

VERSIONAMENTO DE CÓDIGO FONTE

A central red diamond logo with the word "git" in white lowercase letters is positioned in the middle of the slide. Surrounding the logo is a complex network diagram consisting of numerous small colored circles (red, green, blue, yellow) connected by thin lines of the same colors, creating a web-like structure that extends across the background.

Marisangila Alves, MSc

marisangila.alves@edu.sc.senai.br

Versionamento de Código Fonte

O que é?

- Processo de **gerenciar mudanças** feitas no código fonte de um projeto.
- Registra cada modificação para que o histórico completo de desenvolvimento seja mantido.

Por que?

GitHub



GitLab



BitBucket



DropBox



Google Drive



sharing code
through emails



Por que?

- **Rastreabilidade:**
 - Permite identificar quem fez cada alteração e quando.

Por que?

- **Recuperação de Versões Anteriores:**
 - Facilita o retorno a **versões anteriores** do código, se necessário.
 - Permite **reversão** de alterações problemáticas.

Por que?

- **Colaboração Eficiente:**

- Desenvolvedores podem trabalhar **simultaneamente** em partes diferentes do projeto sem sobrescrever o trabalho uns dos outros.
- Facilita a **divisão do trabalho** em equipes, com cada membro podendo contribuir em **paralelo**.

Por que?

- **Manutenção e Evolução do Código:**
 - Suporta o **crescimento** do código ao longo do tempo.
 - É possível criar **versões e patches**.

Quais os benefícios?

- **Suporte para revisões:**
 - Permite que cada contribuição passe por **controle de qualidade** antes de ser incorporada ao código principal.

**Um pouco de
história...**

Como era?

- **Métodos Manuais:**

- No início, o versionamento era feito manualmente (cópias de segurança de pastas e arquivos).

- **Dificuldades:**

- Trabalhoso, sujeito a erros humanos e limitado para colaboração.

Quando mudou?

Surgimento do Primeiro Sistema: SCCS em 1972 criado por Marc Rochkind e Laboratórios Bell Labs.

- Armazenava versões de arquivos de código-fonte.
- Permitia o gerenciamento de múltiplas versões, facilitando a colaboração.
- Introduziu a ideia de rastrear e controlar mudanças no código, sendo o primeiro sistema de controle de versão da história.

Quando mudou?

Em seguida surgiu o RCS no ano de 1982, criado por Walter F. Tichy

- Armazenamento de mudanças em arquivos individuais.
- Utilização de um modelo de armazenamento delta, que reduz o espaço em disco ao armazenar apenas as diferenças entre as versões.

Quando mudou?

Transição para Sistemas Distribuídos

- Com o aumento da colaboração em projetos de software, surgiu a necessidade de sistemas que permitissem um trabalho mais flexível e descentralizado.

Quando mudou?

Surgiu então o **Git** no ano de 2005 criado por Linus Torvalds.

- Cada desenvolvedor tem uma cópia completa do repositório.
- Permite trabalho *offline* e maior segurança.
- Introduziu o conceito de *branches* de forma eficiente, facilitando o desenvolvimento paralelo.
- O Git se tornou o padrão na indústria, promovendo a colaboração em larga escala e suportando projetos *open-source*.

Sistemas Centralizados

- CVS (*Concurrent Versions System*), SVN (*Subversion*).
- Repositório central onde todos os desenvolvedores acessam e salvam alterações.
- Colaboração facilitada e histórico centralizado.
- Dependência do repositório central (problemas de acesso e falhas).

Sistemas Distribuídos

- Git, Mercurial.
- Cada desenvolvedor tem uma cópia completa do repositório, facilitando o trabalho *offline* e a fusão de mudanças.
- Maior flexibilidade, descentralização e maior tolerância a falhas (não depende de um único ponto).
- Amplamente adotado pela capacidade de suportar grandes equipes e fluxos de trabalho distribuídos, principalmente em projetos *open-source*.

Repositório Remoto

O que é?

É uma cópia do repositório de código em um servidor que pode ser acessado por outros desenvolvedores via [internet](#).

O que faz?

- **Sincronização de Código:** Permite sincronizar o trabalho de desenvolvedores em diferentes máquinas.
- **Backup e Segurança:** Serve como uma cópia segura e centralizada dos dados, reduzindo o risco de perda de trabalho.
- **Colaboração:** Facilita o trabalho colaborativo, permitindo que vários desenvolvedores contribuam para o mesmo projeto.
- **Histórico Unificado:** Todos os *commits* realizados localmente podem ser enviados para o repositório remoto, centralizando o histórico de mudanças.

Quais?

Plataformas Populares de Repositórios Remotos:

- **GitHub**: Muito popular para projetos *open-source* e colaboração em equipe.
Atualmente, pertence a Microsoft.



Quais?

Plataformas Populares de Repositórios Remotos:

- **GitLab**: Oferece recursos para DevOps e integrações CI/CD, amplamente usado para desenvolvimento interno.
 - Projeto *Open Source*.



Quais?

Plataformas Populares de Repositórios Remotos:

- **Bitbucket**: Integrado ao Jira e com suporte ao Git e Mercurial, muito usado em equipes ágeis.



Quais?

Plataformas Populares de Repositórios Remotos:

- **Azure Repos:** Parte do Azure DevOps, facilitando a integração com outras ferramentas da Microsoft.



Azure Repos

Técnicas de versionamento

Git Flow

O Git Flow é uma estratégia de *branching* (ramificação) para o controle de versões no Git

Git Flow

- Estrutura de *Branches*:

- *master*: A *branch* principal. Contém o código que está em produção ou pronto para ser lançado.
- *develop*: A *branch* de desenvolvimento. Contém o código que está em andamento, geralmente a versão mais atual e estável, mas não necessariamente pronta para produção.
- *feature*: *Branches* para desenvolvimento de novas funcionalidades. Criadas a partir de *develop* e integradas de volta ao *develop* quando a funcionalidade está pronta.
- *release*: *Branches* criadas para preparar uma nova versão de produção. Criadas a partir de *develop*, e servem para testar, corrigir bugs e preparar o código para ser integrado ao *master*.
- *hotfix*: *Branches* criadas a partir de *master* para corrigir problemas críticos em produção. Após a correção, o código é integrado tanto ao *master* quanto ao *develop*.

Git Flow

- **Fluxo de Trabalho:**

- **Iniciar um novo desenvolvimento:** Começa com a branch *develop* atualizada.
- **Desenvolver novas funcionalidades:** Crie uma branch *feature* a partir de *develop*. Após concluir a funcionalidade, faça o *merge* de volta para *develop*.
- **Preparar uma nova versão:** Quando o *develop* tiver um conjunto de funcionalidades prontas, crie uma *branch release* para preparar o código para produção.
- **Corrigir bugs críticos:** Se houver problemas em produção, crie uma *branch hotfix* a partir de *master* para corrigir o erro rapidamente, e depois faça o *merge* de volta para *master* e *develop*.

Git Flow

- **Processo de Merge:**

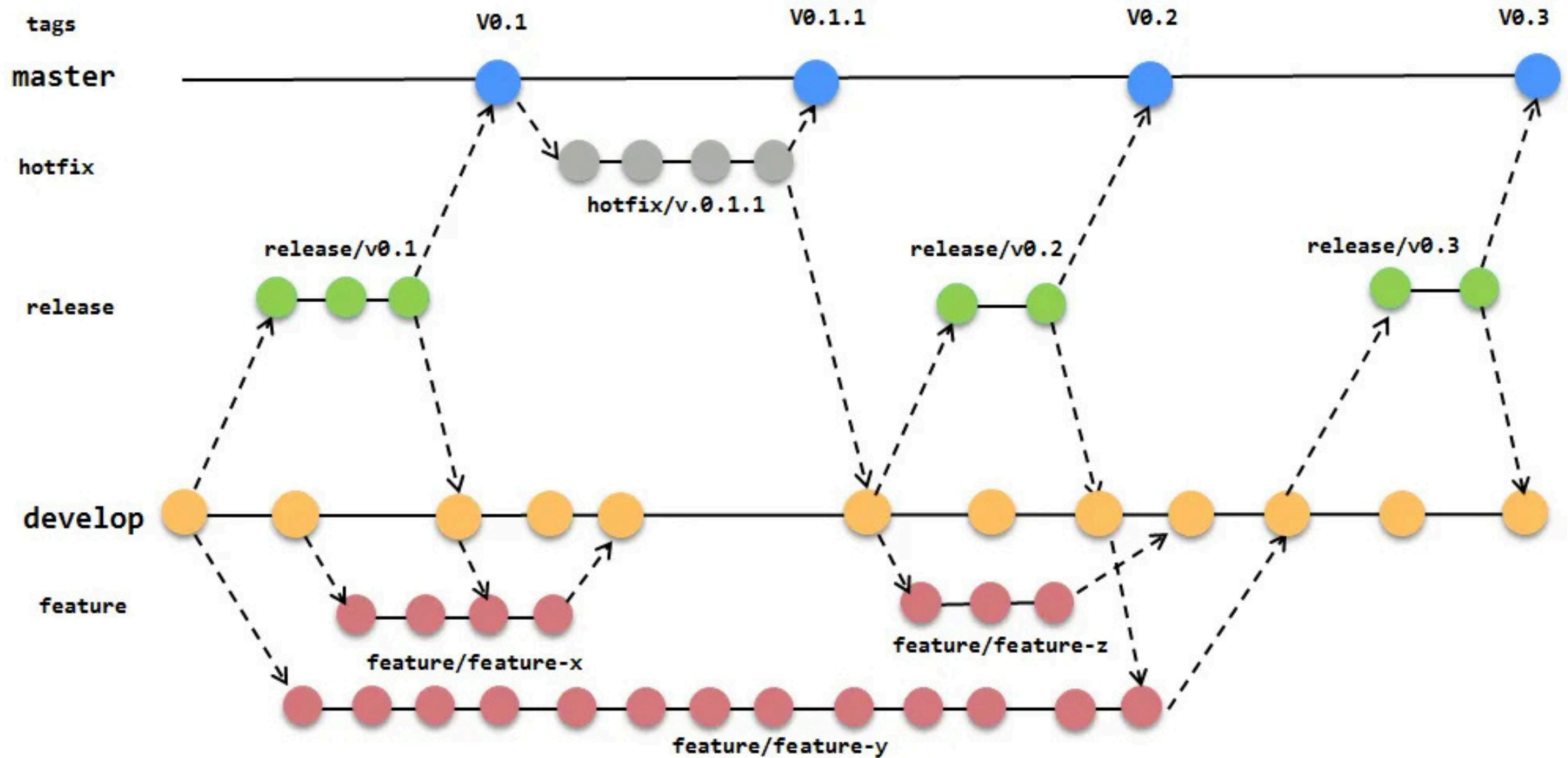
- **Merge entre branches:** Quando uma funcionalidade é concluída, faz-se o *merge* da *branch feature* de volta à *develop*.
- **Finalizando a versão:** Quando a *branch release* estiver pronta, faça o *merge* dela para *master* (onde o código estará em produção) e também de volta para *develop* para manter a sincronização.
- **Correções de emergência:** Para problemas críticos, a *branch hotfix* é mesclada tanto em *master* quanto em *develop*.

Git Flow

- **Tags e Versionamento:**

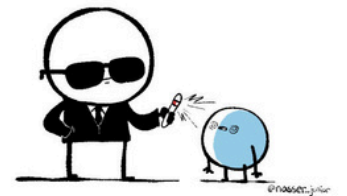
- *Tags no master:* Ao finalizar uma versão estável, uma *tag* é criada em master para marcar aquele ponto como uma versão de produção específica.

Git Flow



Entendendo o Git

`Git reset --hard`



Estrutura

- **Clone:** Criar uma cópia local de um repositório remoto, permitindo trabalhar offline e sincronizar mudanças depois.
- **Repositório Local:** O diretório onde o código é armazenado na máquina local.
- **Repositório Remoto:** O repositório hospedado em servidores, como GitHub, GitLab ou Bitbucket, que pode ser compartilhado com outras pessoas.
- **Staging:** O local onde as alterações são preparadas antes dos *commits*.

Estrutura

- *Commit*: Uma "foto" do estado atual do código. Cada *commit* é uma versão do projeto com um identificador único.
- *Pull*: Baixar alterações de um repositório remoto para o local.
- *Push*: Enviar as alterações locais para um repositório remoto.
- *Branch*: É uma ramificação (cópia) do código que permite desenvolver funcionalidades de forma isolada, sem afetar a versão principal do projeto.



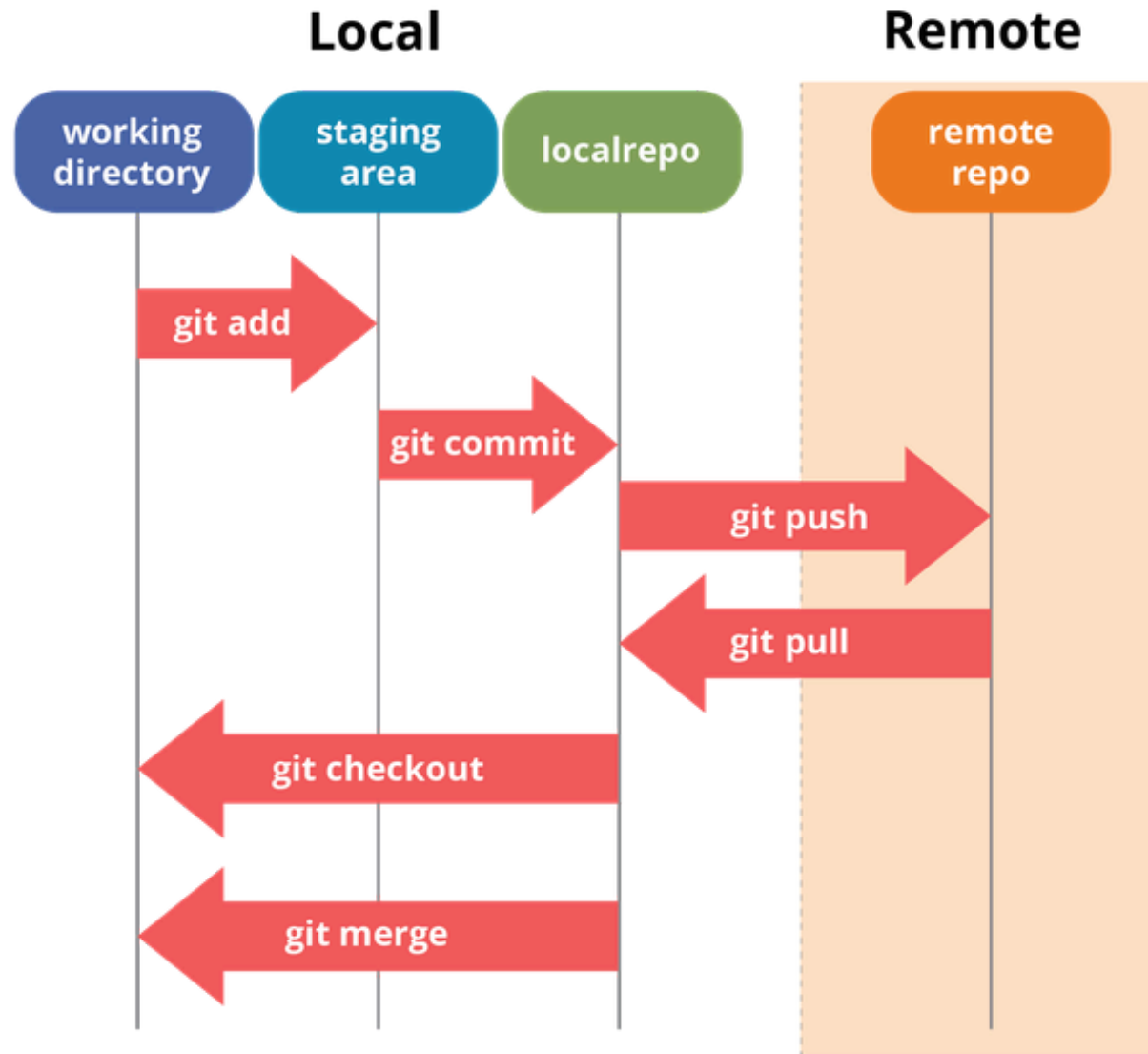
Conceitos

- **Merge**: Ação de combinar o conteúdo de duas branches diferentes, normalmente para integrar alterações de uma branch de desenvolvimento em uma branch principal (como develop para master).
- **Checkout**: Comando usado para alternar entre diferentes branches ou para reverter arquivos para um estado anterior.

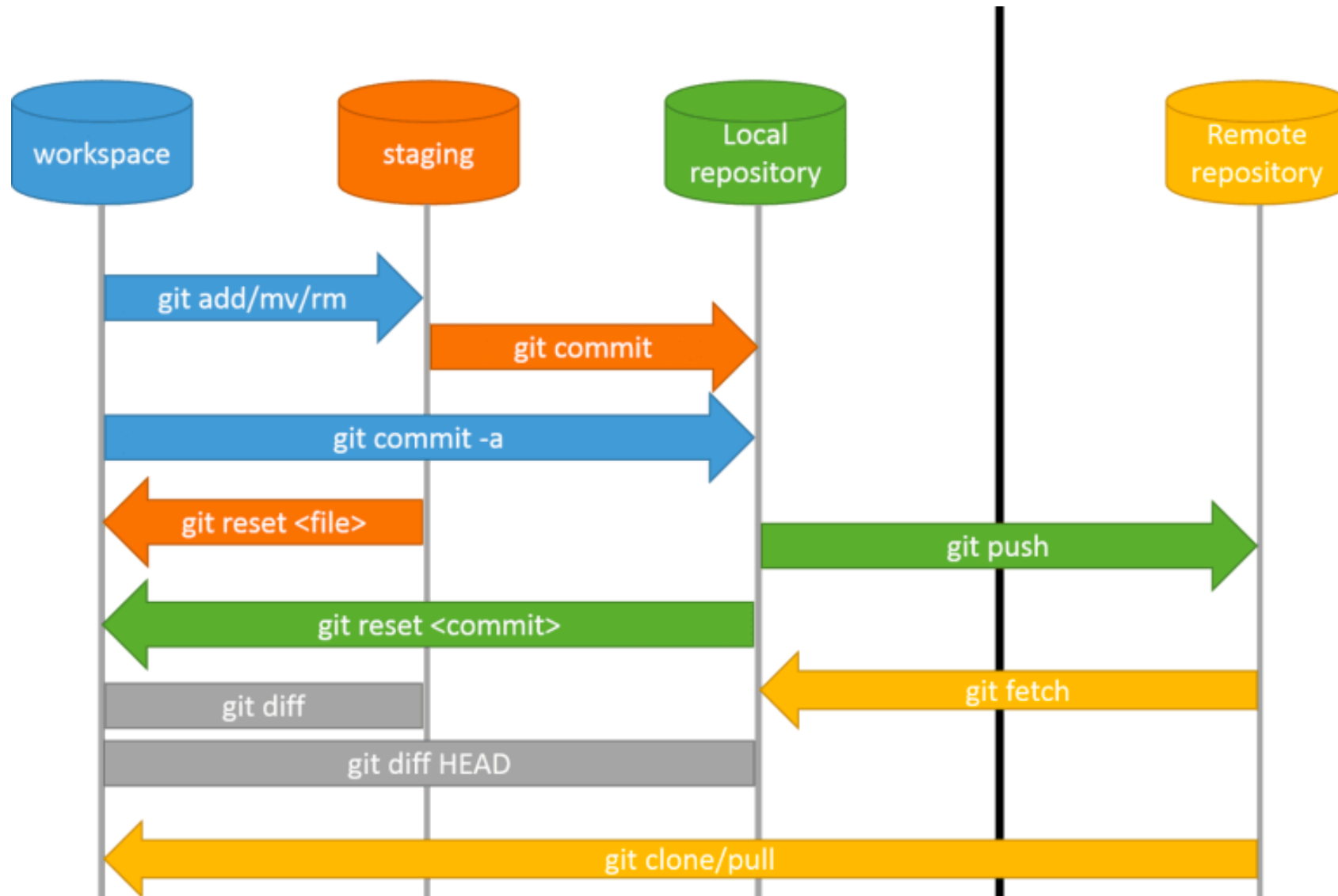
Conceitos

- *Tag*: Marcações especiais em commits específicos, usadas para indicar versões de release ou pontos importantes no histórico.
- *Fork*: Cópia de um repositório remoto, geralmente usada para contribuir em projetos *open-source*. Permite modificar sem afetar o repositório original.
- *.gitignore*: Arquivo usado para especificar quais arquivos ou pastas não devem ser rastreados pelo Git (ex: arquivos temporários, dependências).

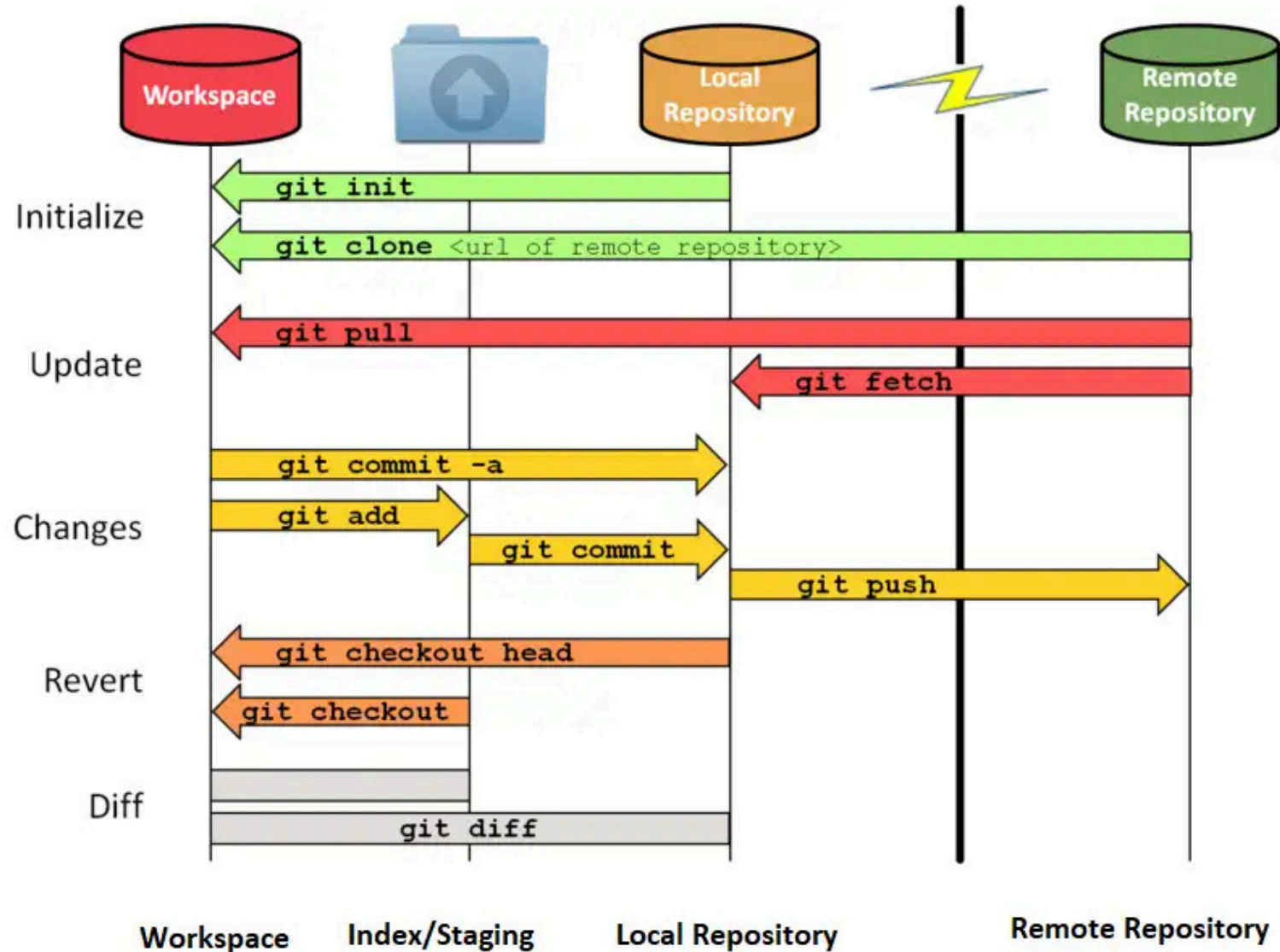
Conceitos



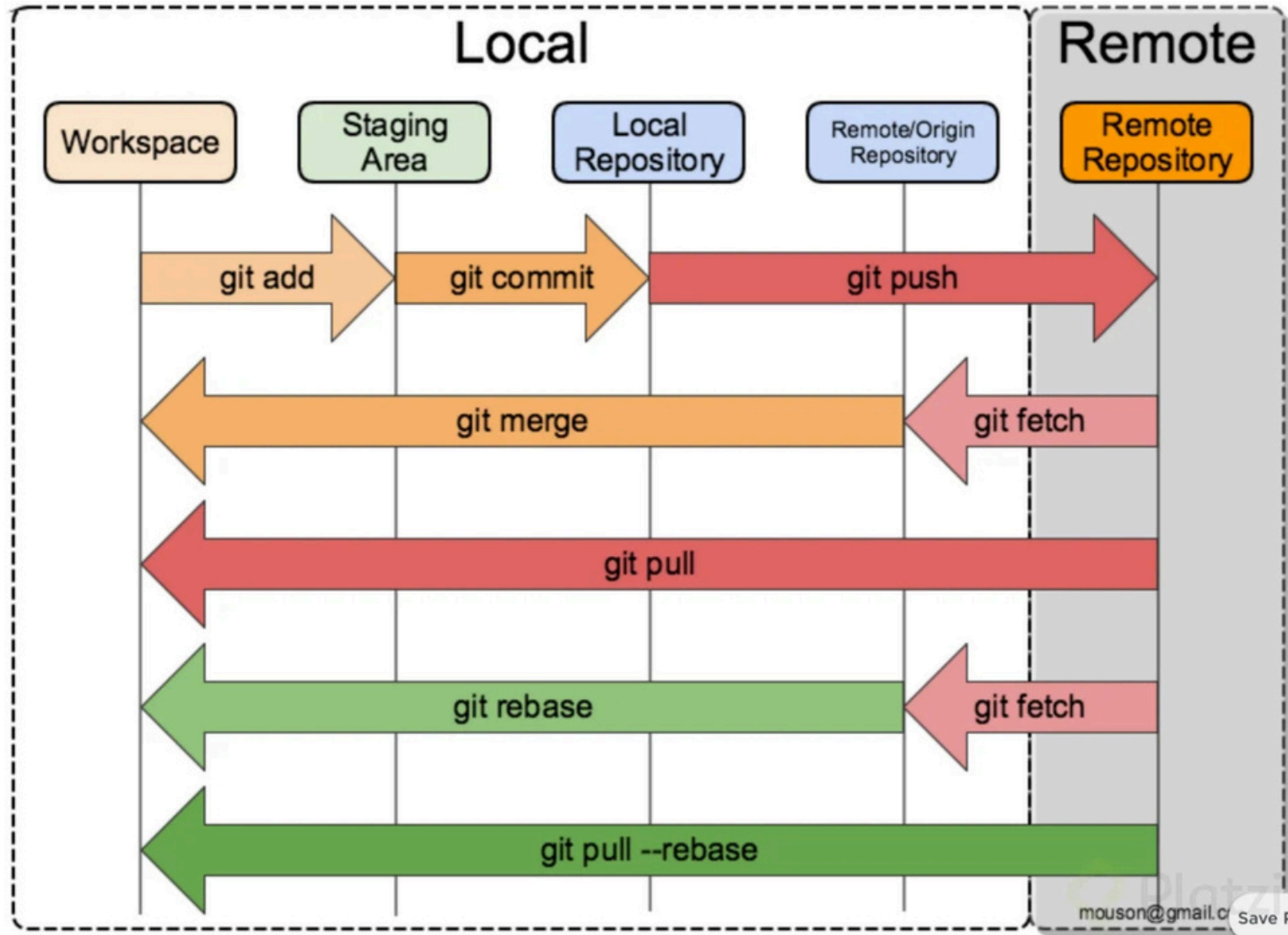
Conceitos



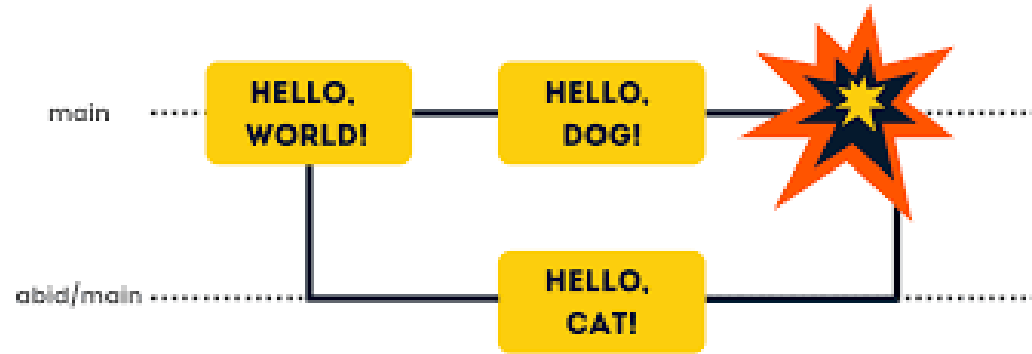
Conceitos



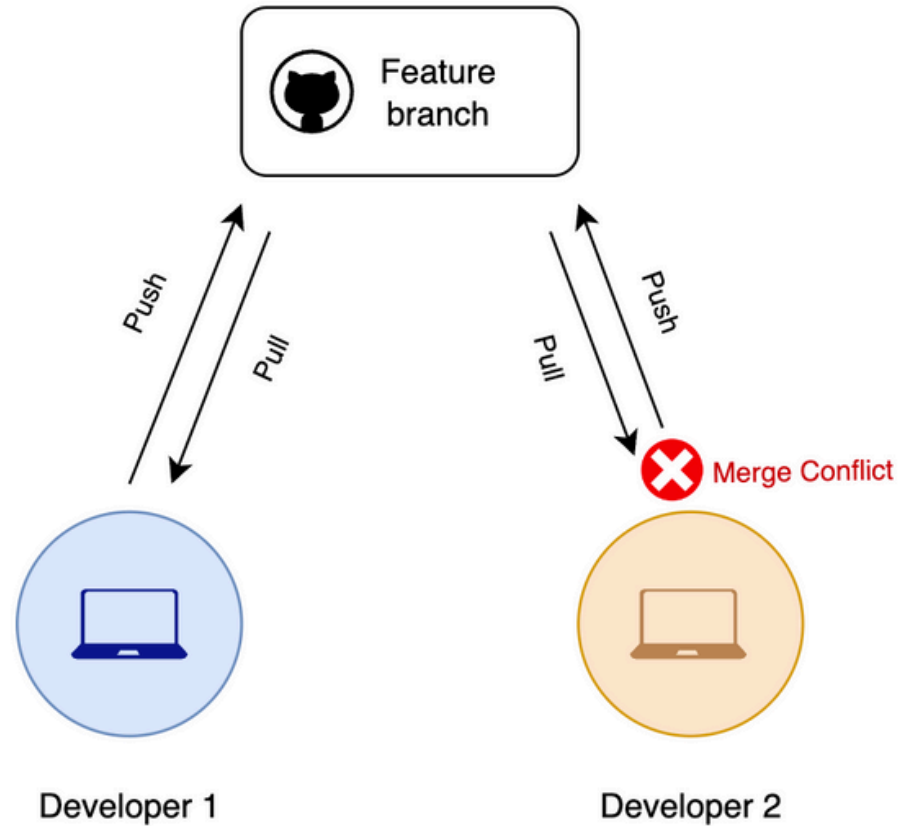
Conceitos



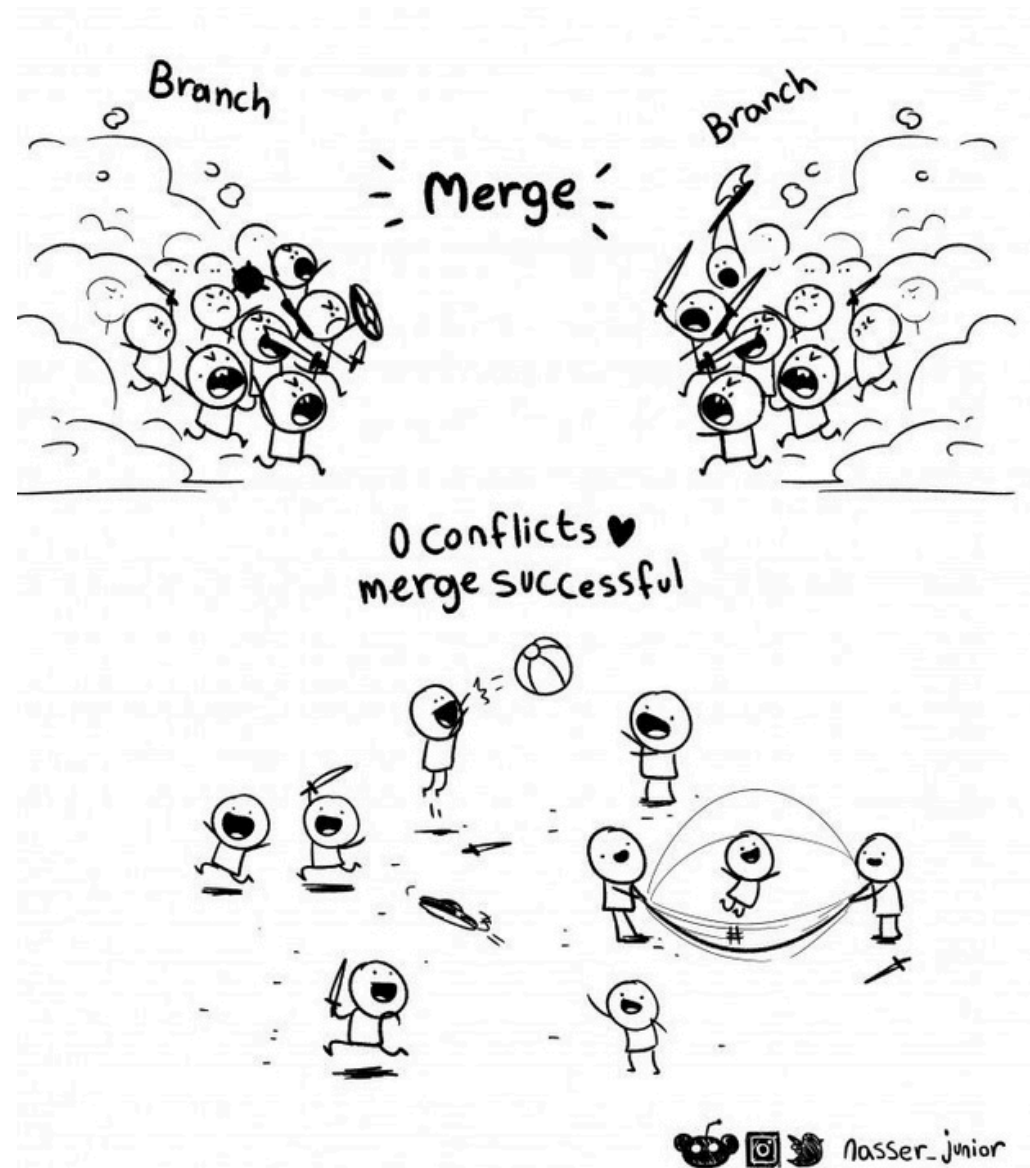
Merge e Conflitos



Merge e Conflitos



Merge e Conflitos



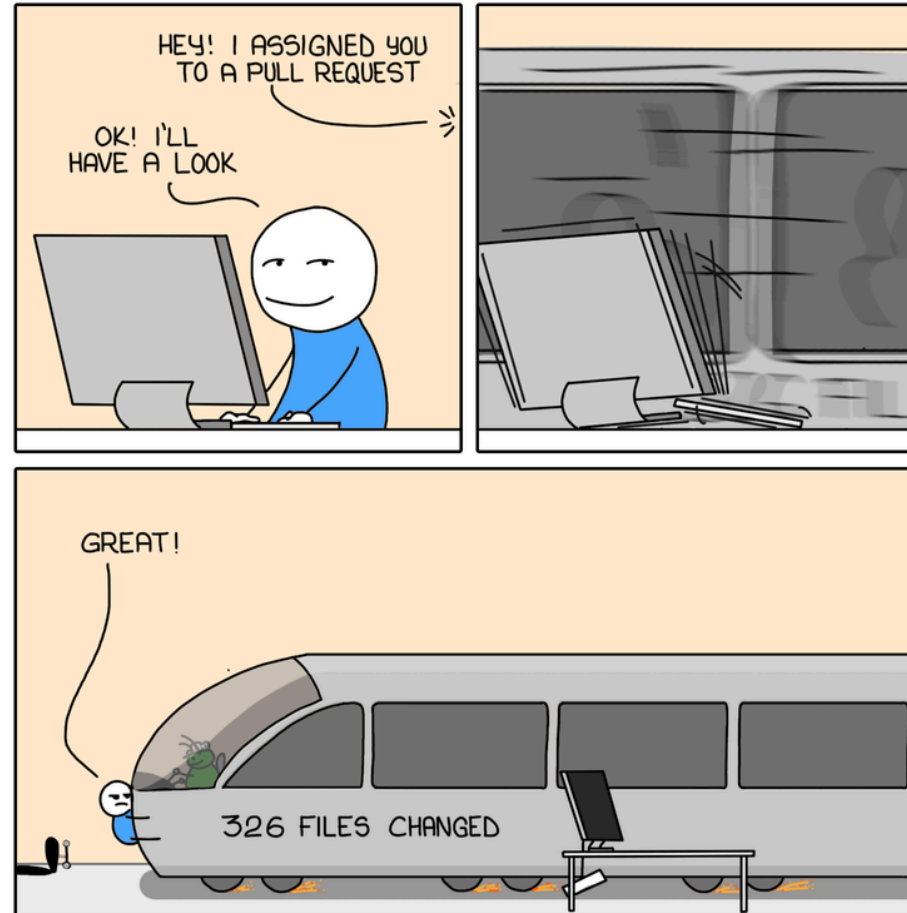
Pull Request

Um *pull request* é uma solicitação para revisar e integrar as alterações de uma *branch* (geralmente de um colaborador) na *branch* principal de um repositório, permitindo que as mudanças sejam avaliadas antes de serem mescladas.



Pull Request

PULL REQUEST



MONKEYUSER.COM

Prática

Principais comandos

- Como Configurar um repositório remoto:

```
$ git config --global user.name "John Doe"
```

```
$ git config --global user.email johndoe@example.com
```

Principais comandos

Comando	Descrição
<code>git init</code>	Inicializa um novo repositório Git em um diretório.
<code>git clone [URL]</code>	Cria uma cópia local de um repositório remoto.
<code>git add [arquivo]</code>	Adiciona mudanças específicas de um arquivo ao índice (staging area).
<code>git commit -m "mensagem"</code>	Salva as mudanças no repositório com uma mensagem descritiva.
<code>git status</code>	Exibe o estado atual do repositório, mostrando arquivos modificados e não rastreados.
<code>git pull</code>	Atualiza o repositório local com as mudanças do repositório remoto (combina fetch e merge).
<code>git push</code>	Envia as mudanças do repositório local para o repositório remoto.
<code>git branch</code>	Lista todas as branches no repositório local.
<code>git checkout [branch]</code>	Altera para uma branch específica.
<code>git merge [branch]</code>	Combina as mudanças de uma branch específica na branch atual.

Comandos Avançados

Comando	Descrição
<code>git rebase [branch]</code>	Move ou aplica commits da branch atual na base de outra branch, permitindo um histórico mais limpo.
<code>git cherry-pick [commit]</code>	Aplica as mudanças de um commit específico em outra branch.
<code>git stash</code>	Salva temporariamente mudanças não commitadas para uma área de armazenamento, permitindo trabalhar em outra tarefa.
<code>git stash pop</code>	Restaura as mudanças armazenadas no stash e as remove da pilha de stash.
<code>git log</code>	Exibe o histórico de commits, permitindo visualizar as alterações feitas ao longo do tempo.
<code>git diff</code>	Mostra as diferenças entre alterações não commitadas, entre commits ou entre branches.
<code>git remote -v</code>	Lista os repositórios remotos associados ao repositório local.
<code>git reset [commit]</code>	Reverte o repositório para um estado anterior, permitindo desfazer commits (usado com cuidado).
<code>git clean -fd</code>	Remove arquivos não rastreados e diretórios do diretório de trabalho.
<code>git tag [nome]</code>	Cria uma nova tag para marcar uma versão específica do projeto, geralmente usada em lançamentos.

Boas práticas

Boas práticas

- ***Branches e Merges:*** A importância de trabalhar com *branches* e práticas de *merge* frequente para evitar conflitos.
- ***Tags e Releases:*** Uso de *tags* para marcar versões importantes.
- **Documentação e Mensagens de *Commit*:** Importância de mensagens claras e de documentação de mudanças.

Referências

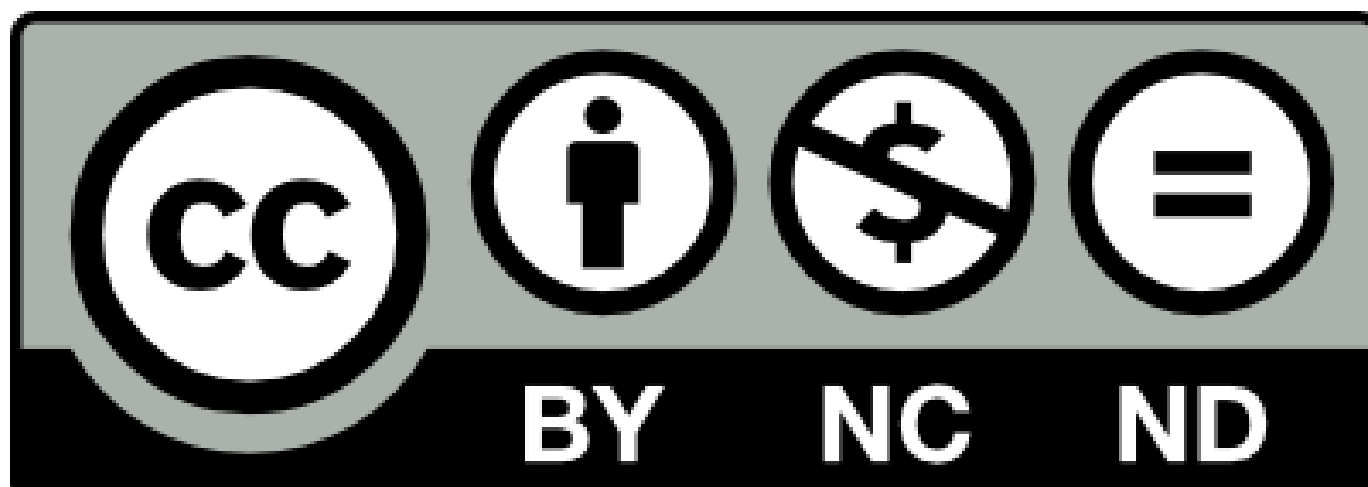
GIT SCM. Git: documentação oficial. Disponível em: <https://git-scm.com/doc>. Acesso em: 7 ago. 2025.

GITLAB. Sobre o GitLab. Disponível em: <https://about.gitlab.com/>. Acesso em: 7 ago. 2025.

GITHUB. GitHub. Disponível em: <https://github.com/>. Acesso em: 7 ago. 2025.

BITBUCKET. Bitbucket. Disponível em: <https://bitbucket.org/>. Acesso em: 7 ago. 2025.

**Estes *slides* possuem direitos autorais reservados por uma licença
Creative Commons:**



<https://creativecommons.org/licenses/by-nc-nd/4.0/legalcode>

<https://br.creativecommons.net/licencas/>

VERSIONAMENTO DE CÓDIGO FONTE

A central red diamond logo with the word "git" in white lowercase letters is positioned in the middle of the slide. Surrounding the logo is a complex network diagram consisting of numerous small colored circles (red, green, blue, yellow) connected by thin lines of the same colors, creating a web-like structure that extends across the background.

Marisangila Alves, MSc

marisangila.alves@edu.sc.senai.br