

# Projeto Integrado III

DD - UFC - Quixadá



**Prof.: Aníbal Cavalcante**

# Sistemas de Controle de Versões (SCV)



# O que é um SCV?

Um **sistema de controle de versão** (SCV) é um software para **gerenciar** diferentes **versões** de um documento qualquer durante seu **desenvolvimento**.

São utilizados no desenvolvimento de software para controlar as diferentes versões de:

- **códigos-fontes**
- **documentação**

# Benefícios

- Histórico de Modificações (Máquina do tempo do seu código).
- Permite o trabalho em equipe sem conflitos e sem bagunçar o código.
- Marcação e resgate de versões estáveis.
- Ramificação de projeto (linhas de desenvolvimento)

# Funcionamento Básico de SCV - Repositório Centralizado



# O Git - Um SCV distribuído.



- O Git é SCV distribuído.
- Foi criado por Linus Torvalds em 1995 para gerenciar o código-fonte do Kernel do Linux.
- É um software livre e pode ser instalado em qualquer servidor.
- Foi escrito em C, Pearl e Shell.
- A versão atual estável do Git é 2.22.0.
- Sua manutenção é atualmente supervisionada por **Junio Hamano**.

# O Git - Um SCV distribuído.

Os seguintes websites provêm hospedagem gratuita de código fonte para repositório Git:

BerliOS, **GitHub**, Gitorious, **Sourceforge**, GNU Savannah, Project Kenai, Unfuddle, SourceRepo, Google Code, **Bitbucket**, **GitLab** e Azure DevOps

# Características do Git

## 1 - Desenvolvimento distribuído

Cada desenvolvedor possui uma cópia local completa de todo o histórico de desenvolvimento.

Os desenvolvedores podem criar novas versões (branches) para desenvolver novas funcionalidades e depois realizar mesclas (merges) na versão completa do software.

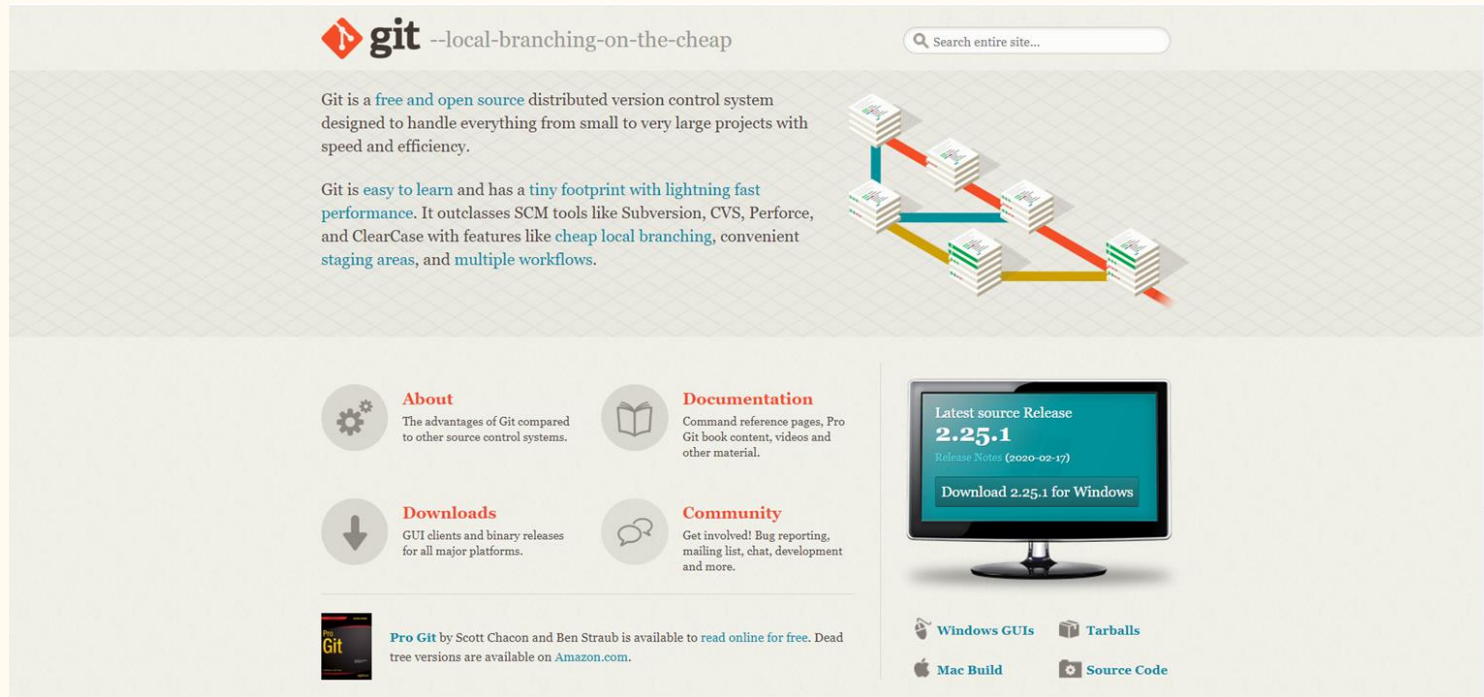


# Exemplo de como o Git funciona.



# Baixando o Git para sua máquina.

<https://git-scm.com/>



The screenshot shows the Git website homepage. At the top left is the Git logo (a red octocat) followed by the text "--local-branching-on-the-cheap". To the right is a search bar with the placeholder text "Search entire site...". Below the logo, there is a paragraph describing Git as a "free and open source" distributed version control system. To the right of this text is a diagram showing a branching model with stacks of code and colored lines representing branches. Below the description, there are four circular icons with corresponding text: "About" (gears), "Documentation" (book), "Downloads" (down arrow), and "Community" (speech bubbles). On the right side, there is a monitor displaying the "Latest source Release 2.25.1" and a button to "Download 2.25.1 for Windows". At the bottom left, there is a small image of the "Pro Git" book cover. At the bottom right, there are four icons with labels: "Windows GUIs", "Tarballs", "Mac Build", and "Source Code".

**git** --local-branching-on-the-cheap

Search entire site...

Git is a **free and open source** distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

Git is **easy to learn** and has a **tiny footprint with lightning fast performance**. It outclasses SCM tools like Subversion, CVS, Perforce, and ClearCase with features like **cheap local branching**, convenient staging areas, and **multiple workflows**.

**About**  
The advantages of Git compared to other source control systems.

**Documentation**  
Command reference pages, Pro Git book content, videos and other material.

**Downloads**  
GUI clients and binary releases for all major platforms.

**Community**  
Get involved! Bug reporting, mailing list, chat, development and more.

**Latest source Release**  
**2.25.1**  
Release Notes (2020-02-17)  
[Download 2.25.1 for Windows](#)

**Pro Git** by Scott Chacon and Ben Straub is available to [read online for free](#). Dead tree versions are available on [Amazon.com](#).

[Windows GUIs](#) [Tarballs](#)  
[Mac Build](#) [Source Code](#)

# Configurando o Git pela primeira vez...

## Sua Identidade

A primeira coisa que você deve fazer ao instalar Git é configurar seu nome de usuário e endereço de e-mail. Isto é importante porque cada `commit` usa esta informação, e ela é carimbada de forma imutável nos `commits` que você começa a criar:

```
$ git config --global user.name "Fulano de Tal"
$ git config --global user.email fulanodetal@exemplo.br
```

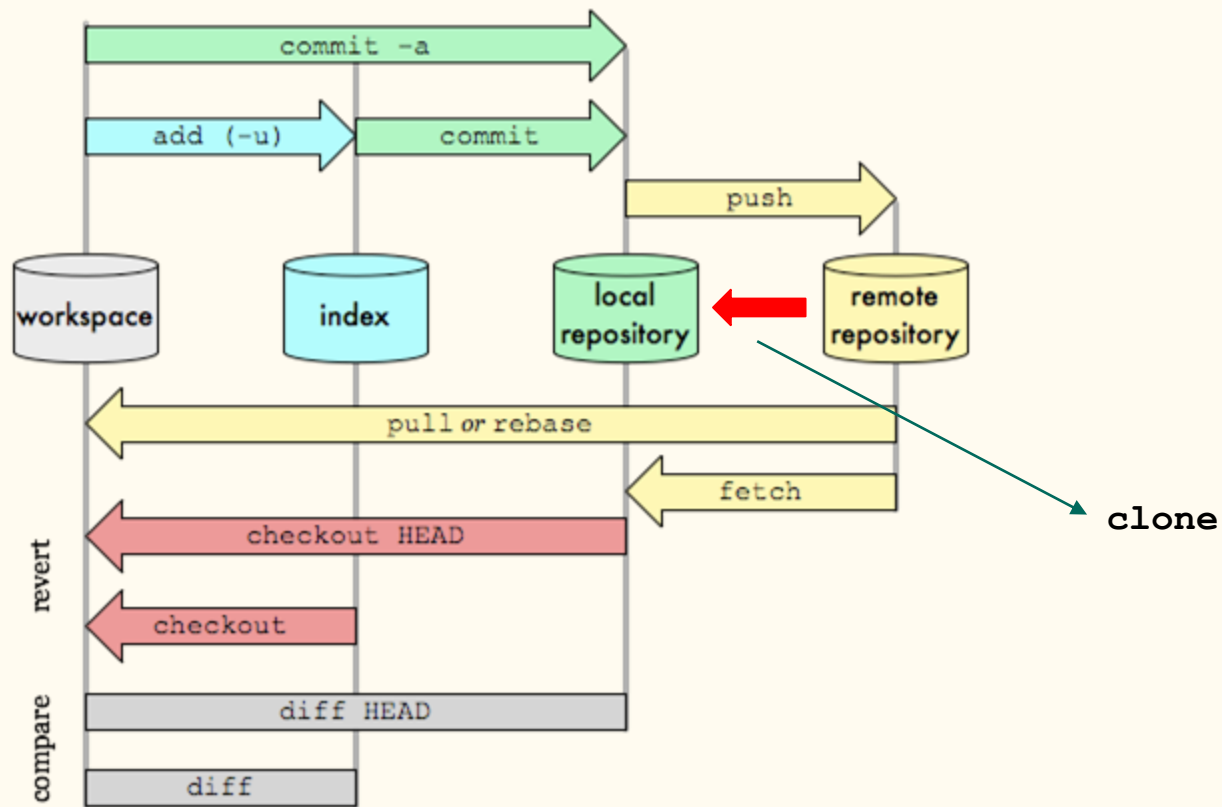
Reiterando, você precisará fazer isso somente uma vez se tiver usado a opção `--global`, porque então o Git usará esta informação para qualquer coisa que você fizer naquele sistema. Se você quiser substituir essa informação com nome diferente para um projeto específico, você pode rodar o comando sem a opção `--global` dentro daquele projeto.

Muitas ferramentas GUI o ajudarão com isso quando forem usadas pela primeira vez.

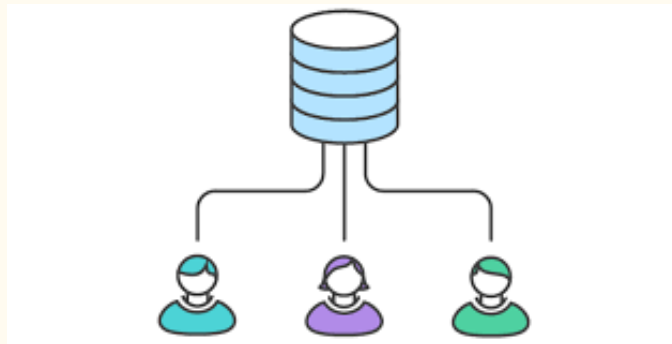
# Fluxo de Trabalho do Git (Git Workflow)



# Fluxo de Trabalho do Git (Git Workflow)

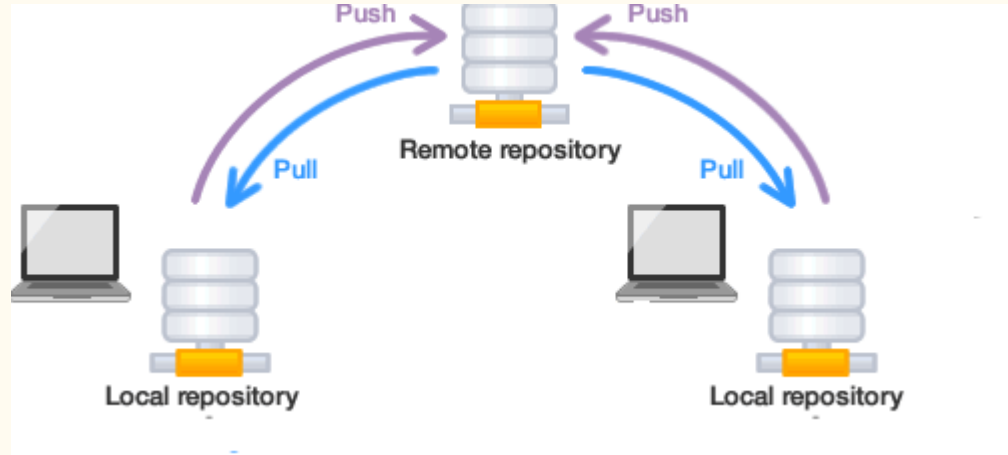


# 1 - Repositório Remoto



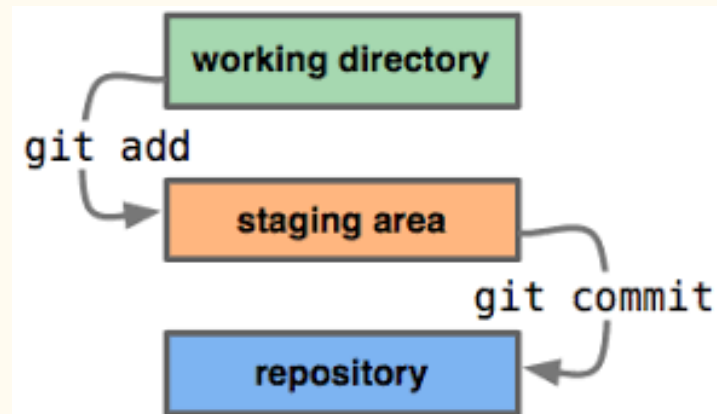
- 1. É onde ficam salvos os arquivos de forma centralizada.
- 1. Os desenvolvedores podem obter uma cópia desses arquivos utilizando o comando “clone”.
- 1. Um histórico de todos os arquivos é mantido nesse repositório, assim como, os responsáveis pelas alterações.

## 2 - Repositório Local



1. Inicialmente é uma cópia do repositório remoto.
2. Cada usuário tem uma cópia desse repositório.
3. É o nele que realizamos os commits.
4. A cada commit, o histórico dos arquivos fica salvo localmente.
5. Para compartilhar a sequência de commits com outros membros da equipe utiliza-se o comando “Push”.

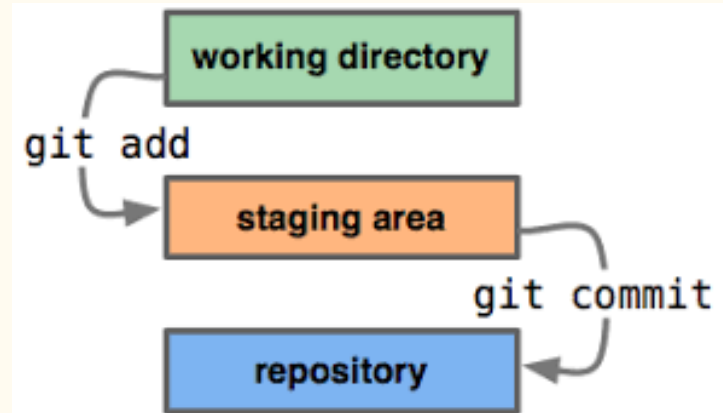
### 3 - Index ou Stage Area (Área de Espera)



1. Faz o meio de campo entre o Workspace e o repositório local.
2. Gerencia quais arquivos deverão ser:
  - a. “commitados”
  - b. ignorados (Não versionados)
  - c. incluídos (arquivos recentemente criados que serão incluídos no repositório)



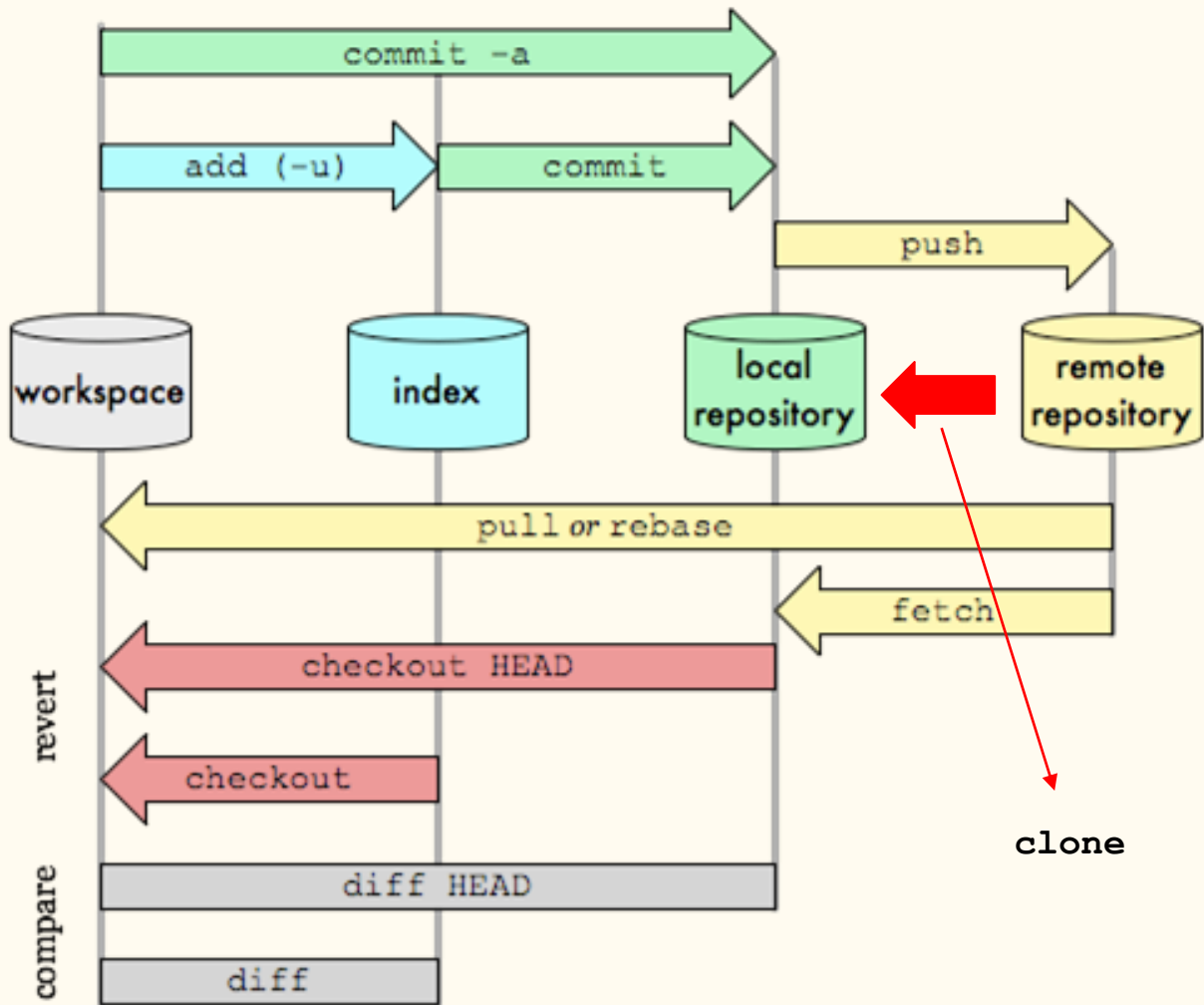
## 4 - Workspace



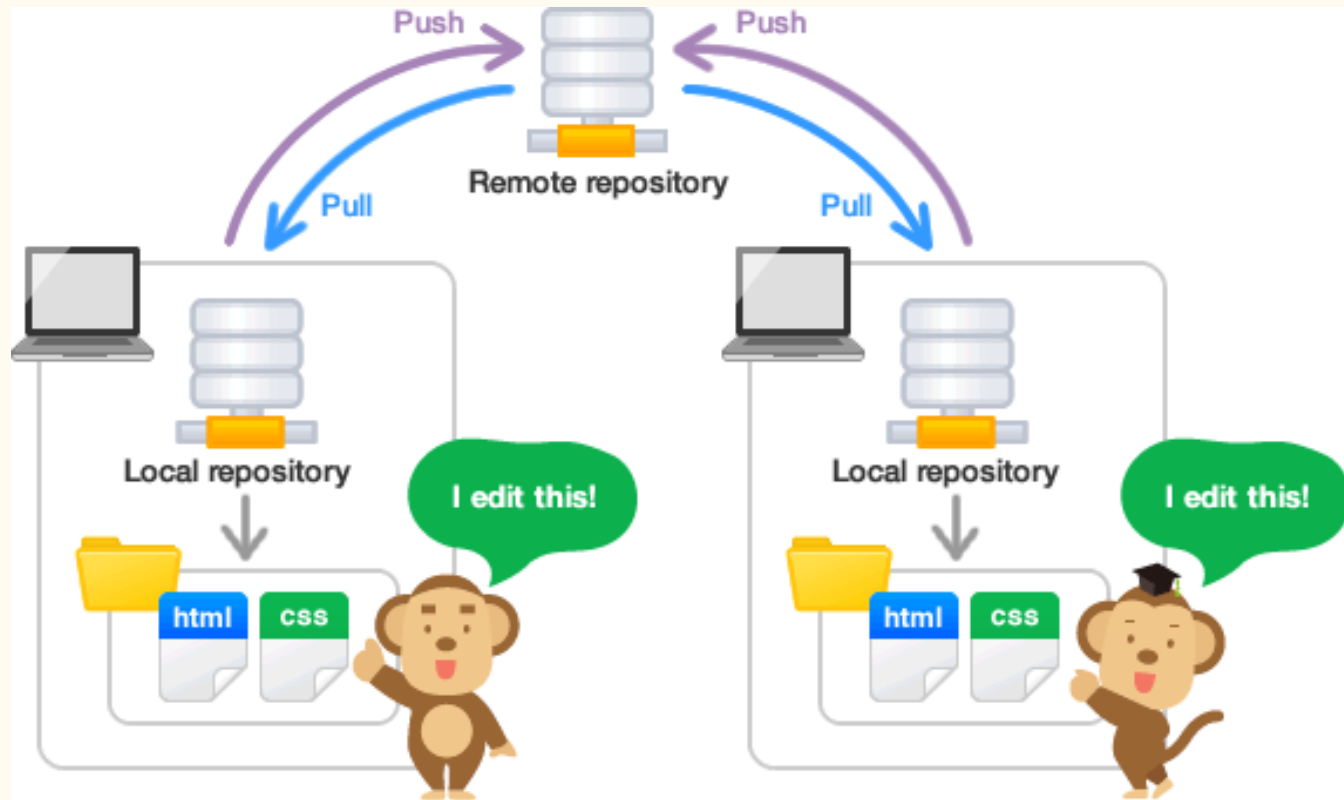
1. É o diretório local onde os seu códigos estão sendo editados.
2. Inicialmente ele está igual ao repositório local.
3. Modificações nos arquivos do workspace, podem ser comparadas com o repositório local.
4. A comparação é feita graças ao index.
5. Novos arquivos criados no workspace devem ser adicionados à área de testes, através do comando **“add”**

# Git Workflow

1. clone
2. add (-u) # git < 2.0
3. commit (-a)
4. push
5. pull or rebase
6. fetch
7. checkout HEAD
8. checkout
9. diff HEAD
10. diff
11. checkout -b <branch>



O cenário até o momento é esse, mas....



# Trabalhando com Ramos (Branches)

Trunk



# Trabalhando com Ramos (Branches)



**Repositório Remoto**

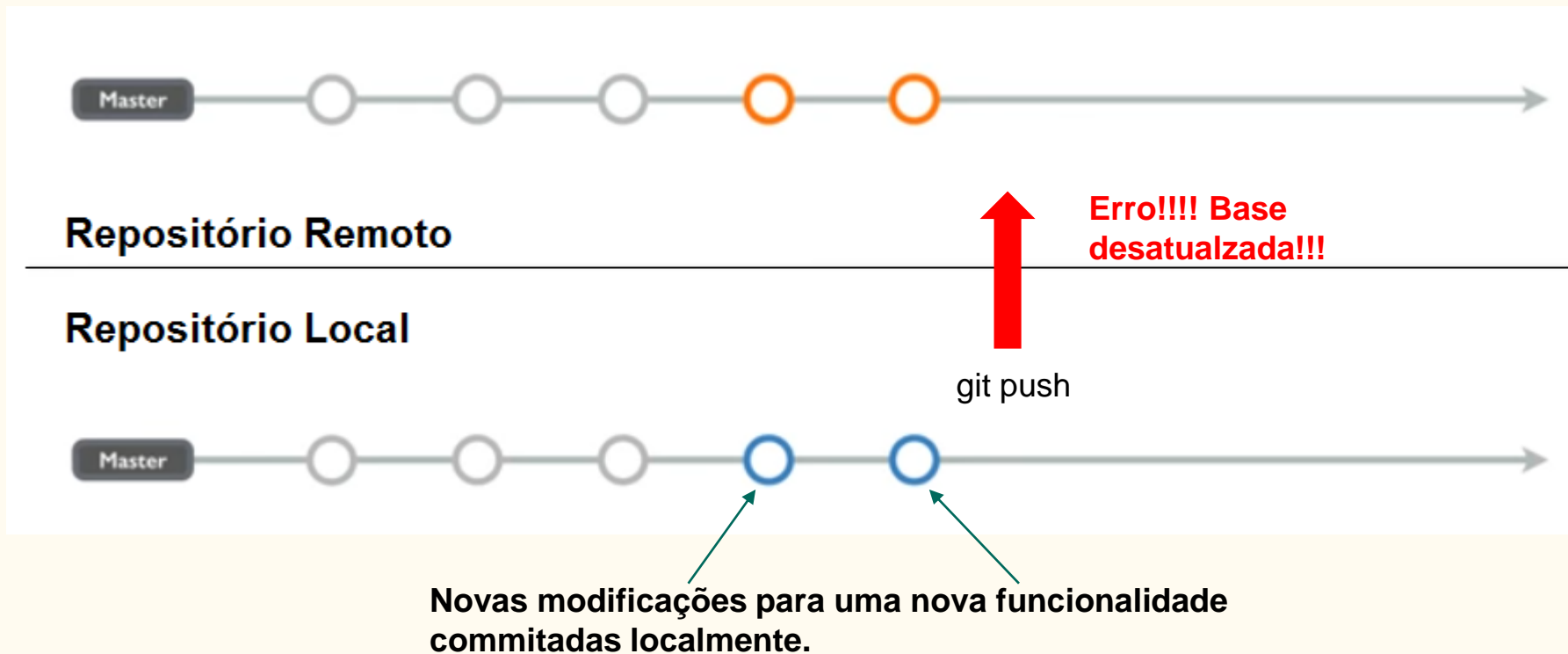
---

**Repositório Local**

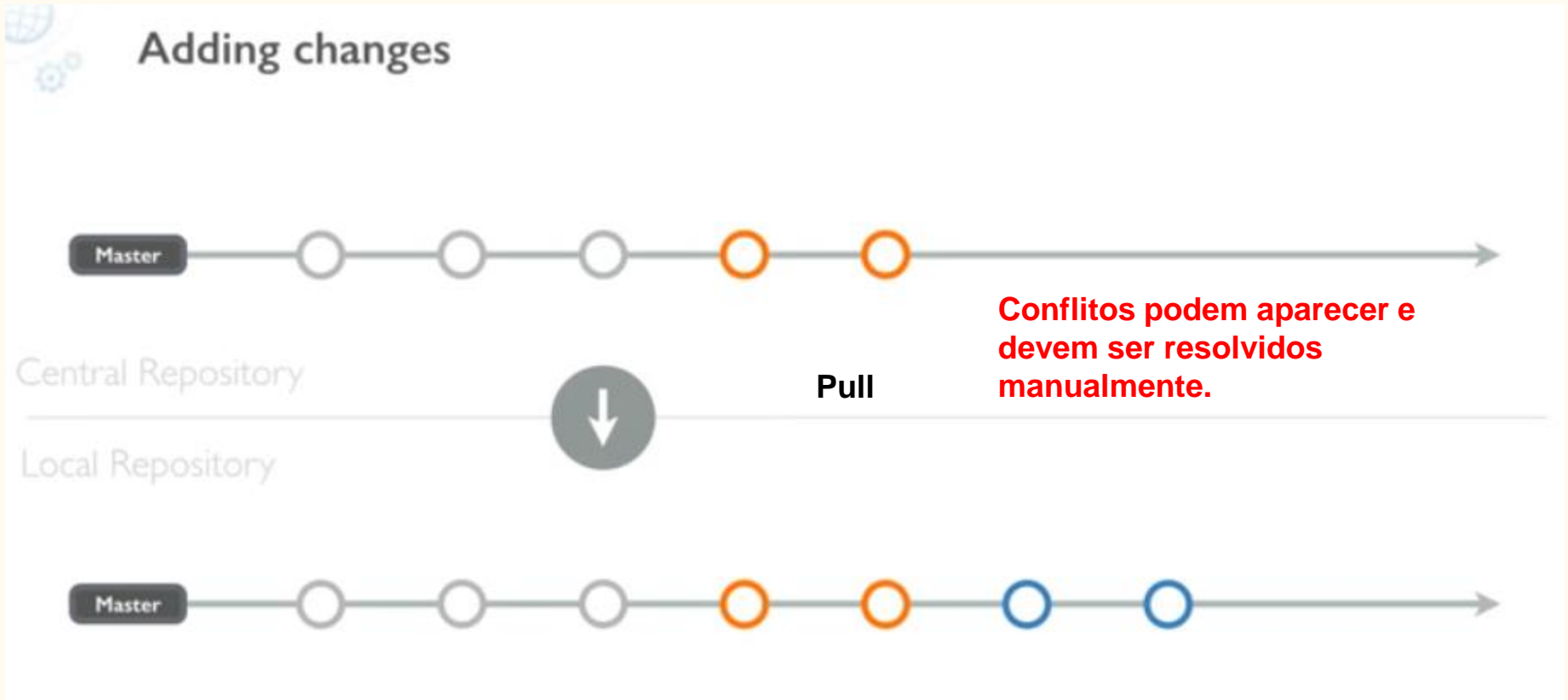


**Novas modificações para uma nova funcionalidade  
commitadas localmente.**

# Trabalhando com Ramos (Branches)



# Trabalhando com Ramos (Mesclagem Otimista)

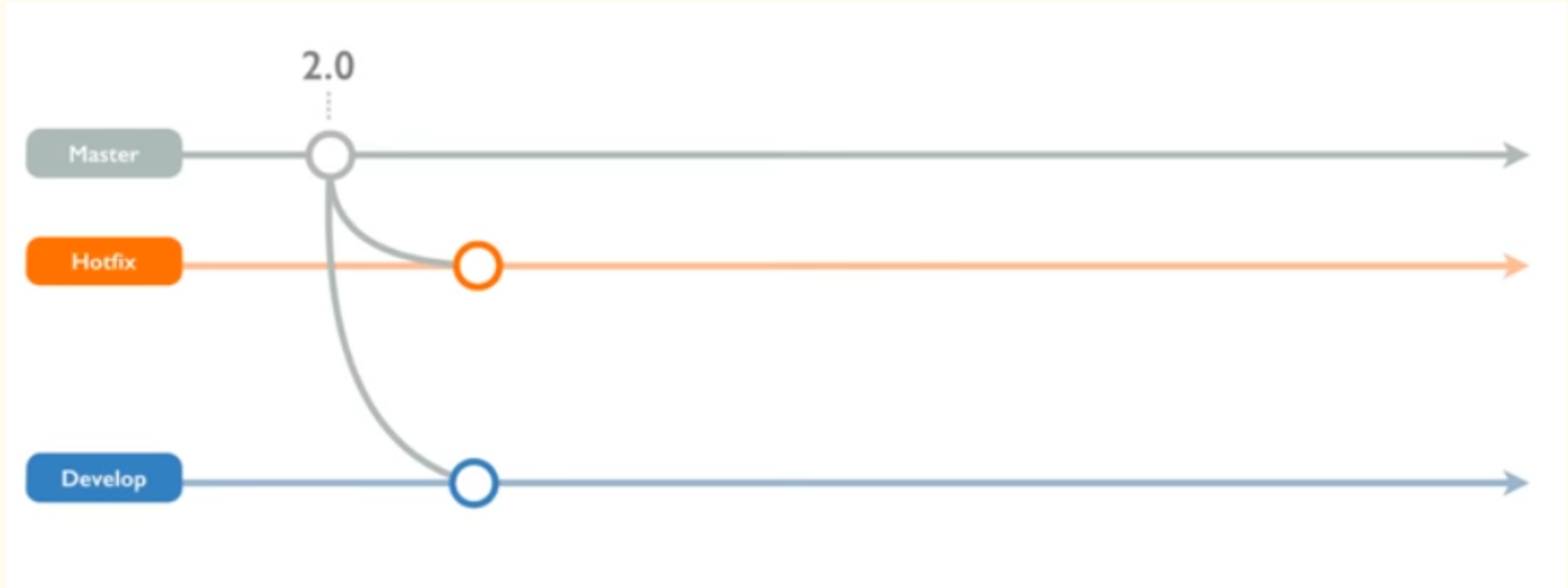


# Linhas de trabalho em paralelo com branches de desenvolvimento

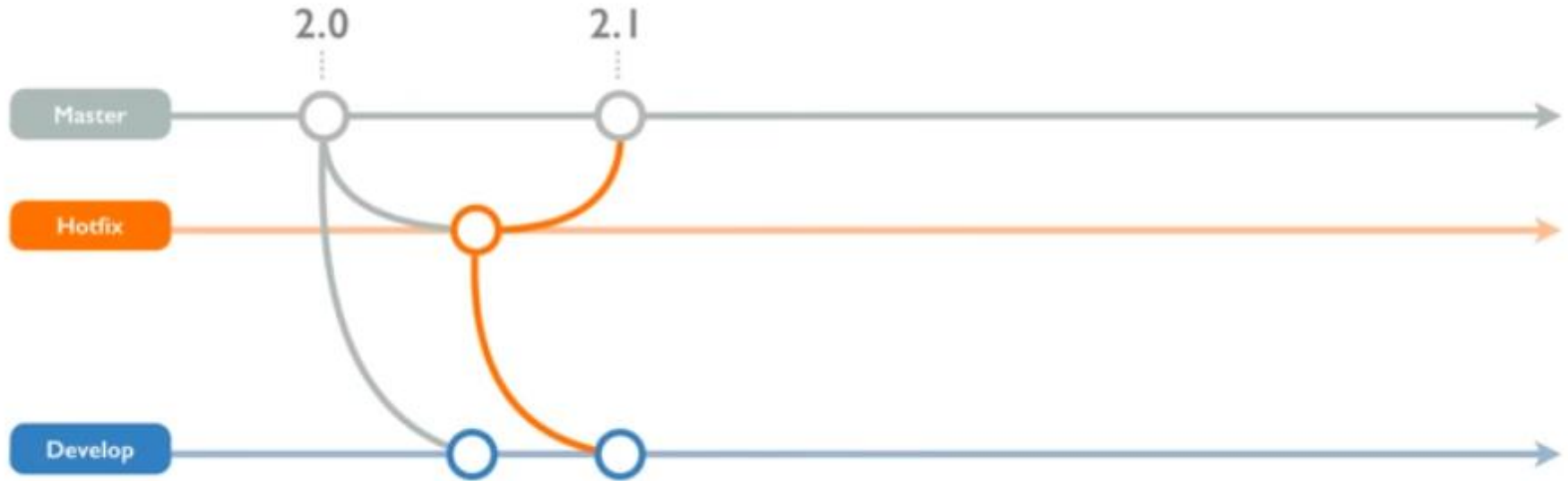




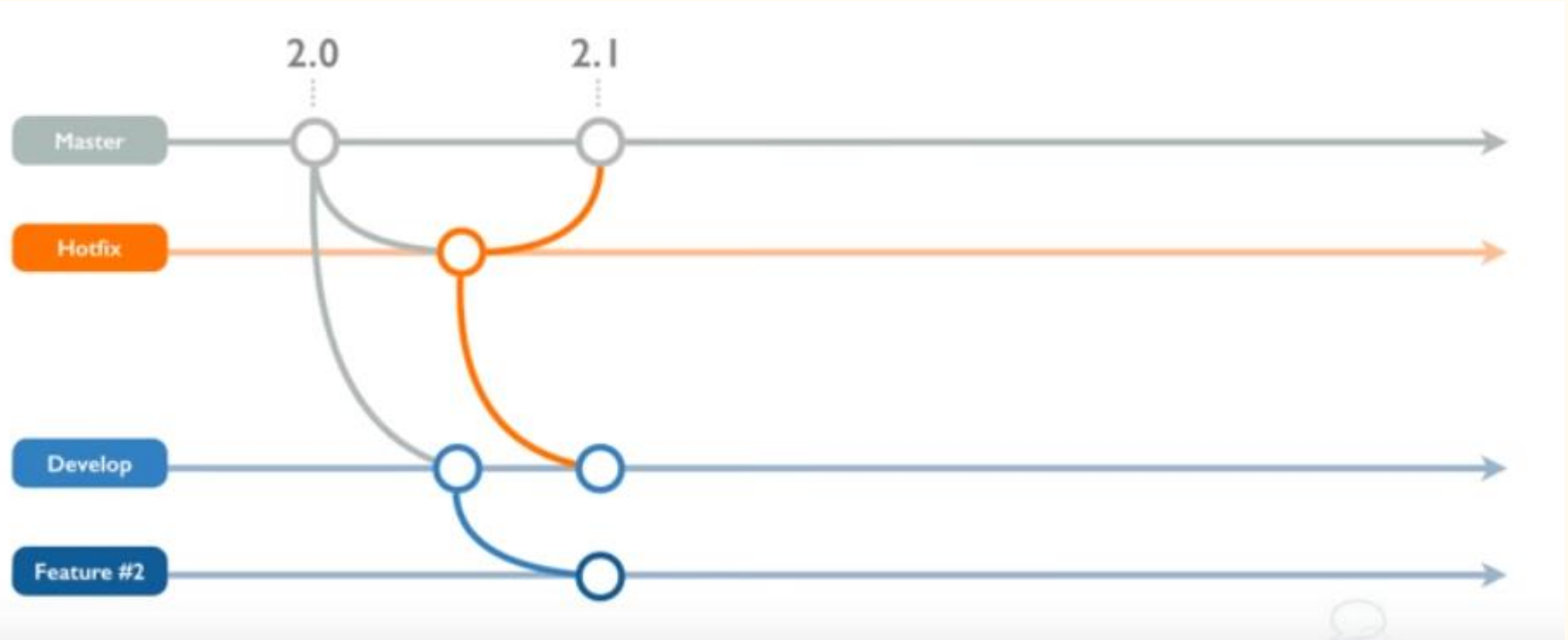
# Linhas de trabalho em paralelo com branches de desenvolvimento



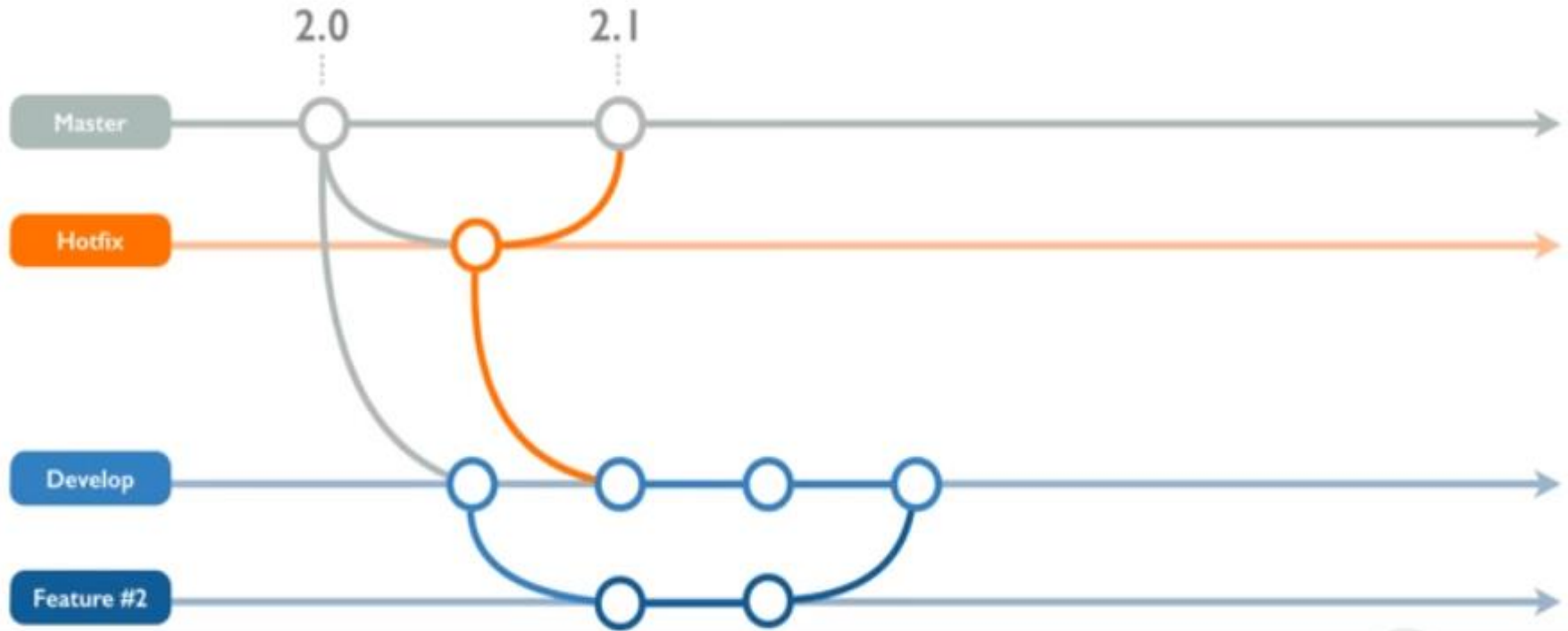
# Linhas de trabalho em paralelo com branches de desenvolvimento



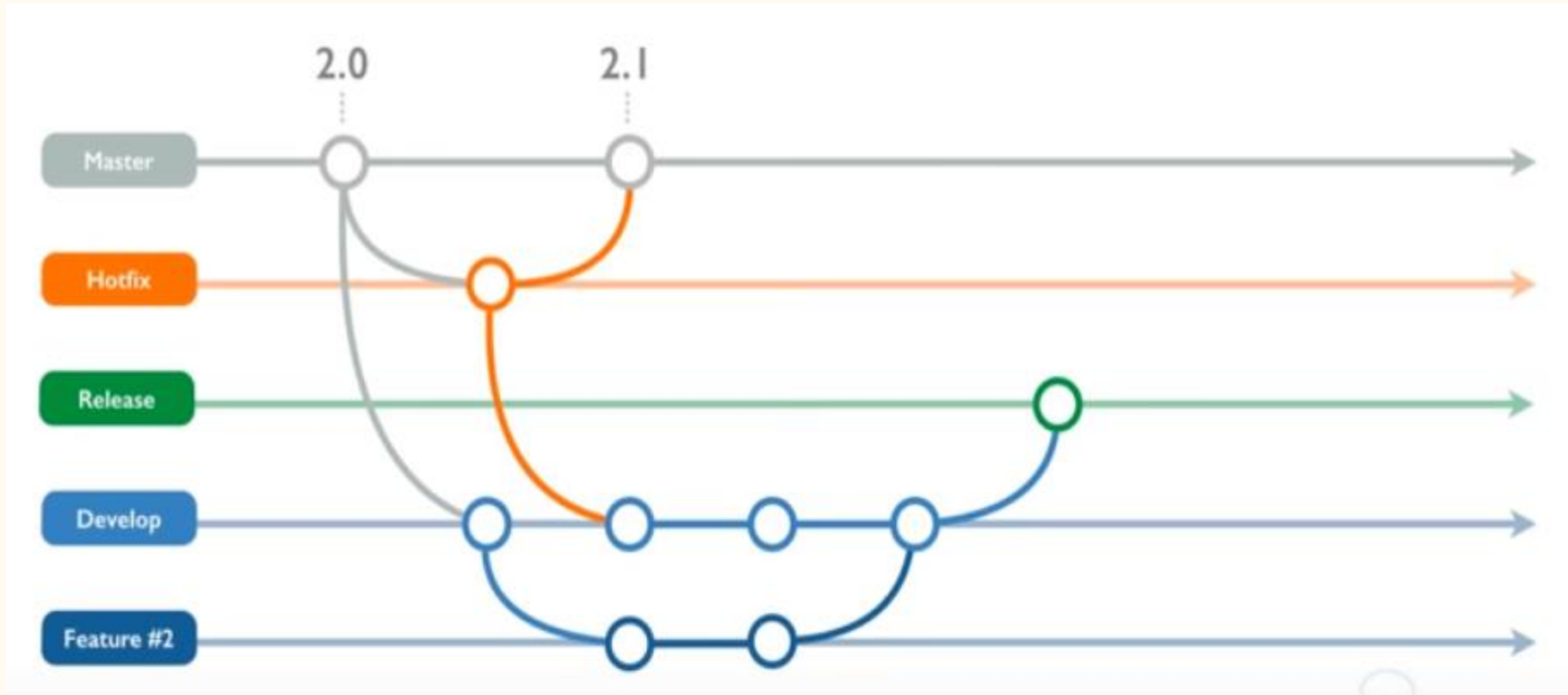
# Linhas de trabalho em paralelo com branches de desenvolvimento



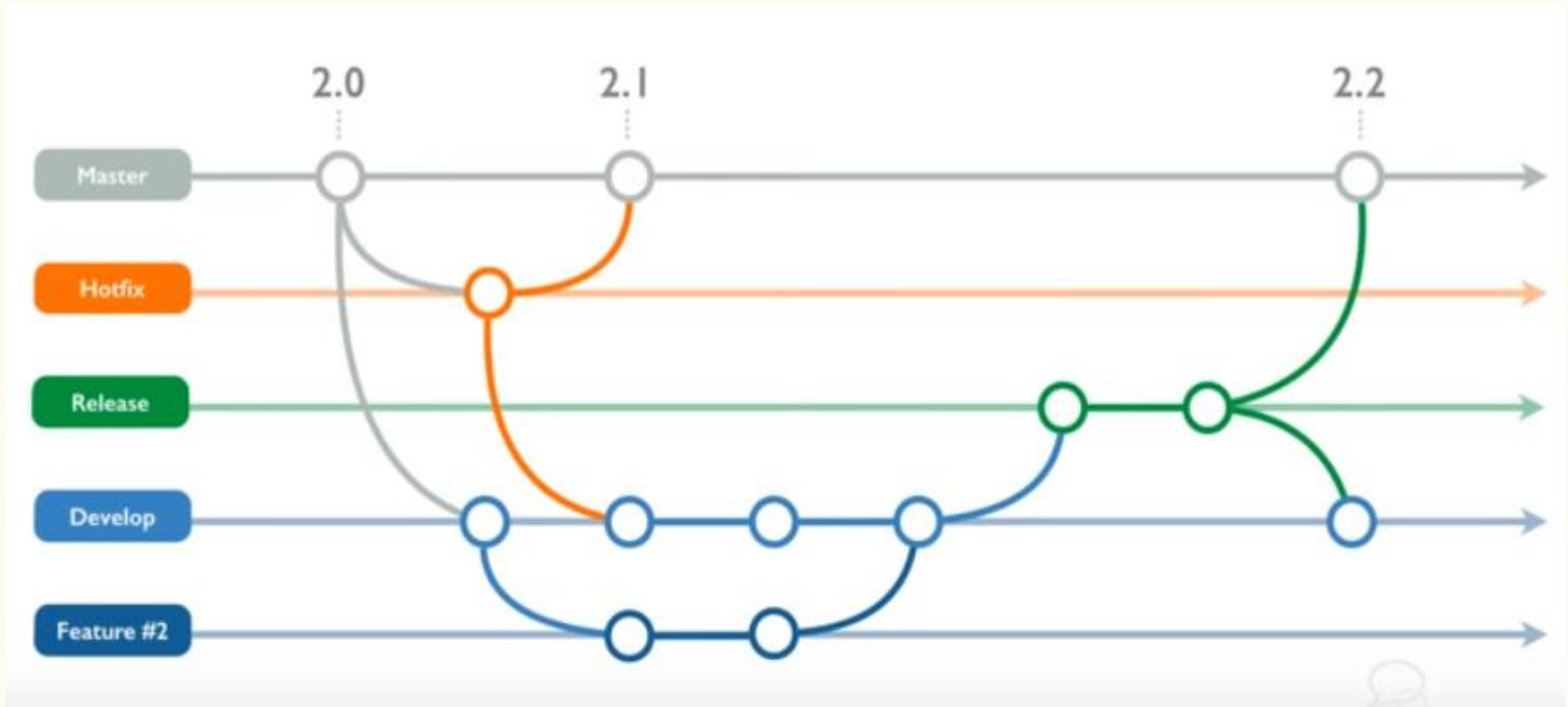
# Linhas de trabalho em paralelo com branches de desenvolvimento



# Linhas de trabalho em paralelo com branches de desenvolvimento



# Linhas de trabalho em paralelo com branches de desenvolvimento



# Práticas com Git utilizando o Github