

Tchelinix 2019  
Unipampa – Campus Bagé

# Git em projetos acadêmicos



[github.com/sandrocustodiobr/palestras](https://github.com/sandrocustodiobr/palestras)

# Git em projetos acadêmicos

Sandro Custódio

TI desde 1993

Servidor Público desde 1995

Na Justiça Federal desde 1999

Anos 90: Programador e Operador de CPD

Anos 2000: Sysadmin e suporte a usuário

Desde 2005: Suporte a Usuários.

Membro do Tchelinux.org desde 2008.

Em 2018: Retornado ao desenvolvimento.

Contexto:  
**Pequenos projetos**

Projetos com poucos participantes que buscam desenvolver um sistema ou ferramenta e precisam integrar seus esforços num produto final.

# Roteiro

- Conceitos
- Config. Inicial
- Commit local
- Repositório Remoto: pull/push
- Branch / Merge
- Repositório num pendrive ou rede
- Extra - Comandos / Extensões VS Code

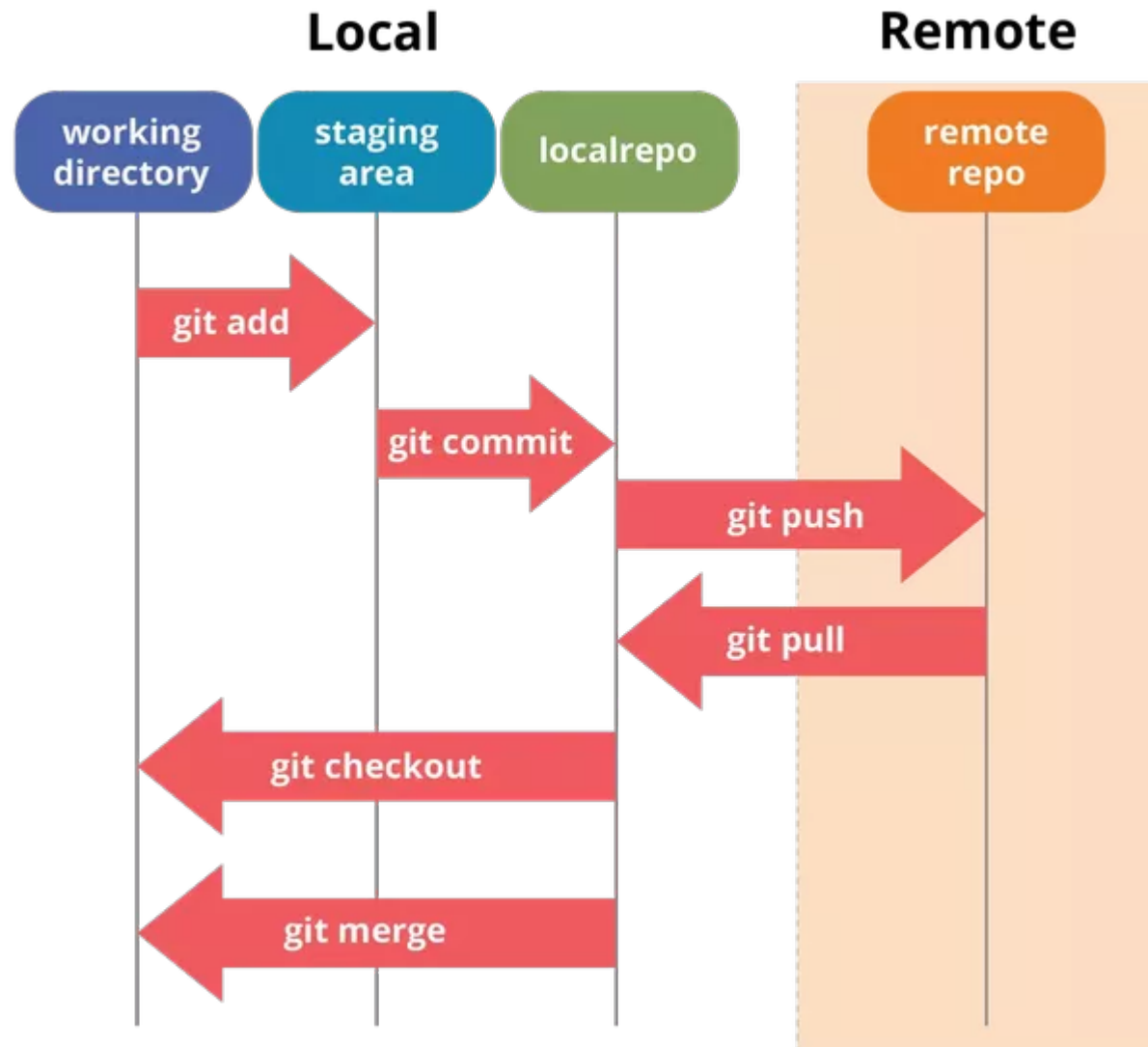
# Git – O que é?

- Manter histórico de alterações no código
- Autores (Quem?)
- Data das alterações (Quando?)
- Motivo das alterações (Por que?)

# Git – O que é?

- Facilita a colaboração (trabalho em equipe).
- Auxilia na identificação e resolução de problemas.
- Histórico das m..., digo, dos problemas.
- Permite fechar versões.

# Git – Conceitos



# Git – Conceitos

- Working directory
  - Arquivos e pastas do seu projeto.
- Staging area
  - Arquivos ou pastas modificados e selecionados para o próximo commit.
- Repositório local
  - Pasta local do git (.git) onde ficam os arquivos que contém os vários commits realizados.
- Repositório remoto
  - Pasta ou serviço remoto onde seu projeto está armazenado, nos mesmos moldes do repositório local



# Git – Conceitos

- Working directory
  - Armazena arquivos e pastas do seu projeto.
  - Contém personalização, ignorando arquivos e pastas desnecessárias ao projeto.
  - Nesta camada se decide o que entrará no projeto e o que será ignorado do controle de versão dos arquivos.
  - Principais comandos:
    - \$ git status
    - \$ git add <arquivos>

# Git – Conceitos

- Staging area
  - Antes do commit, os arquivos são selecionados para ser comitados
  - Comandos:
    - \$ git add <arquivo>
    - \$ git add .    # adiciona todos os arquivos modificados

# Git – Conceitos

- Repositório local
  - Pasta local do git (.git) onde ficam os arquivos que contém os vários commits realizados.
  - Armazena cada alteração de cada arquivo controlado do projeto.
  - Contém todos os commits de todos os participantes do projeto em todos os micros.
  - Comando básico:
    - \$ git commit ...
    - \$ git commit -m “Descricao (sem acentos) dos commit”

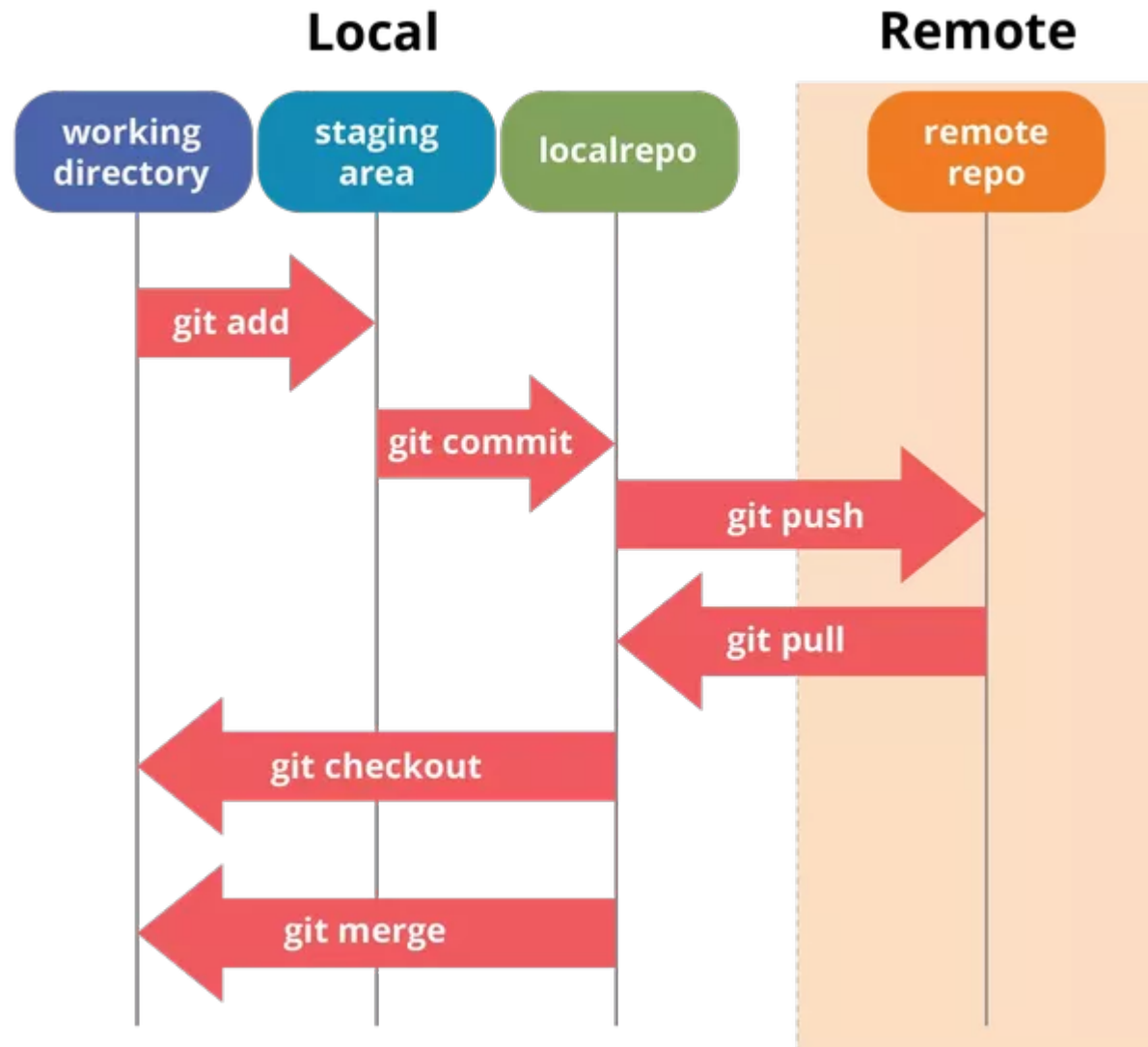
# Git – Conceitos

- Repositório remoto
  - Pasta ou serviço remoto onde seu projeto está armazenado, nos mesmos moldes do repositório local.
  - Geralmente é um projeto num site de hospedagem de projetos git.
  - <https://github.com>
  - <https://gitlab.com>
  - <https://bitbucket.com>
  - Repositório numa pasta da rede local.
  - Repositório numa pasta de um pendrive.

# Git – Conceitos

- Todos os repositórios (locais e remoto) contêm todos os conteúdos de todos os commits.
- Basta um repositório ok para todos serem restaurados.
- Todos os micros envolvidos ficam sempre sincronizados (atualizados).

# Git – Conceitos



# Git - Configuração Inicial

Do seu usuário Git, só uma vez por máquina

```
$ git config --global user.name "Fulano de Tal"
```

```
$ git config --global user.email "asd@asd.com"
```

# Git - Configuração Inicial

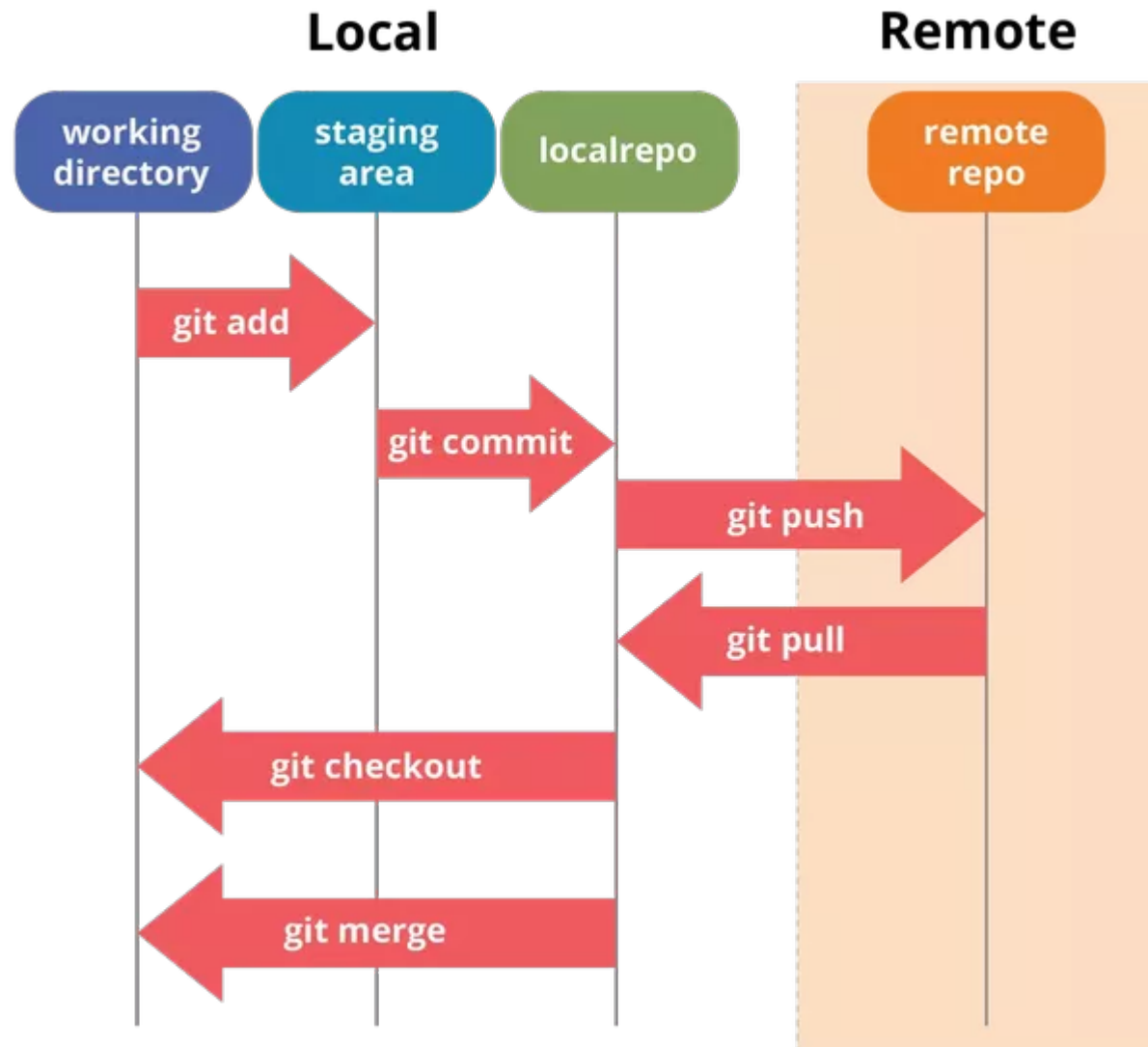
Criando/inicializando o git em um projeto:

```
$ cd pasta_do_projeto
```

```
$ git init
```



# Git – Commit local



# Commit

- Commit: Registro das alterações no repo local
  - linha de comando
  - VS Code
  - Netbeans
  - Etc.

# Commit – linha de comando

Ver situação/status

```
$ git status
```

Adicionando arquivos para o próximo commit

```
$ git add <arquivos>
```

Comitando no repositório local

```
$ git commit -m "Primeiro commit do projeto."
```

# Commit – mensagens

Bons exemplos (simples)...

```
$ git commit -m "Cliente add, ajustes formularios"
```

```
$ git commit -m "Mascara telefone ajustada"
```

```
$ git commit -m "Fix bug ao add sem telefone"
```

# Commit – mensagens

Maus exemplos...

```
$ git commit -m "Commit"
```

```
$ git commit -m "xyz"
```

```
$ git commit -m "hahaha"
```

# Commit – mensagens

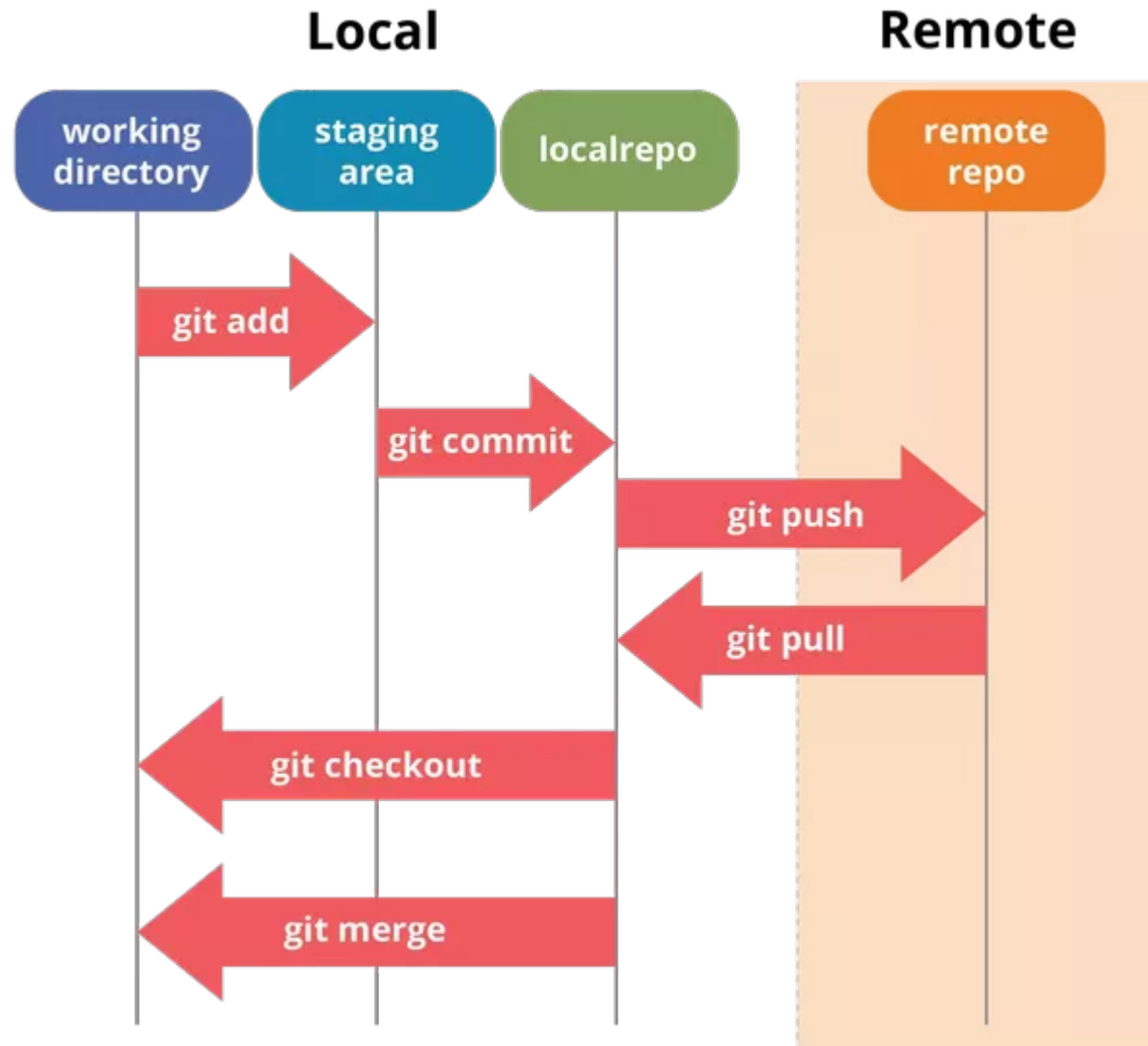
```
$ git commit
```

(vai abrir um editor e vai te permitir um texto completo)

Alteracao nos formularios do Cliente

Ajustes nas mascaras de telefone, e-mail e nome, com validacao dos mesmos, SANITIZE no SQL e detalhes menores.

# Git – Remote: pull e push



# Git - Remote

- Adicionar um repositório remoto

```
$ git remote add origin https://github.com/...
```



# Git - Pull

- Traz as alterações dos colegas antes de iniciarmos os trabalhos na máquina local

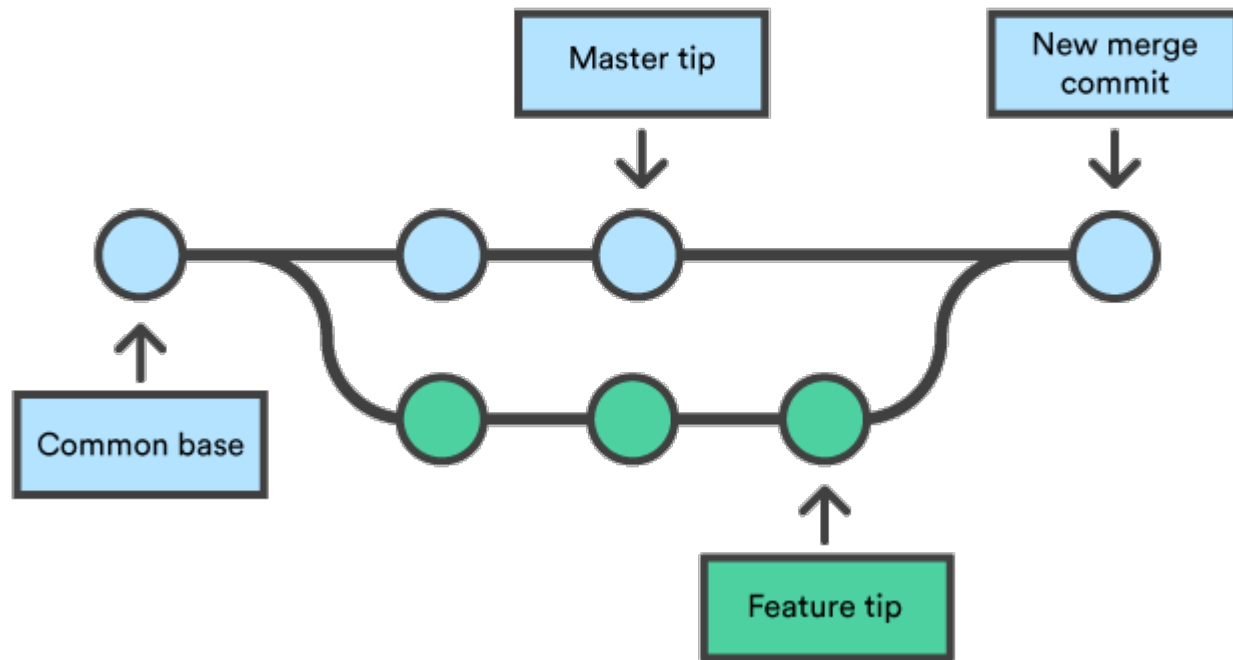
```
$ git pull origin master
```

# Git - Push

- ENVIA as alterações da máquina local para o repositório remoto, unificando o projeto.

```
$ git push origin master
```

# Branches



# Git - Branch

# cria o branch

\$ git branch teste

# passa para o branch teste (sai do master)

\$ git checkout teste

# cria o branch teste já fazendo o checkout

\$ git checkout -b teste

# Git - Branch

# para sair do branch teste e voltar ao master

\$ git status

# ou informar o que foi feito até o momento como "funcoes basicas"

\$ git commit -a -m "ponto de commit para poder voltar depois"

\$ git checkout master

# depois, para voltar ao teste e continuar o trabalho

\$ git checkout teste

# Git - Branch

# trazendo as alterações do branch de teste para o branch master (unifincando no master)

# vai para o master

\$ git checkout master

# traz as alteracoes de teste para master

\$ git merge teste

# Git - Branch

# caso não precise mais trabalhar no branch teste, podemos apaga-lo

```
$ git branch -d teste
```

# Git – Merge

Um exemplo...

# em caso de conflito ao fazer merge

\$ git checkout -b teste2 # cria novo merge e faz checkout nele

... alterar um arquivo em determinadas linhas ...

\$ git commit -a -m "bug em x e y corrigido, agora com parametro \$ID"

... enquanto isso, outro colega no branch master ...

(git checkout master)

... altera o mesmo arquivo quase nas mesmas linhas ...

\$ git commit -a -m "bug em x e z corrigido, agora com verif. de \$valor"

... quando voce tenta fazer merge (unificar) com master ...

(git checkout master)

\$ git merge teste2 # ele faz o merge mas vai informar que há conflitos

... no código fonte onde há conflitos, o git insere ambas as versões das linhas com os devidos comentários ...

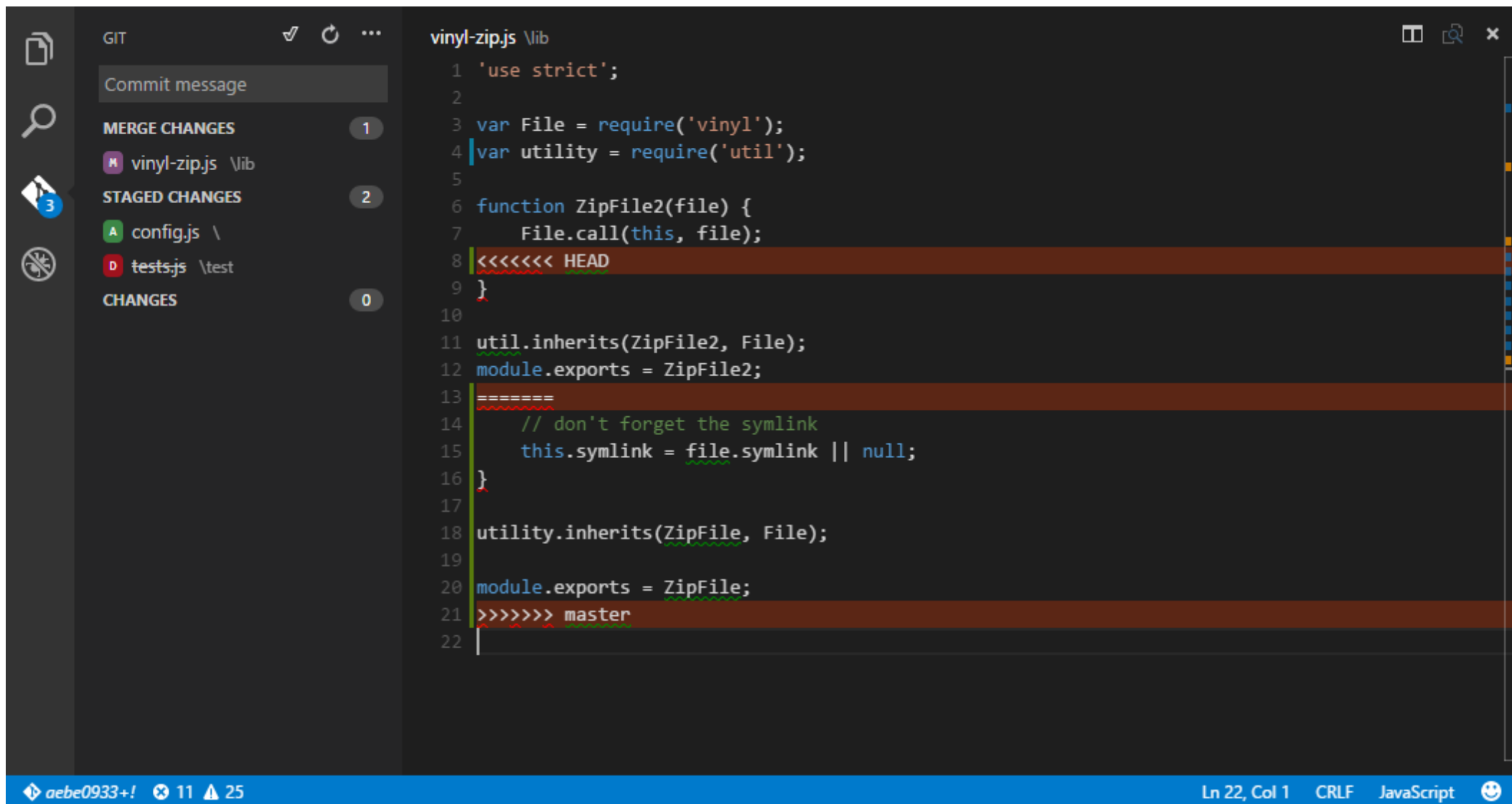
... e voce edita diretamente no código fonte para deixar apenas a versão que vai ficar ...

# commit dos arquivos já sem os conflitos (isto já no master, após o merge)

\$ git commit -a -m "bug em x, y e z corrigido, agora com parametro \$ID e verif. de \$valor"



# Git – Merge

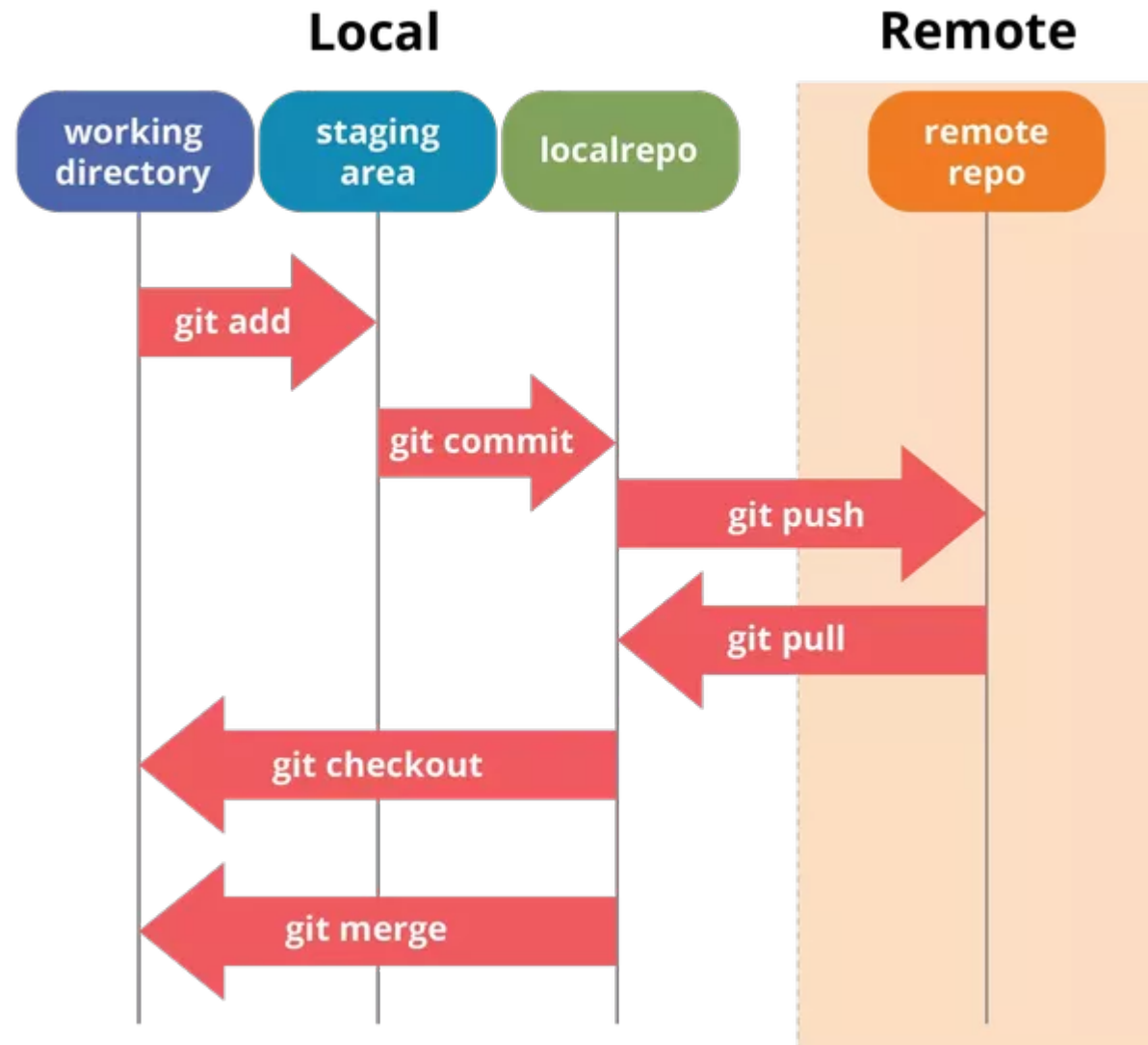


The screenshot shows a code editor with a dark theme. On the left, a sidebar displays the Git interface with a 'Commit message' field and a list of changes: 'MERGE CHANGES' (1), 'vinyl.zip.js \lib' (M), 'STAGED CHANGES' (2), 'config.js \' (A), 'tests.js \' (D), and 'CHANGES' (0). The main editor area shows the file 'vinyl.zip.js \lib' with JavaScript code. A merge conflict is visible, with lines 8 and 21 highlighted in brown. Line 8 is marked with '<<<<<<< HEAD' and line 21 with '>>>>>> master'. The code includes 'use strict', 'require' calls for 'vinyl' and 'util', a 'ZipFile2' function, and 'util.inherits' calls. The status bar at the bottom shows 'Ln 22, Col 1', 'CRLF', 'JavaScript', and a smiley face icon.

```
vinyl.zip.js \lib
1  'use strict';
2
3  var File = require('vinyl');
4  var utility = require('util');
5
6  function ZipFile2(file) {
7      File.call(this, file);
8  <<<<<<< HEAD
9  }
10
11  util.inherits(ZipFile2, File);
12  module.exports = ZipFile2;
13  =====
14      // don't forget the symlink
15      this.symlink = file.symlink || null;
16  }
17
18  utility.inherits(ZipFile, File);
19
20  module.exports = ZipFile;
21  >>>>>> master
22
```

aebe0933+! 11 25 Ln 22, Col 1 CRLF JavaScript

# Git – Repositório num pendrive



# Git – Repositório num pendrive

# criando o repositório central numa pasta

# pode ser em rede, local ou num pendrive

\$ cd nome\_da\_pasta

**\$ git init --bare**

# Git – Repositório num pendrive

**# a primeira máquina**

\$ cd nome\_da\_pasta

\$ git init

(fazer alterações)

\$ git remote add origin /caminho/ate/a/pasta

\$ git push origin master

# Git – Repositório num pendrive

**# as demais máquinas**

```
$ cd nome_da_pasta
```

```
$ git init
```

```
$ git remote add origin /caminho/ate/a/pasta
```

```
$ git pull origin master
```

(fazer alterações)

```
$ git push origin master
```

# Git – Repositório num pendrive

## **# forçando um Merge**

Na máquina A alterar linha N de um arquivo.

Na máquina B alterar a mesma linha.

Fazer push em ambas as máquinas.

# Git – Comandos e Extensões

Comandos:

status, remote, pull, push, clone, fetch, branch, checkout, merge, diff, log, ...

VS Code:

Git History

Git Lens

# Git – Exemplos práticos

Ver comandos no arquivo extra em  
[github.com/sandrocustodiobr/palestras](https://github.com/sandrocustodiobr/palestras)

VS Code, integrado na barra em



Netbeans, botão direito.



# In case of fire



1. git commit



2. git push



3. leave the building

# **Sandro Custódio**

Tchelinix desde 2008

Servidor Publico Federal desde 1999

TI desde 1993



[github.com/sandrocustodiobr/palestras](https://github.com/sandrocustodiobr/palestras)