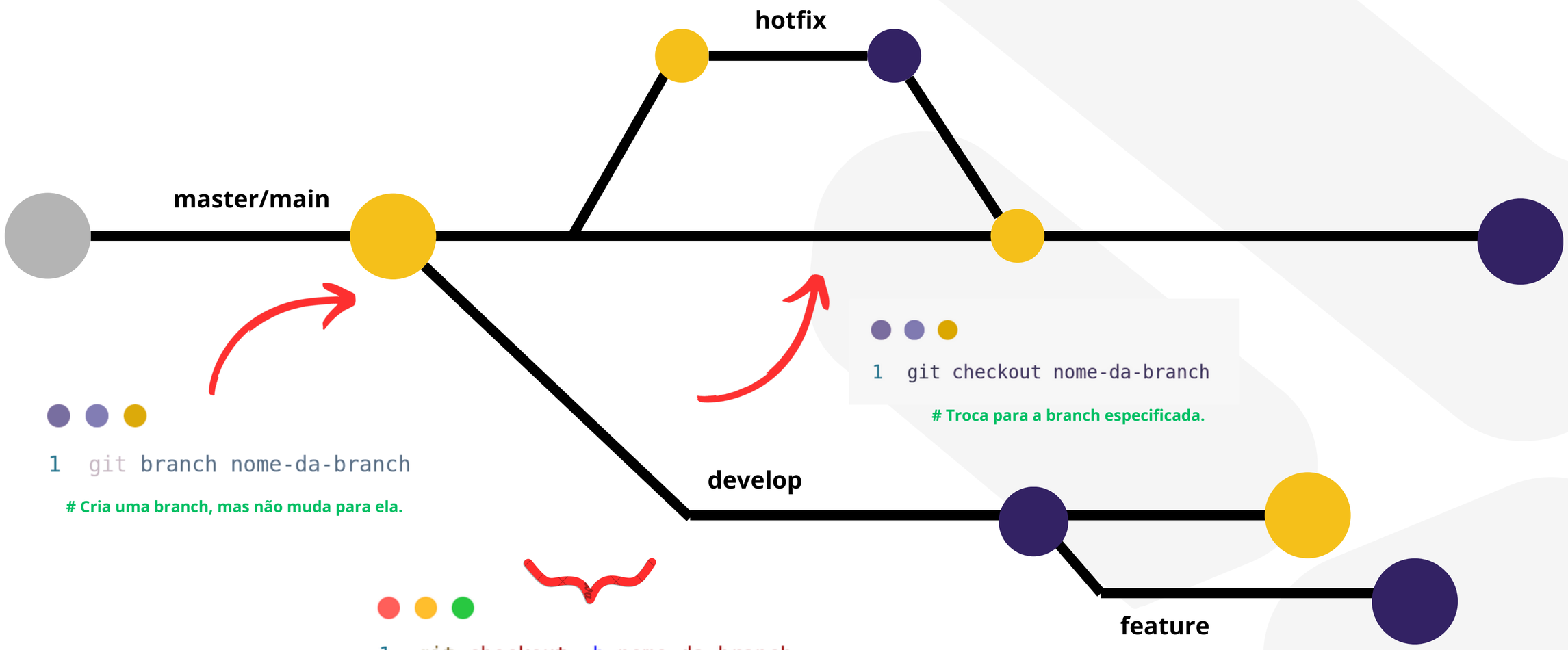


Relembrando: O que são Branches?

Branches são ramificações independentes do código, desenvolver projetos colaborativos atreves da organização de tarefas como adição de funcionalidades e correções de bugs a partir de branches permite um desenvolvimento seguro, ou seja mais integrado e com menor risco de afetar versões estáveis do produto



Criação e manipulação de branches




```
1  git branch nome-da-branch
```

Cria uma branch, mas não muda para ela.

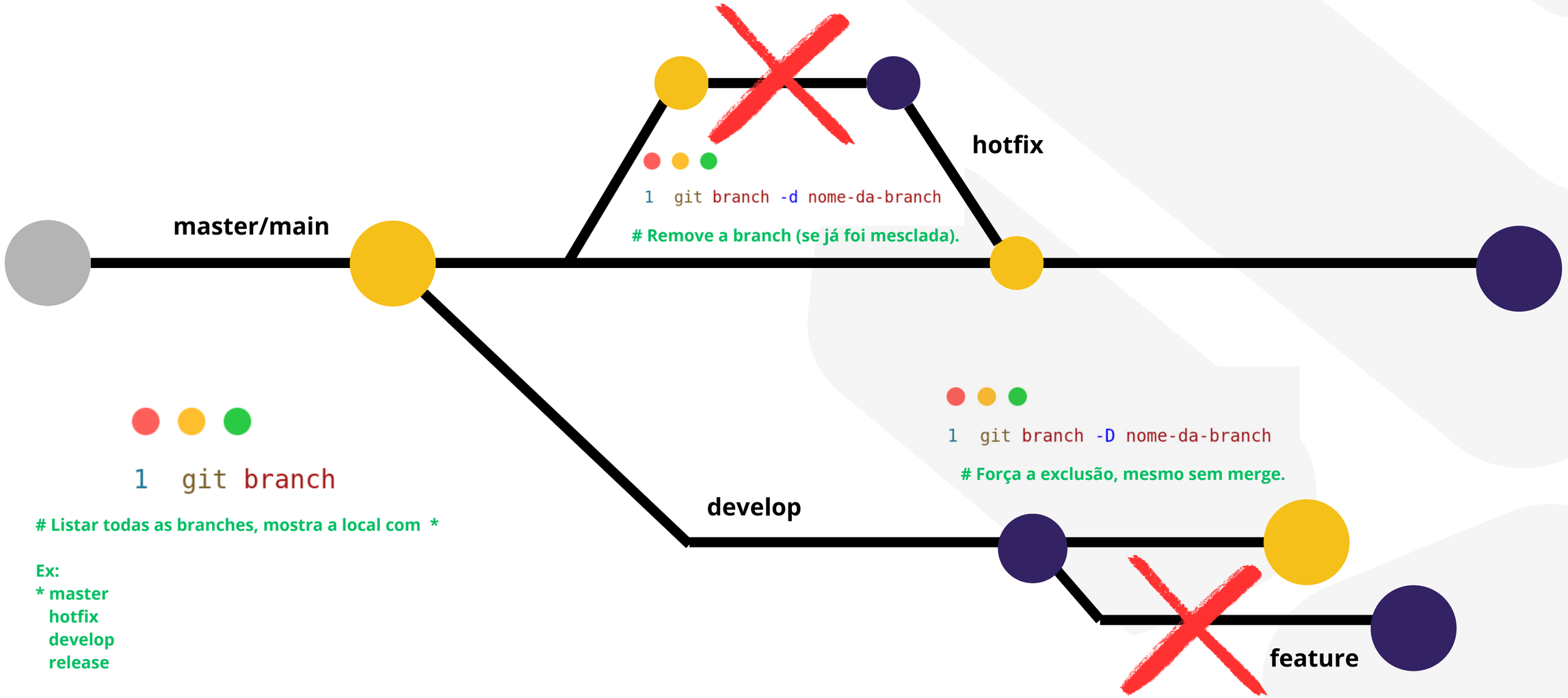
```
1  git checkout nome-da-branch
```

Troca para a branch especificada.

```
1  git checkout -b nome-da-branch
```

Criar e mudar para a branch ao mesmo tempo (muito usado no dia a dia).

Remoção e visualização de branches




```
1  git branch -d nome-da-branch
```

Remove a branch (se já foi mesclada).

```
1  git branch -D nome-da-branch
```

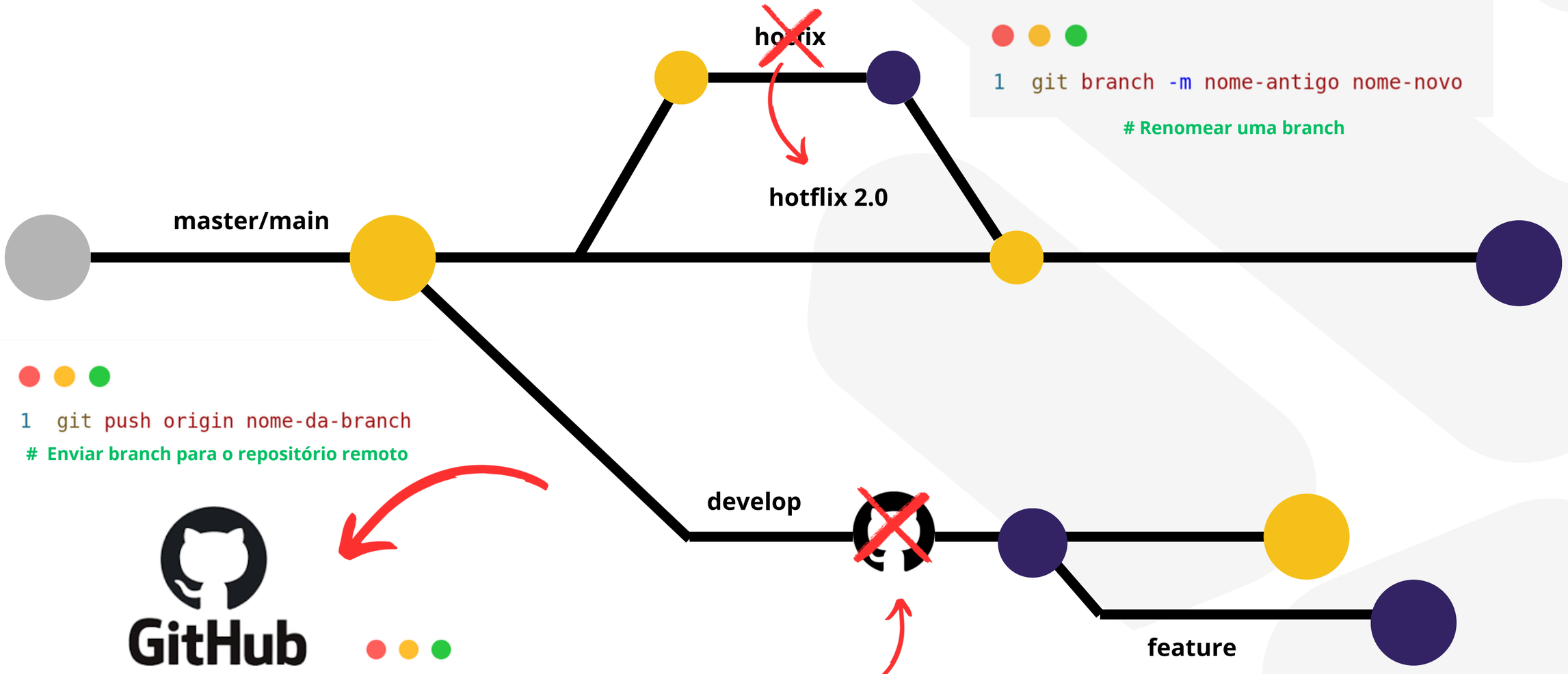
Força a exclusão, mesmo sem merge.

```
1  git branch
```

Listar todas as branches, mostra a local com *

Ex:
* master
  hotfix
  develop
  release

Demais comandos e funcionalidades




```
1  git branch -m nome-antigo nome-novo
```

Renomear uma branch

```
1  git push origin nome-da-branch
```

Enviar branch para o repositório remoto

```
1  git push origin --delete nome-da-branch
```

Excluir branch no remoto

Fluxos de Trabalho com Branches

Diversas abordagens buscam organizar o fluxo de trabalho, projetos grandes exigem um controle maior sobre a produção de branches ao atribuir funcionalidades/responsabilidades a branches especificas, tudo isso para garantir um maior controle de versionamento ao código produzido

Algumas dessas abordagens são:

- **Feature Branch Workflow:** cada funcionalidade em sua branch.
- **GitHub Flow:** branch main estável, cada PR **(Pull Request)** vem de uma branch nova.
- **GitFlow:** fluxo mais estruturado, ideal para projetos complexos.

Observações: Vale ressaltar que esse modelo não é fixo, cada equipe escolhe adaptar essa abordagem de forma a melhor se encaixar nas especificidades do seu projeto.

Davi Cândido de Almeida

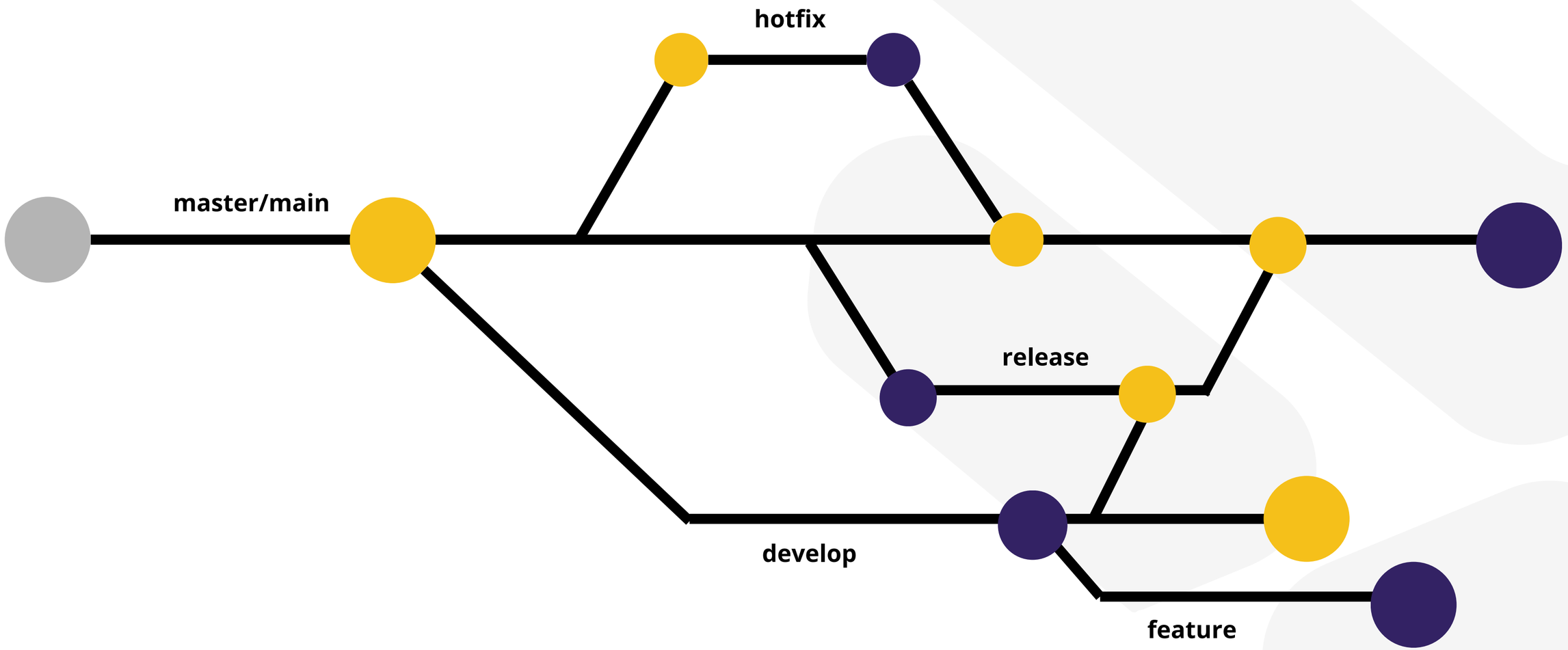
O que é GitFlow?

GitFlow nada mais é do que uma abordagem que busca distribuir responsabilidades, ou seja, estabelecer um padrão de criação e controle de branch para projetos mais complexos. Essa abordagem foi criado e divulgada por Vincent Driessen em 2010, buscando popularizar sua ótima estrategia de organização de projetos colaborativos

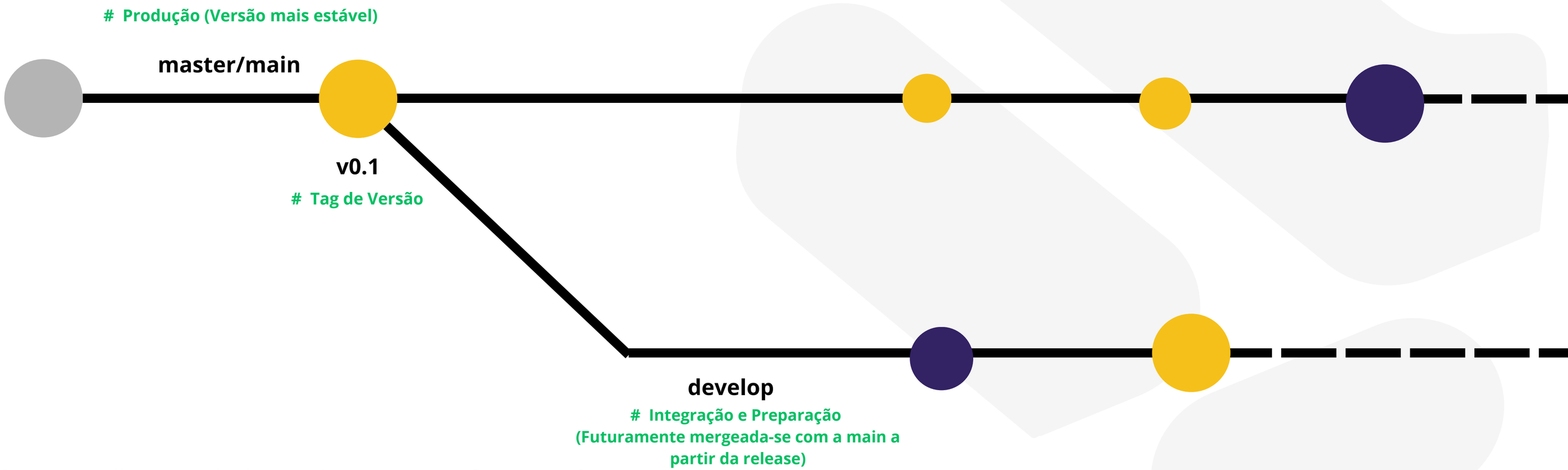
GitHub: https://github.com/nvie

Seu famoso repositório sobre GitFlow

Branches no GitFlow



Branches no GitFlow

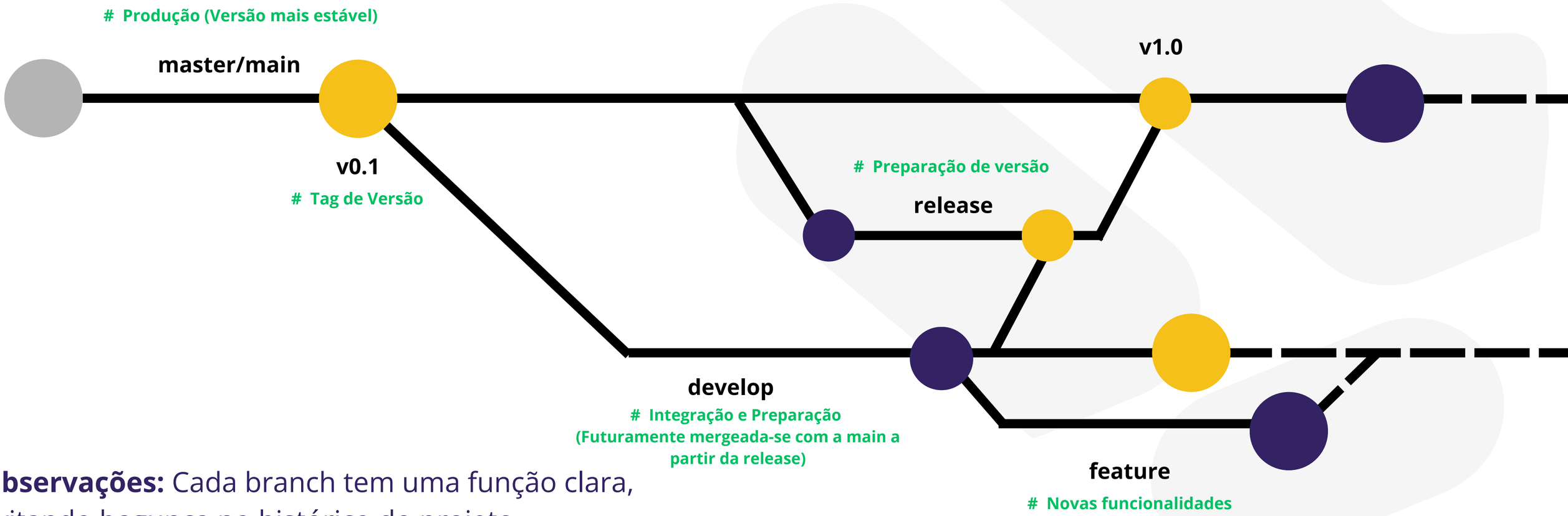


Observações: Cada branch tem uma função clara, evitando bagunça no histórico do projeto.

Branches no GitFlow

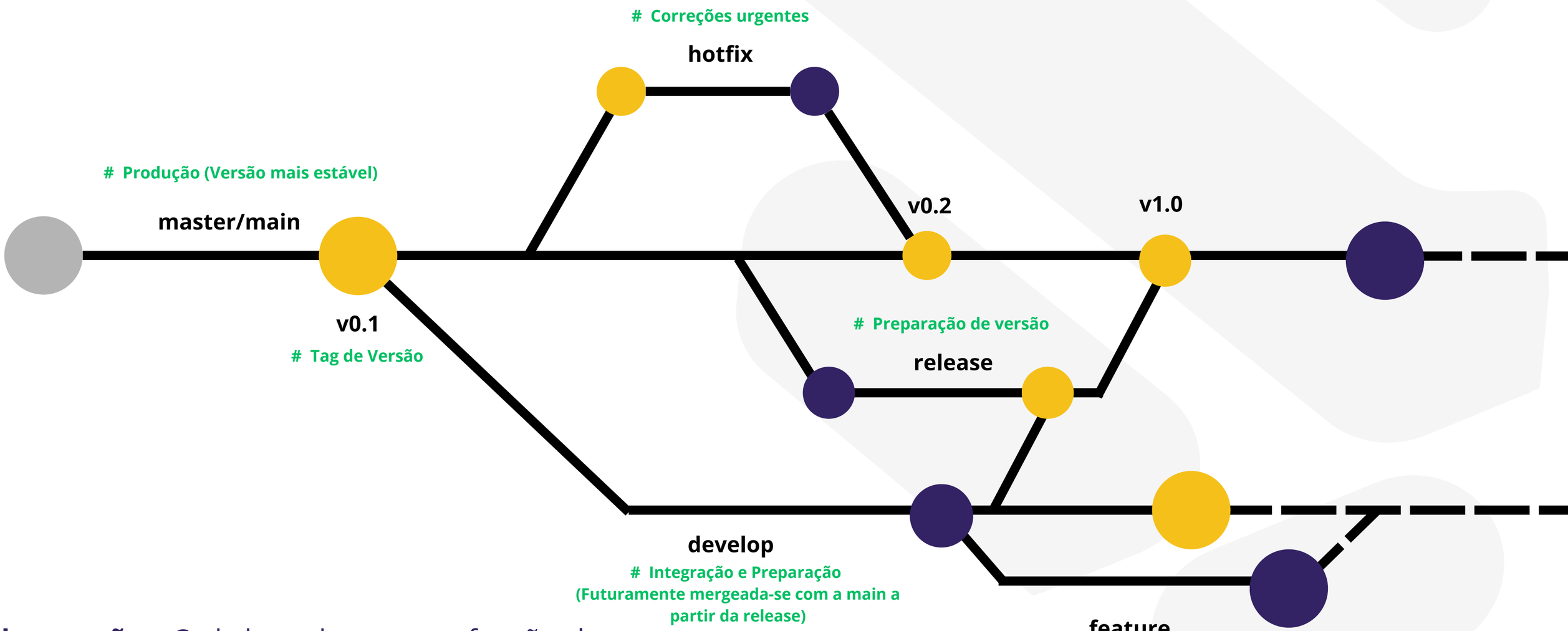
Observações: Cada branch tem uma função clara, evitando bagunça no histórico do projeto.

Branches no GitFlow



Observações: Cada branch tem uma função clara, evitando bagunça no histórico do projeto.

Branches no GitFlow



Observações: Cada branch tem uma função clara, evitando bagunça no histórico do projeto.

Vantagens do GitFlow

- **Organização clara.**
- **Bom para grandes times.**
- **Histórico limpo e previsível.**
- **Facilita releases. (Controle de correções)**

Desvantagens do GitFlow

- **Pode ser pesado em times pequenos.**
- **Mais merges e overhead.**
- **GitHub Flow pode ser melhor em projetos ágeis.**

Observações: O GitFlow é um modelo robusto, mas pode ser burocrático para projetos pequenos, não existe um fluxo perfeito, na maioria das vezes dependerá do contexto

Davi Cândido de Almeida

Extra: Commits semânticos:

Outra boa pratica é também estabelecermos um padrão para as mensagens de commits, ou seja, adotarmos uma convenção, facilitando assim a leitura do histórico e automação. Dessa forma cada commit descreve o que foi feito de forma clara e previsível

Formato Padrão:

<tipo>(escopo opcional): descrição curta

Sendo os tipos mais comuns:

- **feat:** adiciona uma nova funcionalidade.
- **fix:** corrige um bug.
- **docs:** mudanças apenas na documentação.
- **style:** mudanças de formatação/código (espaços, vírgulas, etc.), sem alterar lógica.
- **refactor:** refatoração de código, sem mudar comportamento externo.
- **test:** adição ou correção de testes.
- **chore:** tarefas de manutenção (build, configs, dependências).

Veja mais em:
https://github.com/nvie

Conclusão

- **Branches organizam o trabalho.**
- **Fluxos de trabalho bem feitos garantem previsibilidade.**
- **GitFlow é uma solução completa, mas deve ser usado com consciência.**
- **Adoção de semântica aos nosso commits garante histórico claro, padronizado e fácil de entender.**
- **Teste adapte o fluxo à realidade do seu time, o GitFlow não é uma regra mais um caminho para um a estruturação de projetos mais completos e robustos!**

Esse tutorial foi escrito por Davi Cândido – PUC Minas. Compartilhe com colegas desenvolvedores!

Se aprofunde mais...

- Git Flow: entenda o que é, como e quando utilizar | Alura
- Utilizando o GitFlow por Mikael Hadler | Medium
- Git Flow | Dicionário do Programador
- Saiba tudo sobre o Gitflow Workflow | Atlassian Git Tutorial
- iuricode/padroes-de-commits
- O que é Commit e como usar Commits Semânticos - Blog de TI

Esse tutorial foi escrito por Davi Cândido – PUC Minas. Compartilhe com colegas desenvolvedores!