UNIVERSIDADE FEDERAL
DE MATO GROSSO

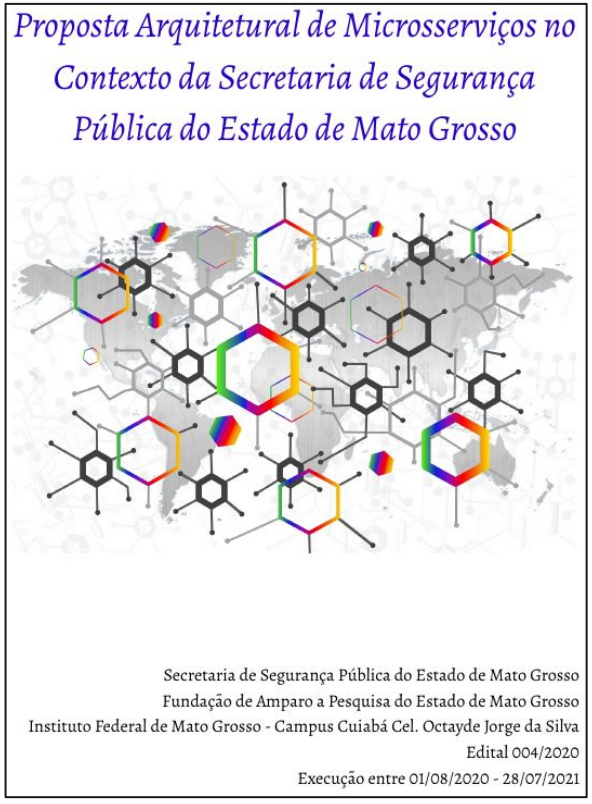
Pós-graduação em DevOps

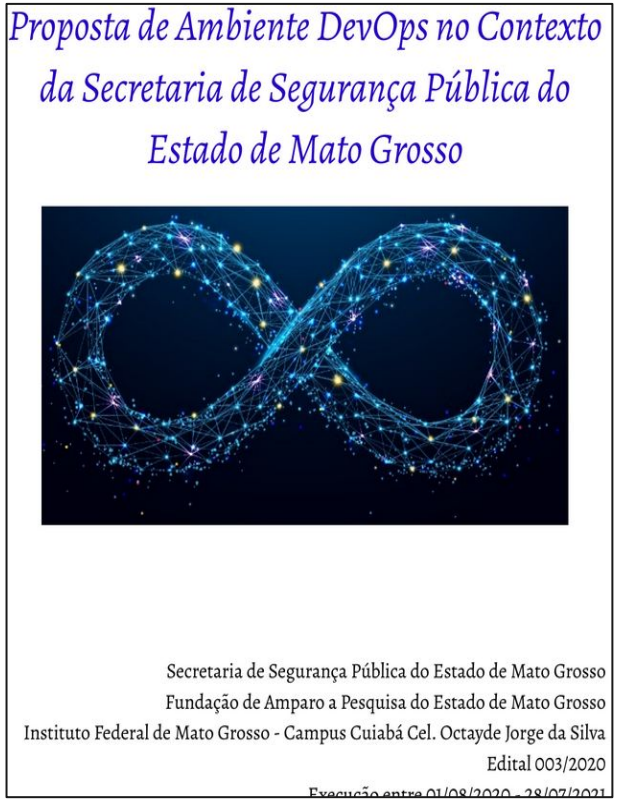
Desenvolvimento e Operações Integrados

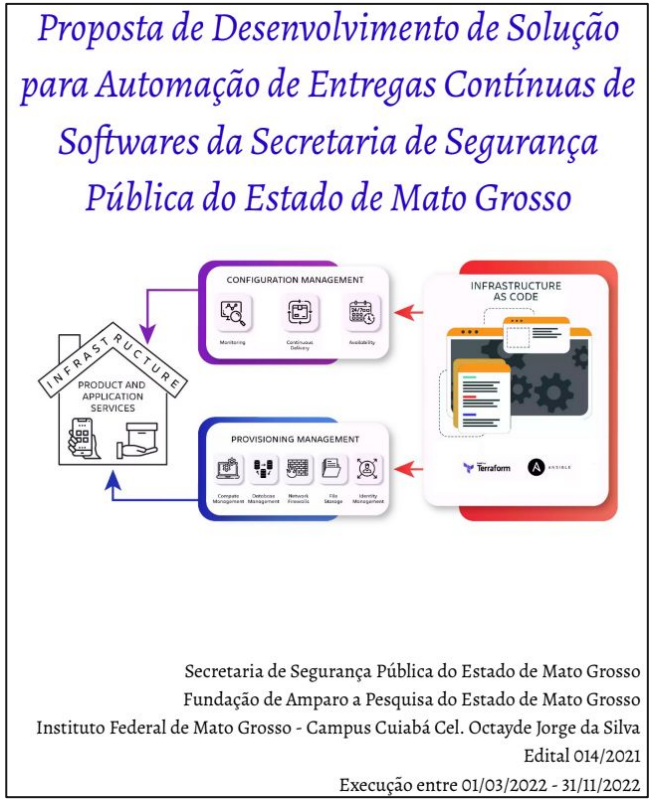
Gerenciamento de Código e Controle de Versão

João Paulo Delgado Preti
Professor Titular do IFMT
Departamento da Área de Computação (DCOM)
preti.joao@ifmt.edu.br

2024







ISBN: 978-65-00-70036-7

ISBN: 978-65-00-70037-4

ISBN: 978-65-00-70035-0

3rd International Conference on Electrical, Communication and Computer Engineering

12 – 13 JUNE, 2021, KUALA LUMPUR, MALAYSIA

Proc. of the 3rd International Conference on Electrical, Communication and Computer Engineering (ICECCE), 12-13 June 2021, Kuala Lumpur, Malaysia

Monolithic to Microservices Migration Strategy in Public Security Secretariat of Mato Grosso

João Paulo Delgado Pires
Software Engineering Applied Research Group (ESA)
Federal Education Institute of Mato Grosso (IFMT)
Cuiabá-MT, Brazil
joao.pires@cba.ifmt.edu.br

Adriano Neves Araújo Souza
Computer Networks Research Group (GPRS)
Federal Education Institute of Mato Grosso (IFMT)
Cuiabá-MT, Brazil
ah.desouza@gmail.com

Evandro Cleni Freiberger
Software Engineering Applied Research Group (ESA)
Federal Education Institute of Mato Grosso (IFMT)
Cuiabá-MT, Brazil
evandro.freiberger@cba.ifmt.edu.br

Tiago de Almeida Lacerda
Software Engineering Applied Research Group (ESA)
Federal Education Institute of Mato Grosso (IFMT)
Cuiabá-MT, Brazil
tiago.lacerda@cba.ifmt.edu.br

Public Safety Secretariat of Mato Grosso Microservice Environment

Abstract

This paper presents the microservice environment of the Public Safety Secretariat of Mato Grosso (SESP-MT) which was conceived to allow a migration process from SESP-MT monoliths and to absorb new organizational agile requirements. Despite the hype of microservice oriented architecture, it's an architectural style, with some general principles and as the nature of

used to understand the organization of SESP-MT systems and the MoSCoW requirements prioritization technique was used for sprint planning.

From the second sprint on there were monthly deliveries of executable products and a tested environment with a demo presentation followed by the next sprint planning.

The activities addressed the following challenges:

Nem Leigo
Nem Especialista

Um bom profissional busca aprender os **conceitos** por trás das suas ferramentas e ganhar experiência no seu **uso**.

FUNDAMENTAÇÃO

Sistema de Controle de Versão (VCS)

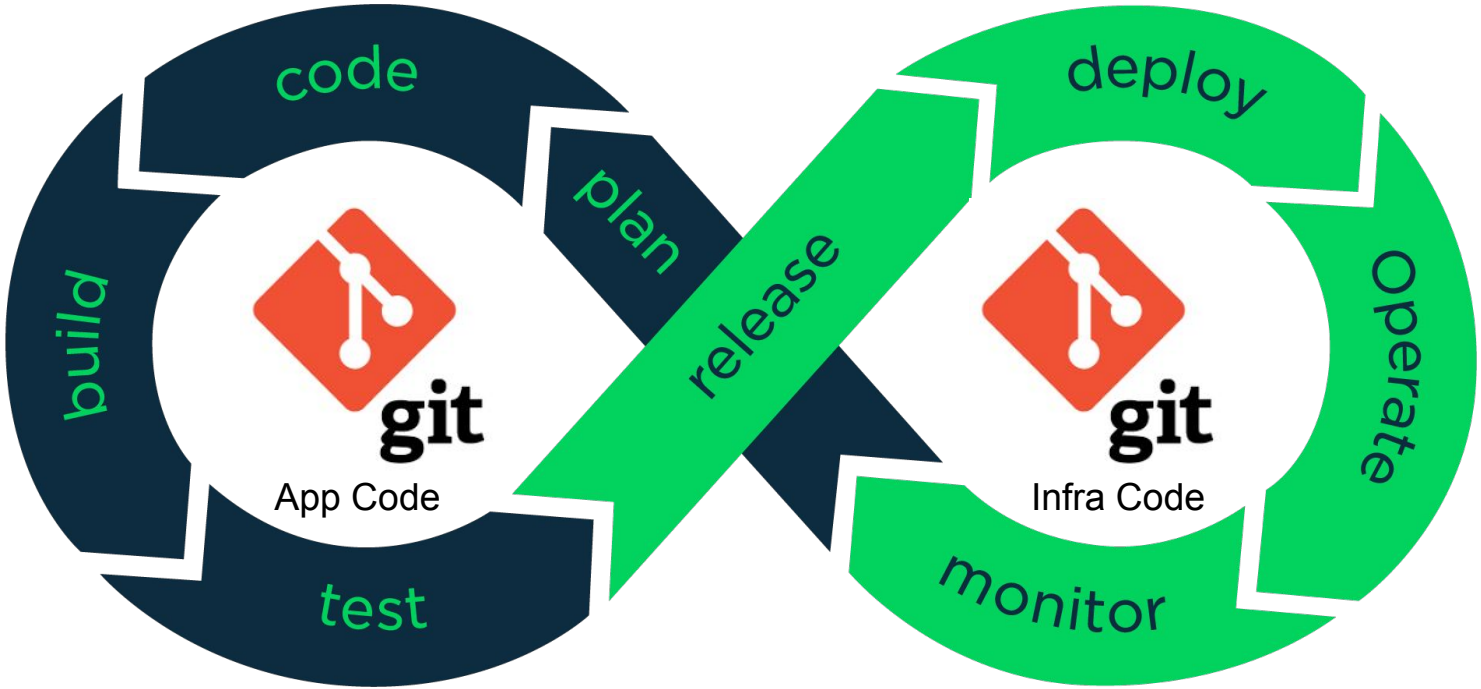
Um sistema que ajuda **pessoas** a trabalhar com código fonte de forma **organizada** e **simultânea**.

Permite:

- Controlar o Histórico
- Trabalhar em equipe
- Marcar e resgatar diferentes versões estáveis
- Criar ramificações do software

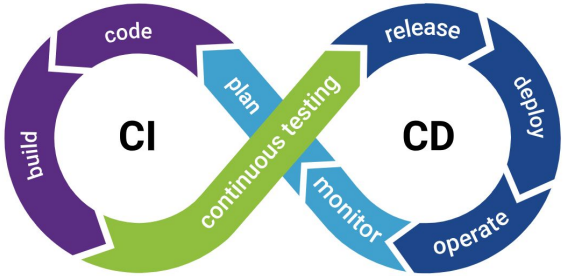
Fonte: Maior parte do conteúdo e das imagens da apresentação foram extraídas do livro localizado em https://git-scm.com/book/en/v2


git

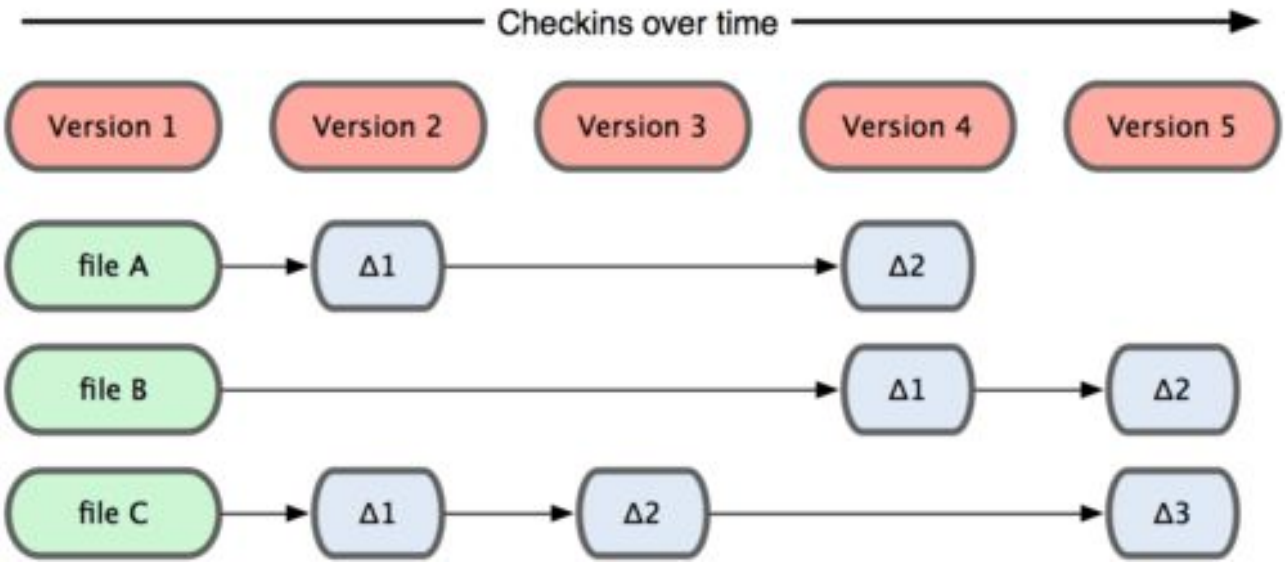

code
plan
build
test
release
deploy
Operate
monitor
git
App Code
git
Infra Code

gitops

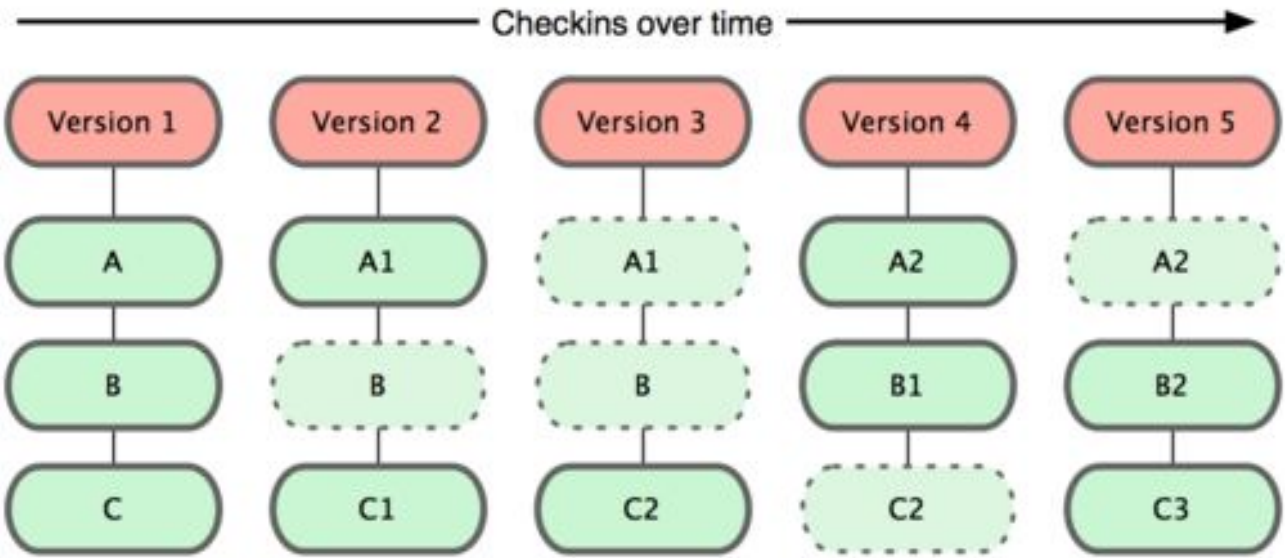

App code
Git
Build
Image Repo


code
plan
build
CI
continuous testing
release
deploy
monitor
operate
CD

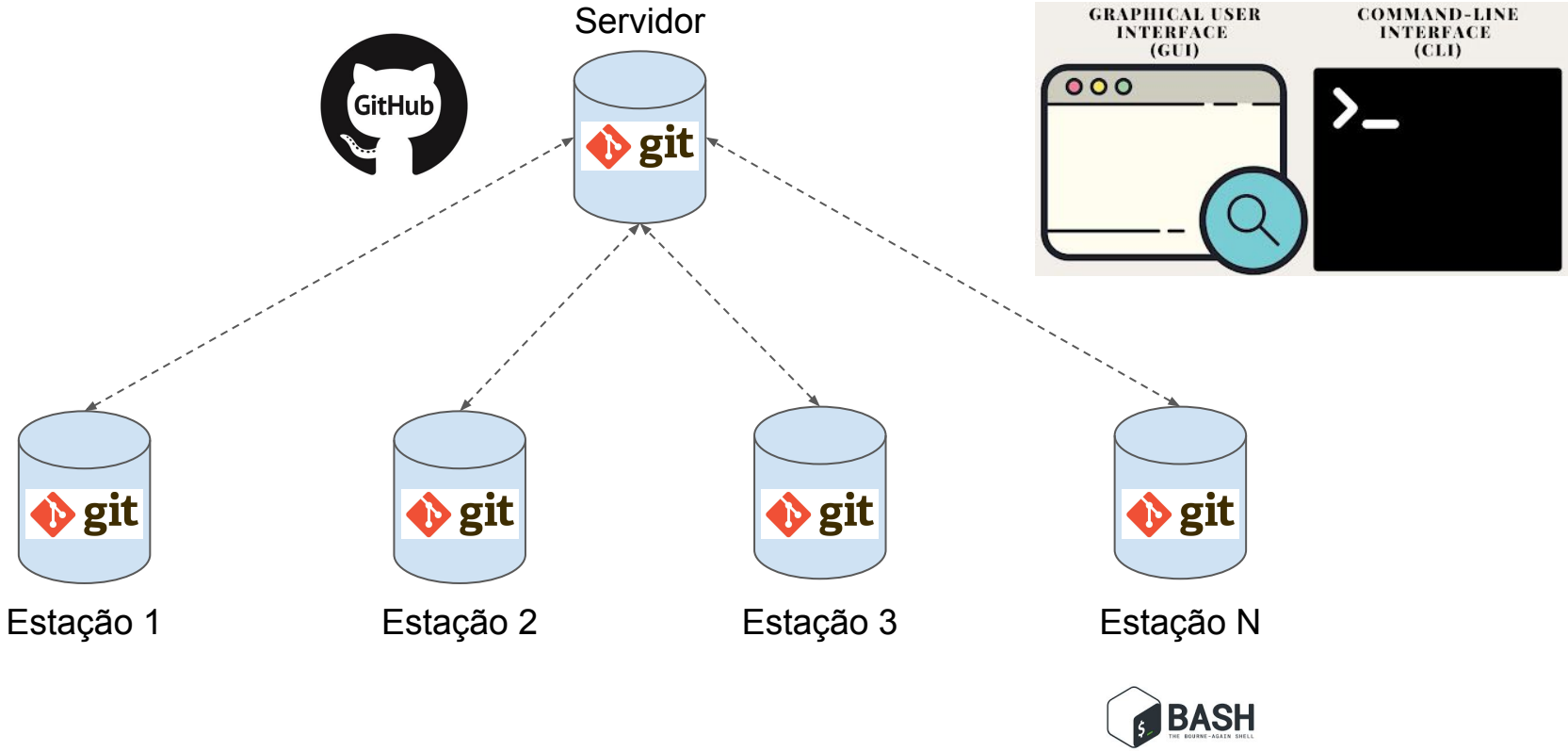
Snapshots e Não Diferenças



Snapshots


Checkins over time
Version 1
Version 2
Version 3
Version 4
Version 5
A
A1
A1
A2
A2
B
B
B
B1
B2
C
C1
C2
C2
C3

Criação do Repositório Remoto e Local



.gitignore

Arquivo que descreve os arquivos que não serão visíveis para o git.

Exemplo:

```
h2.db              # arquivo específico
*.html             # arquivos de extensão html
!index.html        # exceção, esse arquivo será visível
log/               # diretório específico
**/tmp             # qualquer diretório nomeado de tmp
```

Arquivos que já estavam sendo rastreados não são afetados

Configurações Básicas

`--local` configuração aplicada a um repositório local específico
`--global` configuração aplicada a todos os repositórios do usuário
`--system` configuração aplicada a todos os usuários do sistema

```
git config --global user.name <username>

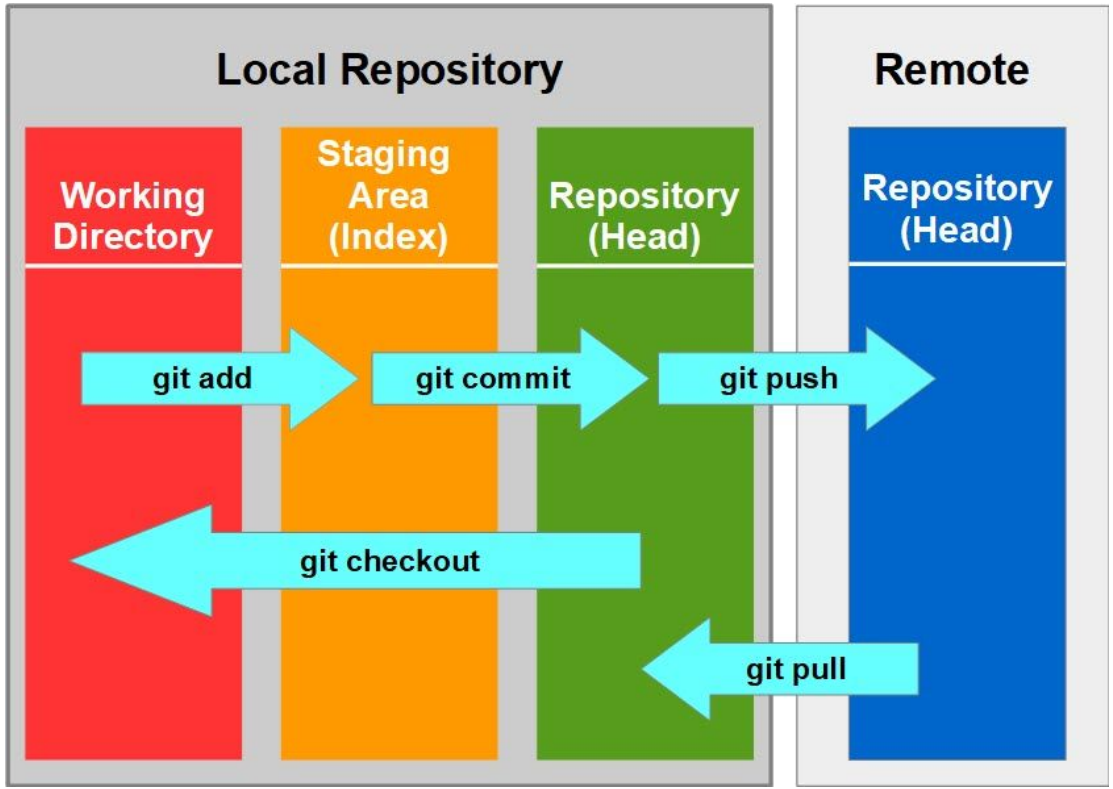
git config --global user.email <email>

git config --global core.editor <editor>

git config --global pull.rebase false

git config --list --show-origin
```

Principais Operações



untracked

não rastreado

Primeiros Comandos

```
git clone ENDEREÇO          # Cria um repositório local fazendo cópia de um remoto

git status                  # Apresenta um status sobre a sua branch e os arquivos

git log                     # Apresenta todos os commits realizados, por quem e a data
git log --stat              # Apresenta a mais o que foi modificado no repositório em cada commit
git log --oneline           # Apresenta de forma resumida em uma única linha cada commit
git shortlog                # Apresenta os commits realizados agrupados por usuário

git diff                    # Apresenta as diferenças entre o conteúdo dos arquivos no repositório

git blame arquivo           # Apresenta quem foi responsável por cada modificação no arquivo

git remote                  # Apresenta quais são os repositórios remotos
git remote -v               # Apresenta quais são os repositórios remotos e onde se encontram (URL)
```

Adicionando e Atualizando os Repositórios

```
git pull                          # Atualiza o repositório local com que há de novo no remoto

git add ARQUIVO_DIRETORIO         # Adiciona o arquivo a área de stage (seleção)

# Confirma as alterações, gerando uma nova versão no repositório local
git commit -m "DESCRIÇÃO DO QUE FOI REALIZADO"
git commit -am "DESCRIÇÃO DO QUE FOI REALIZADO"
git commit --amend -m "DESCRIÇÃO DO QUE FOI REALIZADO"

git push                          # Envia a nova versão para o repositório remoto
```

Versionamento Semântico (SemVer)

v[major].[minor].[patch]

[patch]: correção de bugs / otimização **v0.0.1**

[minor]: novas funcionalidades compatíveis com versões anteriores **v0.1.0**

[major]: novas funcionalidades incompatíveis com versões anteriores **v1.0.0**

Versões de teste: alpha (a), beta (b)

Exemplo: **v0.1.9 < v0.1.10 < v0.2.0a < v0.2.0b < v0.2.0**

Tags

```
git tag -a v0.0.1 -m "..."          # Cria uma tag v0.0.1

git tag                             # Apresenta todas as tags

git tag -l v0.*                     # Lista apenas as tags que começam com v0.

git show v0.0.1                     # Exibe qual é o commit da tag v0.0.1 e quem é o autor

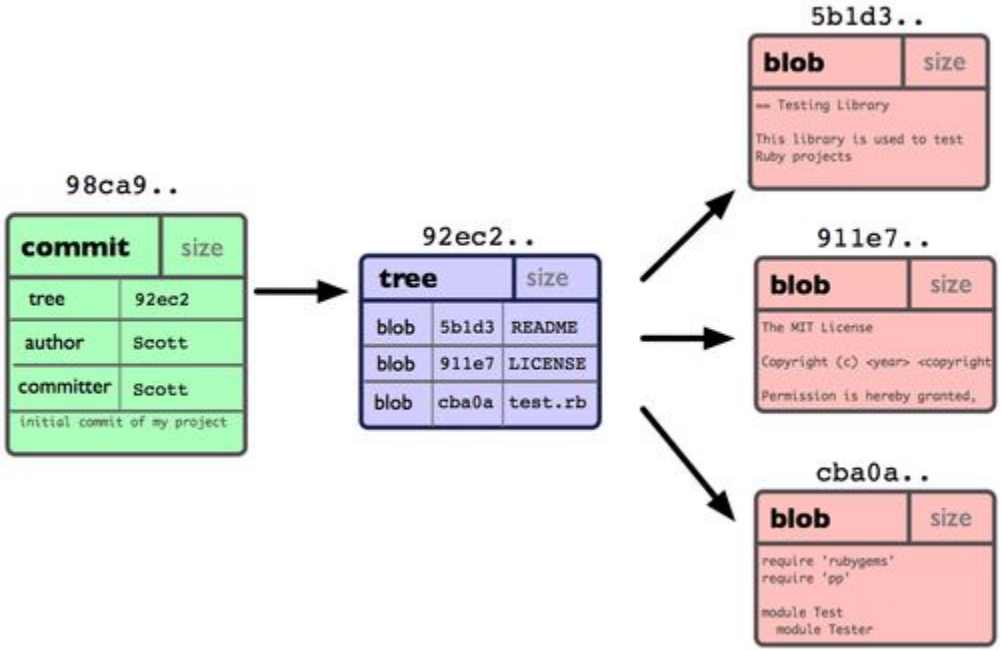
git push origin v0.0.1              # Envia a tag para o repositório remoto

# direciona o desenvolvedor para uma nova branch a partir da tag v0.1
git checkout tags/v0.0.1 -b task112
```

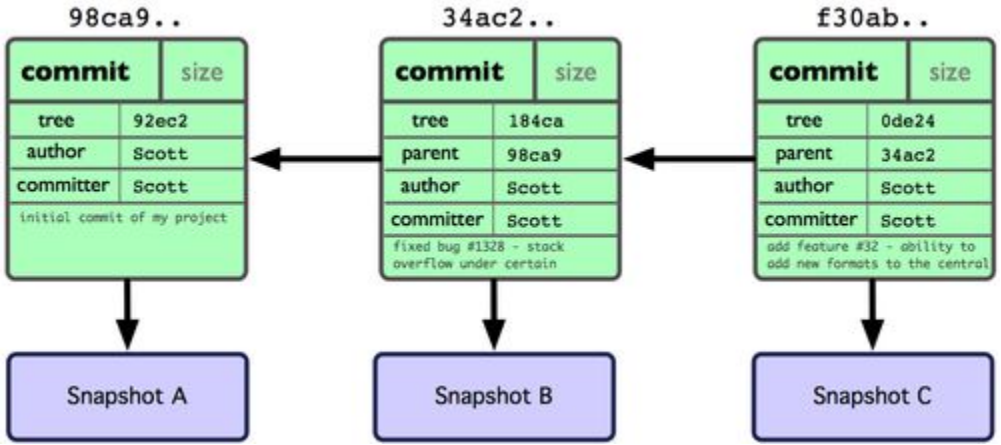
Branches

Ramos

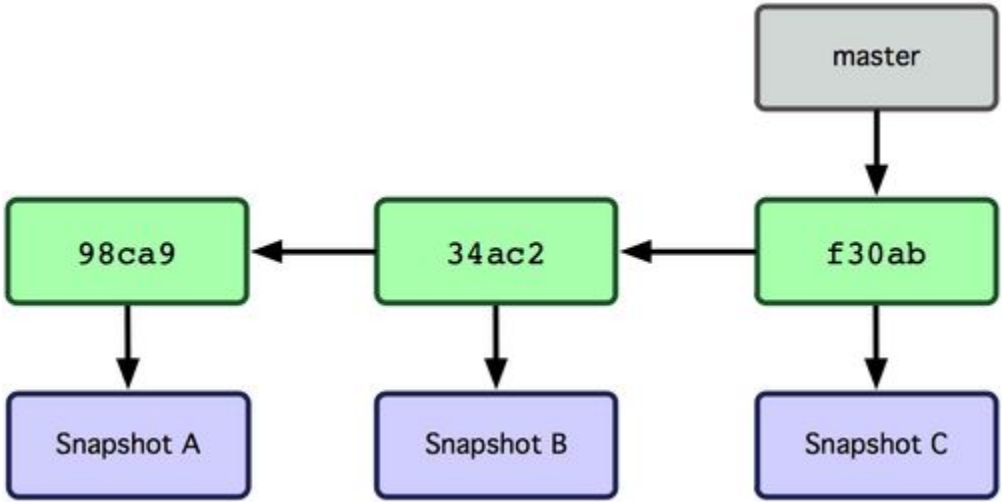
Único Commit



Múltiplos Commits



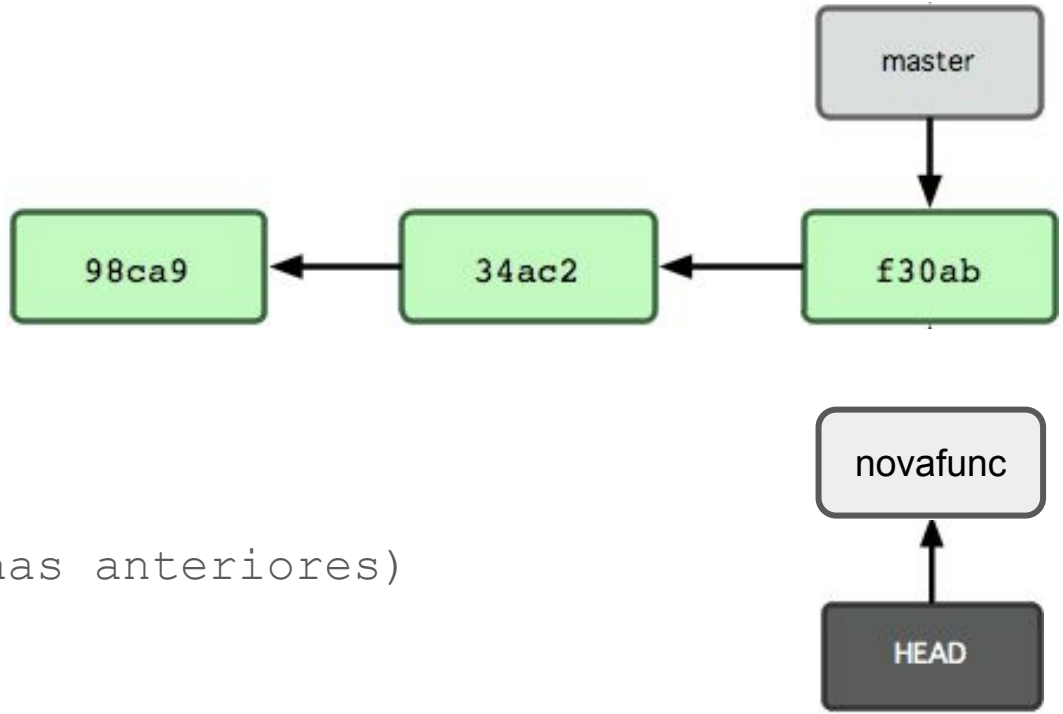
Branch master / main



Criando Novo Branch

```
git branch novafunc

git checkout novafunc
```

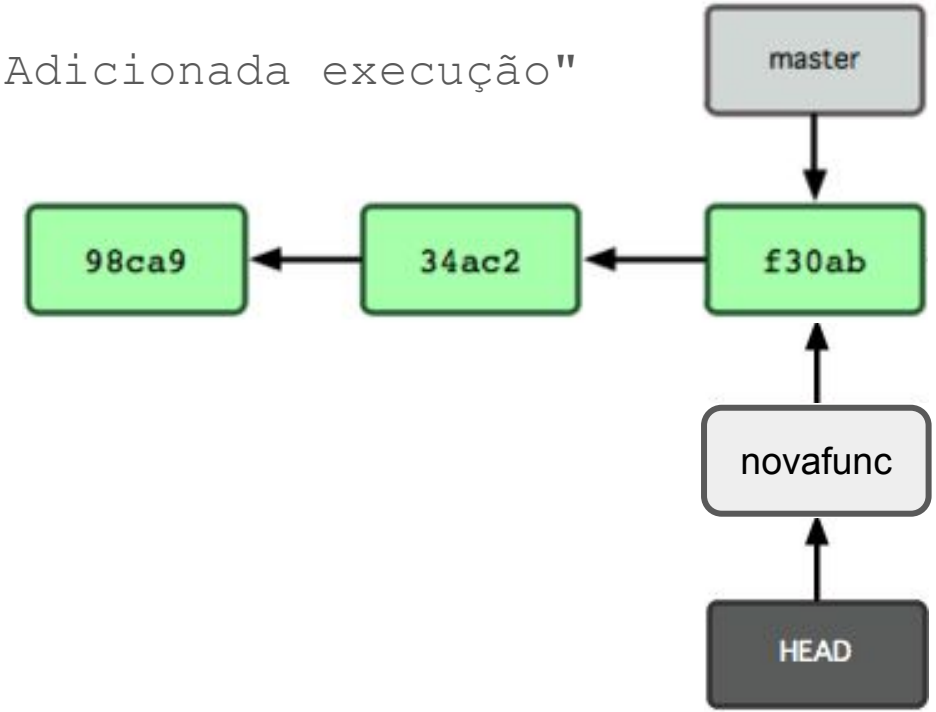



```
OU (equivalente as 2 linhas anteriores)

git checkout -b novafunc
```

Avançando no Novo Branch

```
vi README.md
git commit -am "Adicionada execução"
```



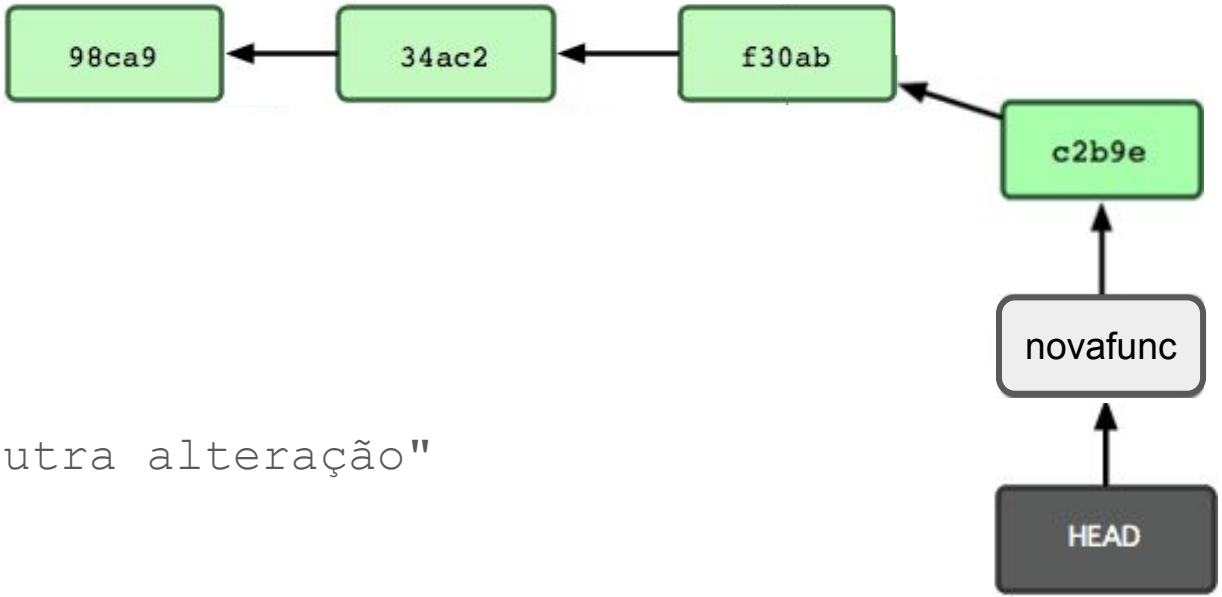
Avançando no Branch master

```
git checkout master
```

```
vi teste.py
```

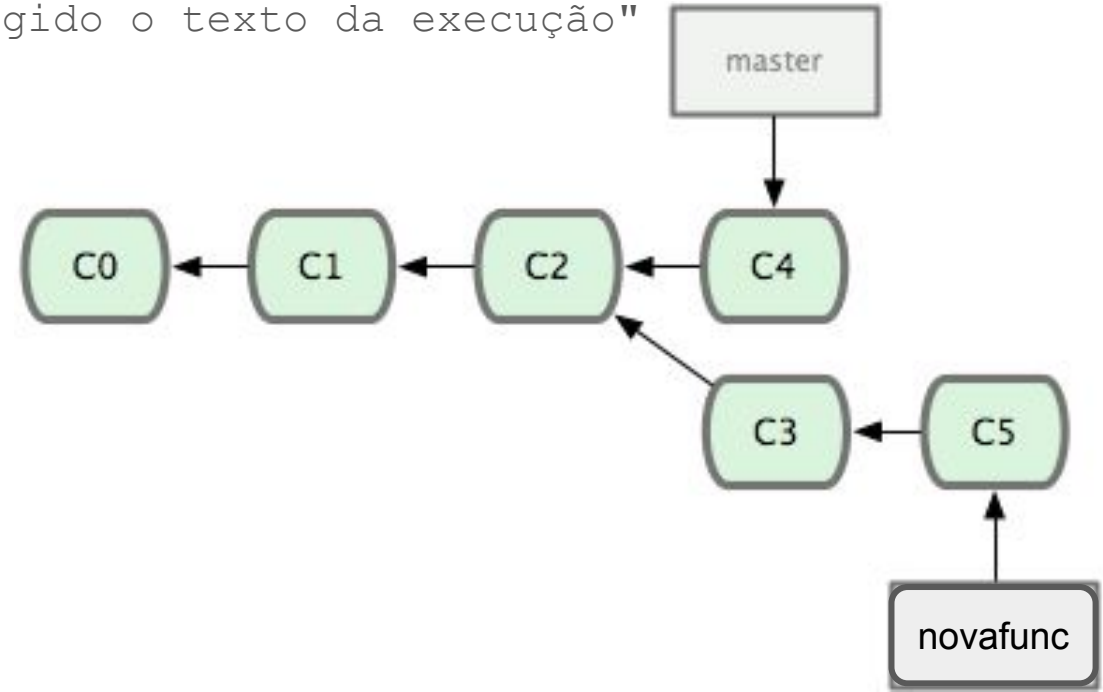
```
git add teste.py
```

```
git commit -m "Fiz outra alteração"
```



Retornando ao Branch novafunc

```
git checkout novafunc
vi README.md
git commit --amend -am "Corrigido o texto da execução"
```

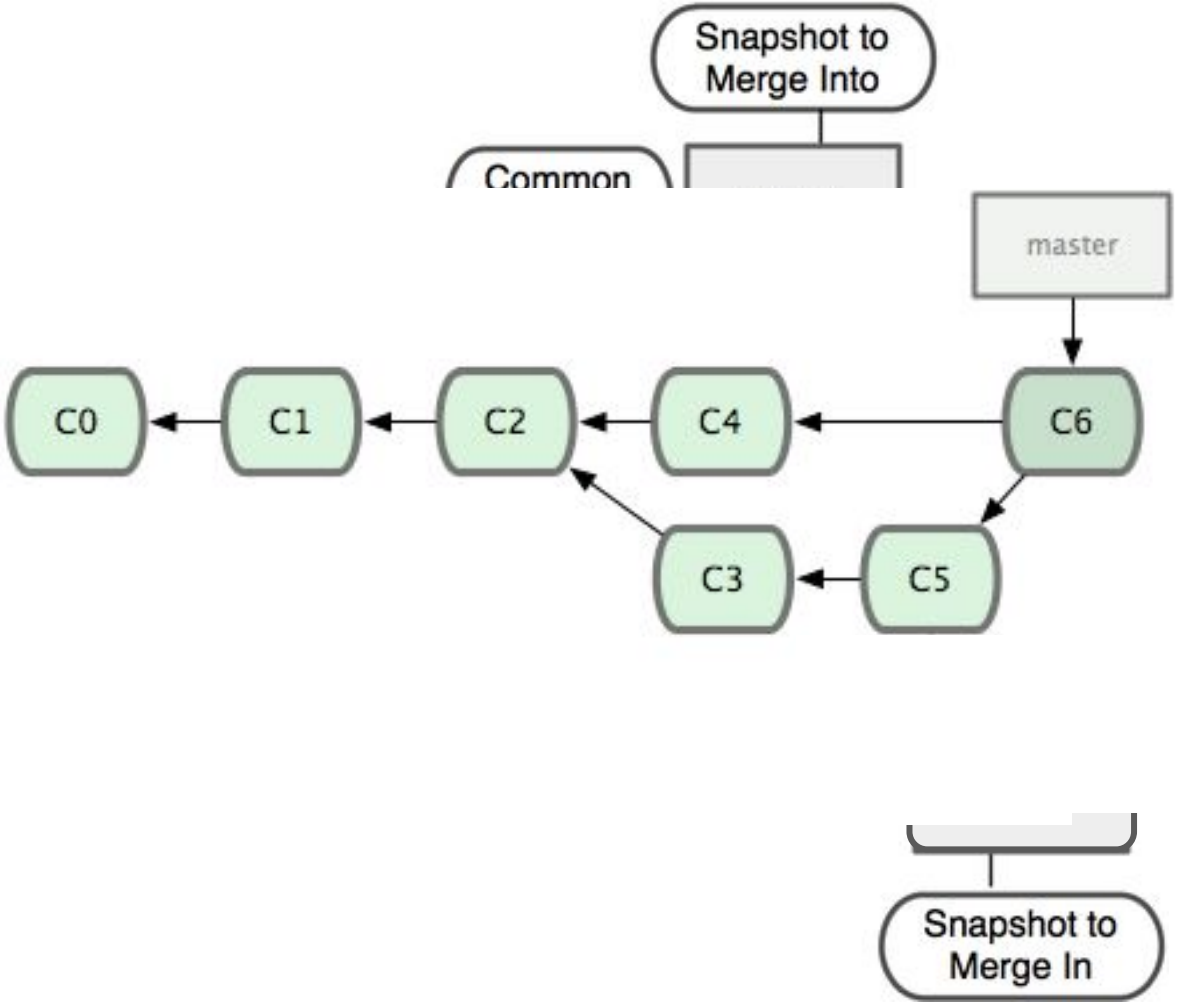


Fazendo o Merge

```
git checkout master

git merge novafunc

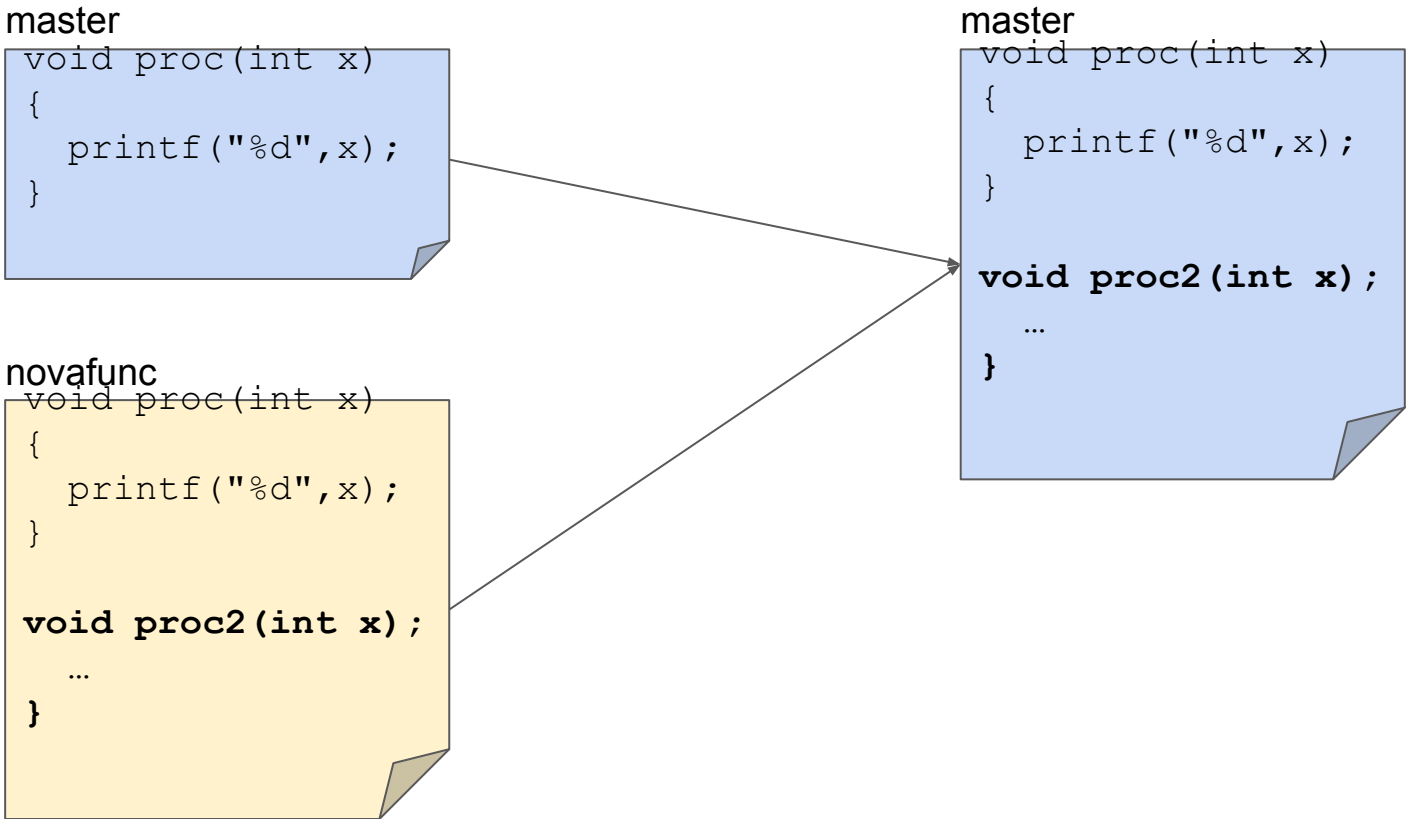
git branch -d novafunc
```



Merge - Mesclando Commits

```
git checkout master

git merge novafunc
```



Merge - Mesclando Commits

```
git checkout master
git merge novafunc
```

master

```
void proc(int x) {
  printf("%d",x+x);
}
```

novafunc

```
void proc(int x) {
  printf("%d",x);
}

void proc2(int x);
  …
}
```

master

```
void proc(int x) {
  printf("%d",x+x);
}

void proc2(int x);
  …
}
```

Merge - Mesclando Commits

```
git checkout master

git merge novafunc
```

master

```
void proc(int x) {
  printf("%d",x+x);
}
```

novafunc

```
void proc(int y) {
  printf("%d",y+y);
}
```

master

```
<<<<<<< HEAD
void proc(int x) {

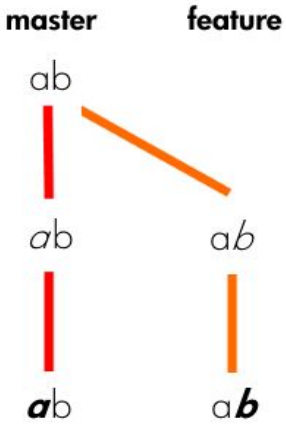
printf("%d",x+x);
}
=======
void proc(int y) {

printf("%d",y+y);
}
>>>>>>> novafunc
```

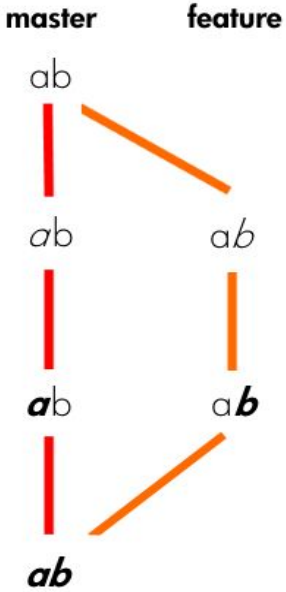

```
git add ARQUIVO.c

git commit -m "..."
```

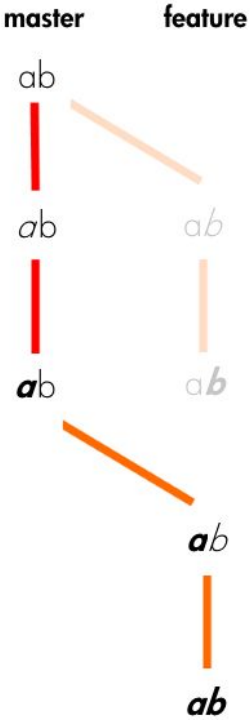
Merge vs Rebase



MERGE



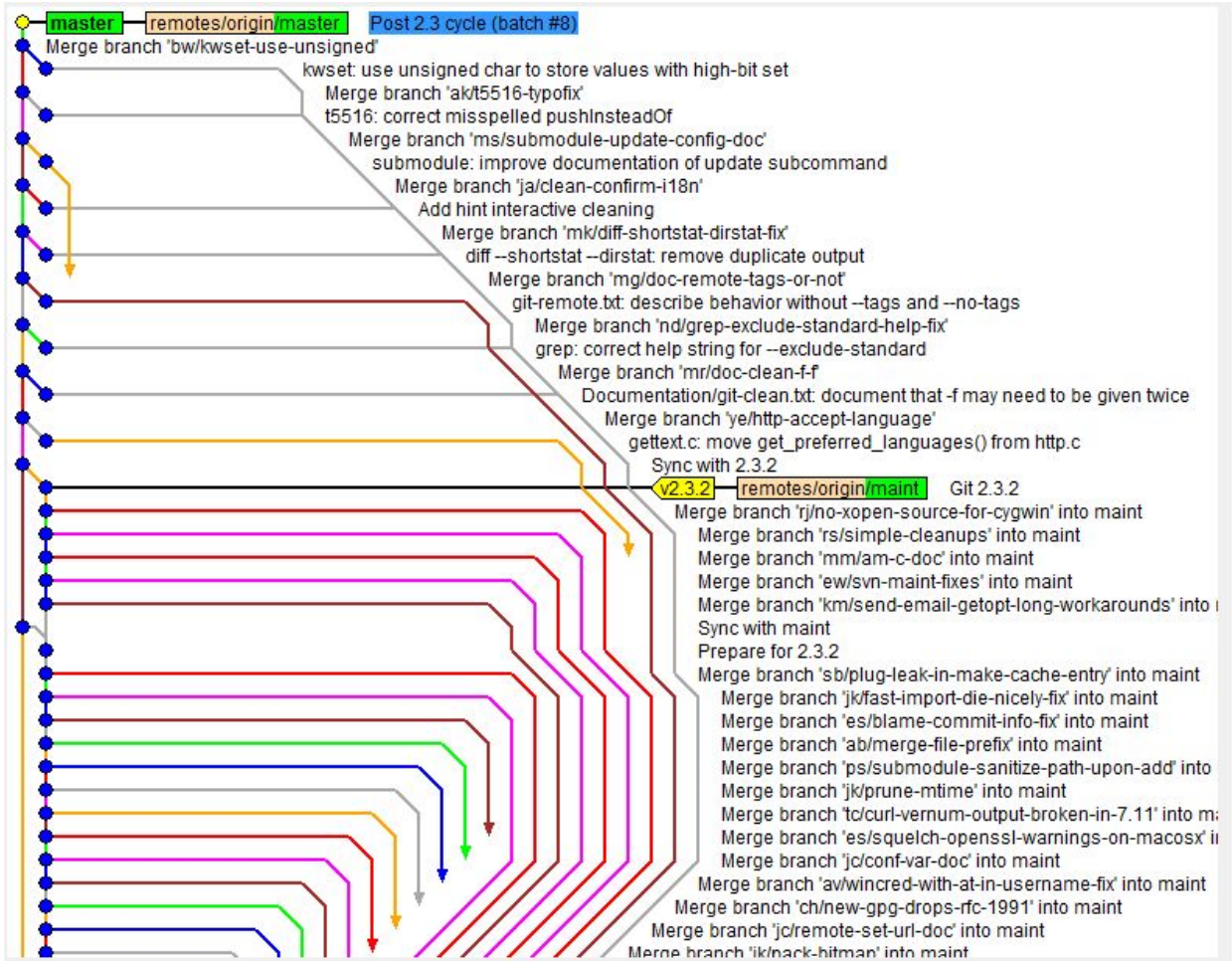
REBASE



https://hackernoon.com/git-merge-vs-rebase-whats-the-diff-76413c117333

Rebase muda o passado!!!

Merge vs Rebase



(Des/Re)fazendo Ações

```
git checkout ARQUIVO                # Recupera o arquivo do último commit

git checkout COMMIT ARQUIVO         # Recupera o arquivo do commit COMMIT

git checkout COMMIT                 # Recupera os arquivos do commit COMMIT

# Atenção que no reset hard pode haver perda de trabalho realizado
git reset --hard COMMIT             # Altera o diretório de trabalho e o stage para o commit COMMIT

# Cria um commit novo que apaga as linhas introduzidas/introduza as linhas apagadas até o commit COMMIT
git revert COMMIT

git clean [-f]                      # Exclui arquivos não rastreados. Pode forçar a exclusão

git clean -n                        # Exibe arquivos não rastreados que serão excluídos

git diff > patch.txt                # Salva o que foi modificado entre o diretório e o último commit
git apply patch.txt                 # Aplica no diretório de trabalho as modificações do patch
```

Alias

```
git config --global alias.lol "log --graph --decorate --oneline"
```

```
git lol
```

```
| |\
| |/
|/|
* |   ec55003 Merge branch 'Crudusuario'
|\ \
| |/
| * 59fa63d crud usuario back-end
* | 3206a2e Melhorias crud departamento
|/
* 41a47e6 Melhoria modelagem das classes
* 237ed2a Finalização crud departamento
* c8ae8a9 impl. crud departamento
* 3765223 corr. pom
* 8c6b9f8 Implementado login
*   db75180 Merge remote-tracking branch 'origin/master'
|\
| * cc2c0fd README.md edited online with Bitbucket
* | dedbfdb Implementado segurança backend
|/
* 1c46405 Update para java 11
* 3c8cee6 Add banner de inicialização personalizado
* 29aa9b6 add. fonte
* 7941286 Initial commit
```

Sites

Para teste de comandos:
https://git-school.github.io/visualizing-git/

Com exercícios:
https://learngitbranching.js.org/?locale=pt_BR

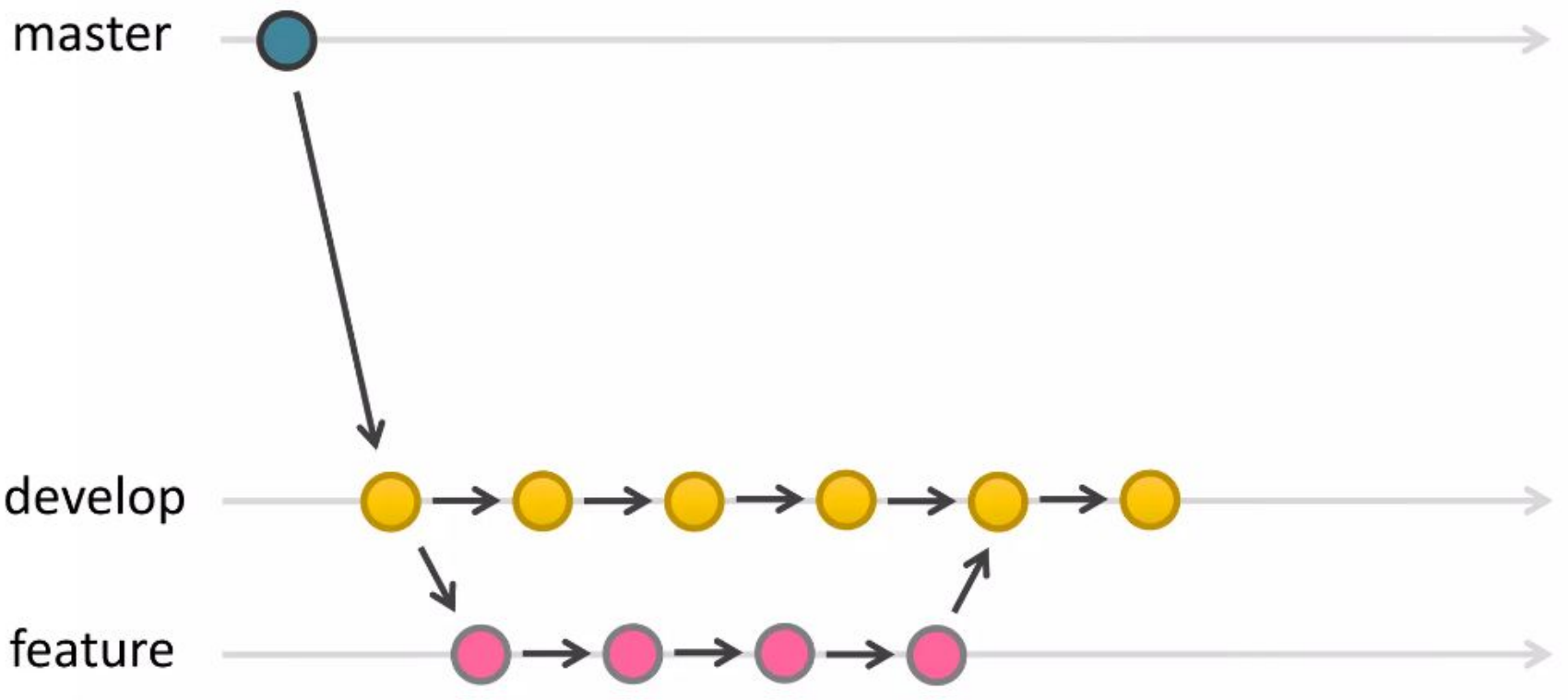
Fluxo de Trabalho

Git Flow

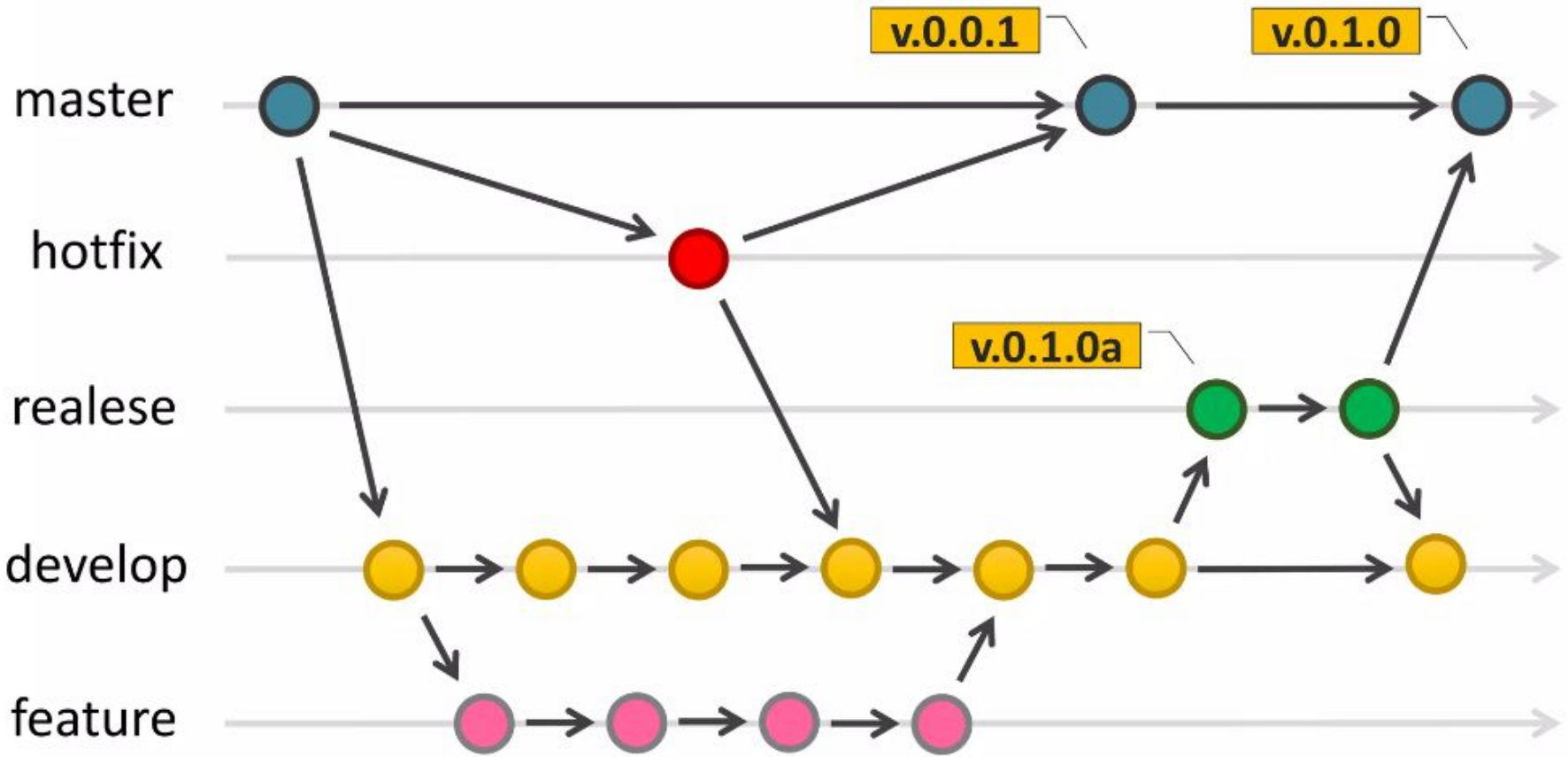
Fluxo de Trabalho

master	Versões estáveis
hotfix	Correção de bugs da versão estável
release	Teste e correções de versões
develop	Desenvolvimento
feature	Implementação de funcionalidades

Fluxo de Trabalho



Fluxo de Trabalho



Utilitário Git Flow

Exemplos:

```
git flow init

git flow feature start   nome-da-feature
git flow feature publish nome-da-feature
git flow feature track   nome-da-feature
git flow feature finish  nome-da-feature

git flow release start   versao-da-release

git flow hotfix  start   versao-da-hotfix
```

Commit Semântico

Fundamentação

- Inspirado nas diretrizes do Angular Commit Messages Convention
- Boa prática para padronizar commits
- Facilita
  - Leitura
  - Consulta
  - Organização
  - Auxilia tanto humanos quanto máquinas

https://www.conventionalcommits.org/en/v1.0.0/

Estrutura

```
<tipo>[(escopo)][!]: <descricao>
<linha em branco>
[corpo]
<linha em branco>
[rodape]
```

- **Tipo**: especifica a atividade que foi realizada
- **Escopo**: parte do código que sofreu a alteração
- **Descrição**: o que foi feito no commit
- **Corpo**: detalhes do que foi feito no commit
- **Rodapé**: informa issue, id ou task utilizada na alteração do código

https://karma-runner.github.io/6.4/dev/git-commit-msg.html

Tipos

`fix`	corrige um problema
`feat`	inclui um novo recurso
`test`	adiciona/refatora testes
`refactor`	refatora código
`docs`	altera documentações
`style`	altera formatação
`ci`	altera arquivos e scripts de integração contínua
`build`	altera o sistema de compilação ou dependências externas
`perf`	melhora o desempenho
`improvement`	adiciona melhorias sem adicionar novo recurso ou corrigir problemas
`env`	modifica arquivos de configuração

Exemplos

- ```
  refactor!: drop support for Node 6

  BREAKING CHANGE: use JavaScript features not available in Node 6.
  ```
- ```
  docs(readme): remove incorrect tag from api document
  ```
- ```
  fix: adjust argument of profile function
  ```
- ```
  fix(middleware): ensure Range headers adhere more closely to RFC 2616

  Add one new dependency, use `range-parser` (Express dependency) to compute
  range. It is more well-tested in the wild.

  Fixes #2310
  ```
- ```
  build(npm): update fsevents to 1.0.14 (#11686)
  ```
- ```
  feat(facade): add bool type
  ```
- ```
  perf(dom): Only send values for existing properties to js interior
  ```

https://jorisroovers.com/gitlint/latest/
https://dev.to/vitordevsp/padronizacao-de-commit-com-commitlint-husky-e-commitizen-3g1n
https://www.freecodecamp.org/news/how-to-use-commitlint-to-write-good-commit-messages/
  ```

Commit Assinado

Commit Assinado

Para gerar as chaves privada e pública:

```
gpg --full-gen-key
```

Para listar as chaves:

```
gpg --list-secret-keys --keyid-format LONG <email>
```

Para visualizar a chave pública e poder adicionar ao Gitlab/Github:

```
gpg --armor --export <GPG_KEY_ID>
```

Informar o Git qual chave será utilizada para assinar os commits:

```
git config --global user.signingkey <GPG_KEY_ID>
```

Commit Assinado

Assinando o commit:

```
git commit -S -m “...mensagem...”
```

ou

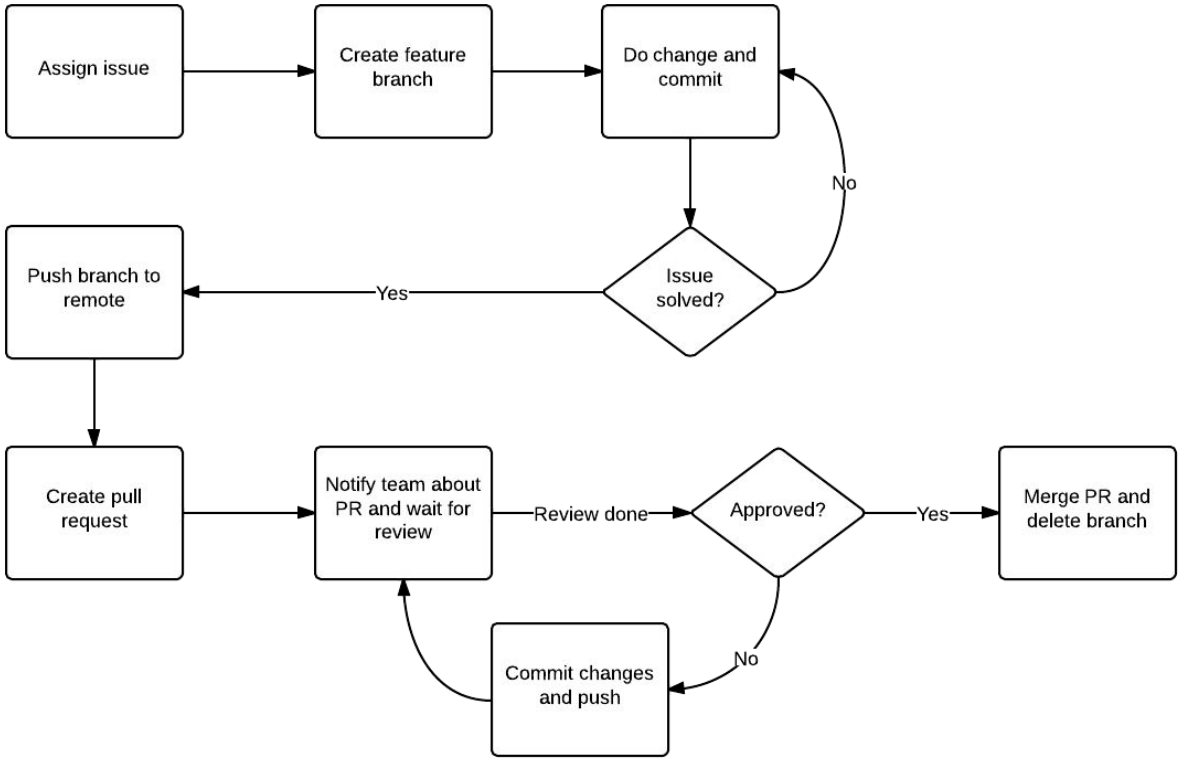
```
git config --global commit.gpgsign true
```

Para WSL2 é preciso o comando abaixo para que seja solicitada a senha da chave:

```
export GPG_TTY=$(tty)
```

Pull Request

Pull Request



Pull Request

```
git request-pull v0.0.1 https://git.ko.xz/project feature/teste

---

git push origin feature/teste

gh auth login

gh pr create --base main --head feature/teste --title “Descrição”
```

In case of fire

1. git commit

2. git push

3. leave building