



# Git e GitHub

Treinamento: JavaScript, Node e React – Entra21  
Instrutor: Ivan J. Borchardt  
©2023

# Versionamento de Software

➤ Forma “manual” de fazer versionamento de software...



Site Cliente XPTO



Site Cliente XPTO.zip



Site Cliente XPTO V2.zip



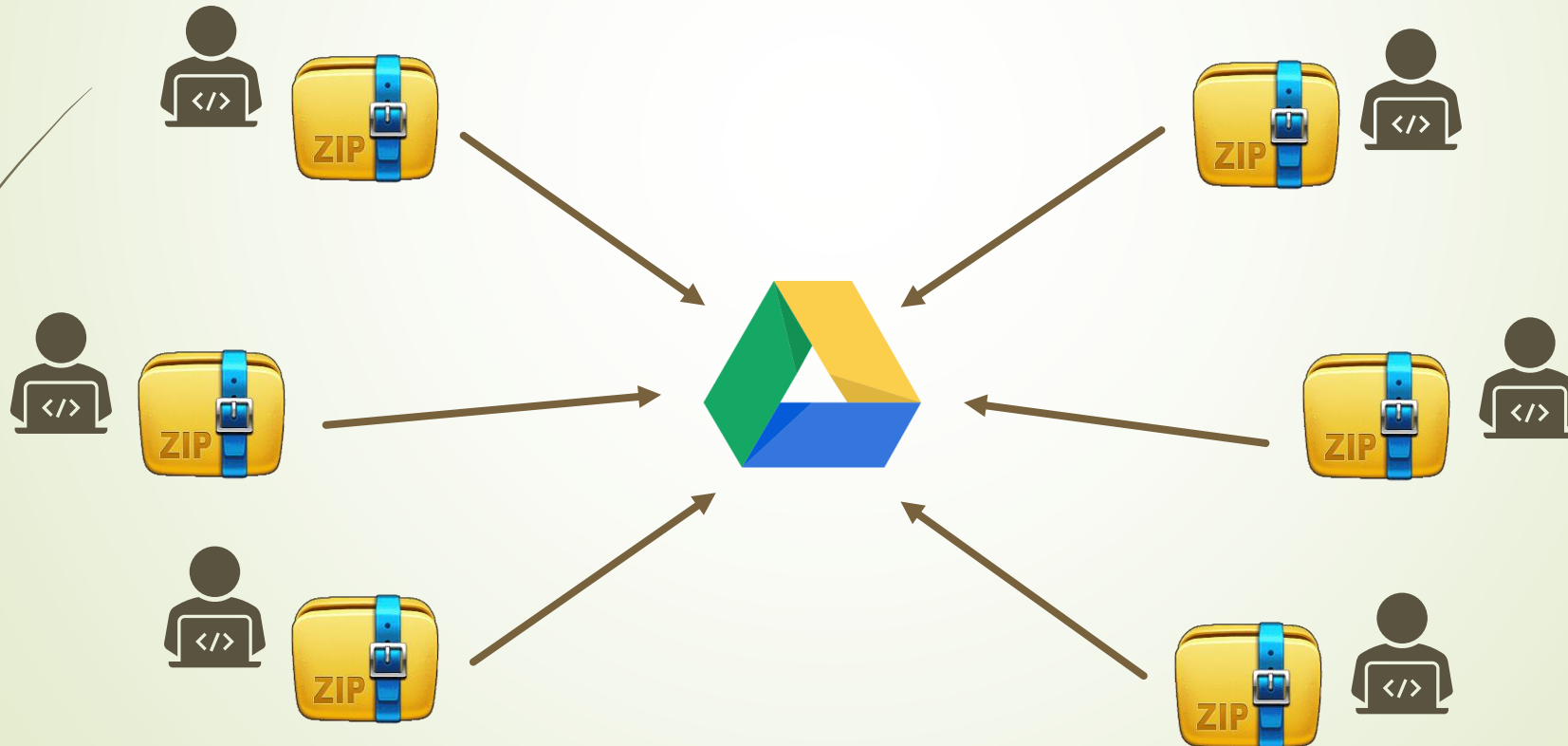
Site Cliente XPTO Bugado.zip



Site Cliente XPTO Bug Ok.zip

# Versionamento de Software

- E se o projeto crescer e o Site passar a ser desenvolvido por vários programadores...?





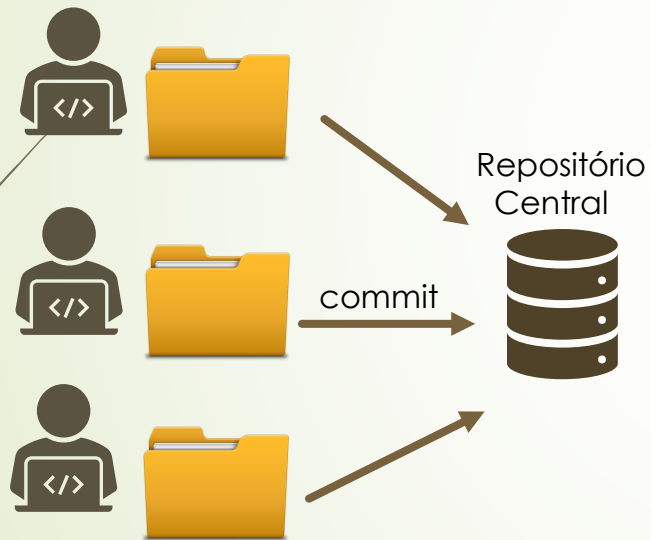
# Versionamento de Software

- 1972 – Primeiro Software de Controle de Versão (VCS).
- Um sistema de controle de versão ou VCS, também conhecido como sistema de controle de revisão ou sistema de controle de fonte, é um utilitário de software que monitora e gerencia as mudanças em um sistema de arquivos.
- Atualmente existem dois tipos de VCS: centralizado e distribuído.

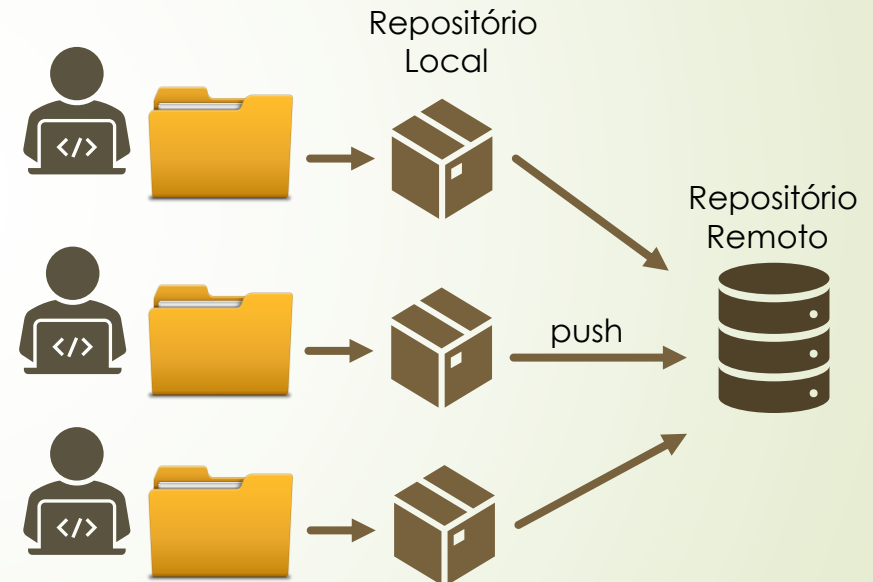
# Versionamento de Software

## Software de Controle de Versão (VCS)

### Centralizado/Linear



### Distribuído





# Versionamento de Software

## Software de Controle de Versão (VCS)

### Vantagens:

- Controle de Histórico
- Trabalho em Equipe
- Ramificação do Projeto
- Segurança
- Organização

# Versionamento de Software

## Software de Controle de Versão (VCS)

### Centralizado/Linear

- ✓ CA Software Change Manager (CCC)
- ✓ Source Code Control System (SCCS)
- ✓ Panvalet
- ✓ Concurrent Version System (CVS)
- ✓ Apache Subversion (SVN)
- ✓ ClearCase
- ✓ Visual SourceSafe
- ✓ Perforce

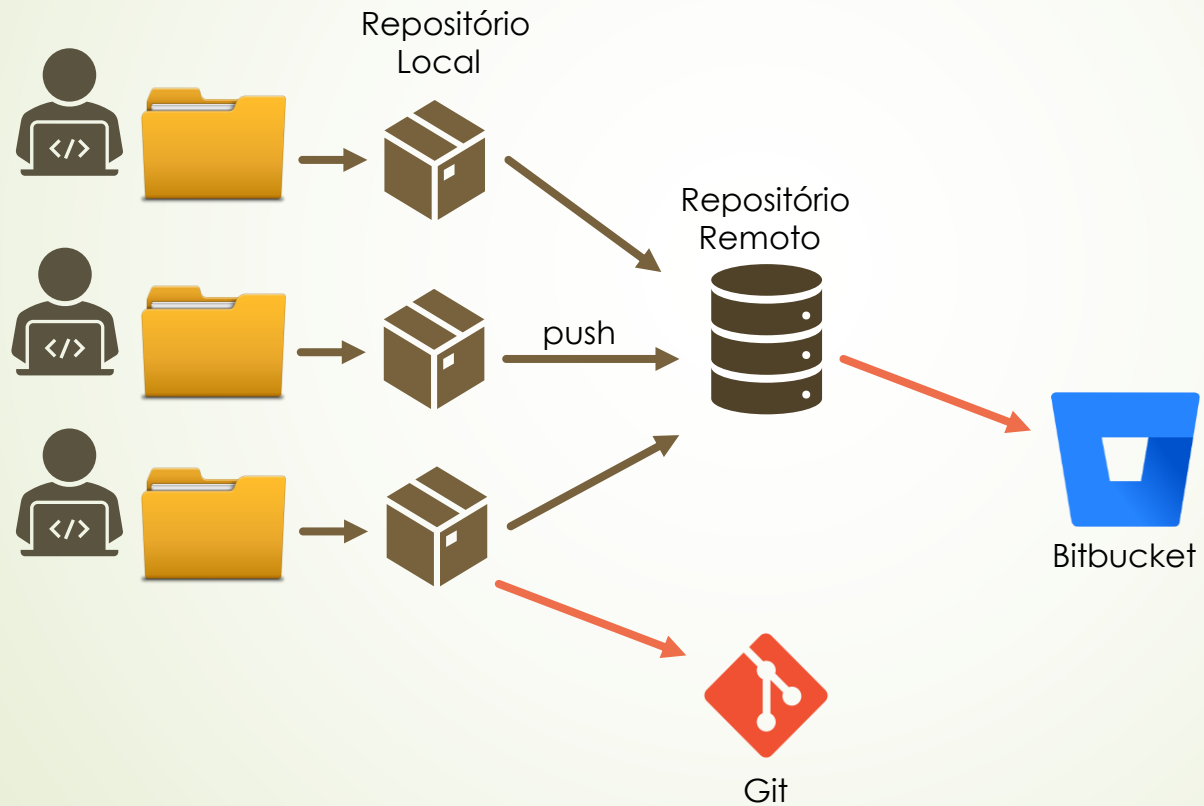
### Distribuído

- ✓ Mercurial
- ✓ Bazaar
- ✓ Code Co-op
- ✓ GNU arch
- ✓ Monotone
- ✓ Fossil
- ✓ BitKeeper
- ✓ Git



# Versionamento de Software

## Software de Controle de Versão (VCS)





# Versionamento de Software

Repositórios remotos compatíveis com Git



Bitbucket



GitHub



GitLab



PHABRICATOR



Gogs



Kallithea

# Versionamento de Software

## História e curiosidades...



# Versionamento de Software

## História e curiosidades...

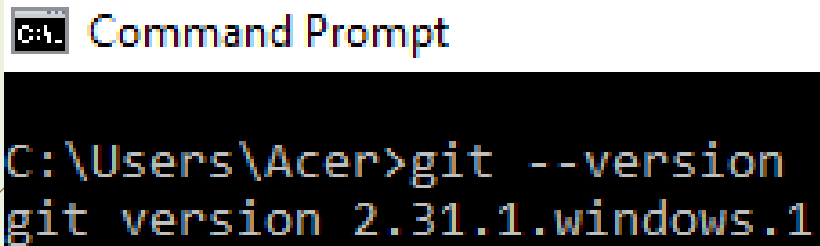




# Git e GitHub

# Git

Teste a instalação do Git:



```
C:\Users\Acer>git --version  
git version 2.31.1.windows.1
```

Faça o Download e Instale o Git:

➡ <https://git-scm.com/>

# Git

## Começando com o git Passo-a-Passo...

### 1. Instalar o git (<https://git-scm.com/>)

- `git --version`

### 2. Configurar o git

- `git config --global user.email "you@example.com"`

- `git config --global user.name "Your Name"`

Ou

- `git config --local user.email "you@example.com"`

- `git config --local user.name "Your Name"`

\*\* --global → Configuração para a máquina toda

\*\* --local → Configuração para o projeto

Visualizar as configurações válidas para o projeto:

`git config <propriedade>`

# Git

## Começando com o git Passo-a-Passo...

3. Navegar até a pasta do projeto
  - `cd..`
4. Inicializar o repositório
  - `git init`
  - `git status`
5. Adicionar arquivos ao stage
  - `git add <file>`
  - `git add .`
  - `git rm` → remove do repositório (cuidado para não deletar arquivos)
6. Commitar
  - `git commit -m "Criando website XPTO"`
7. Adicionar arquivos ao stage e commitar
  - `git commit -am "Criando website XPTO"` (

**\*\*com o argumento -am é possível adicionar a alteração ao Stage e commitar no mesmo comando)**





# Git

## Começando com o git Passo-a-Passo...

### 7. Verificando as alterações

- `git log`
- `git log --oneline`
- `git log -p`

\*\* Outros formatos de log: <https://devhints.io/git-log>

# Git

## Começando com o git Passo-a-Passo...

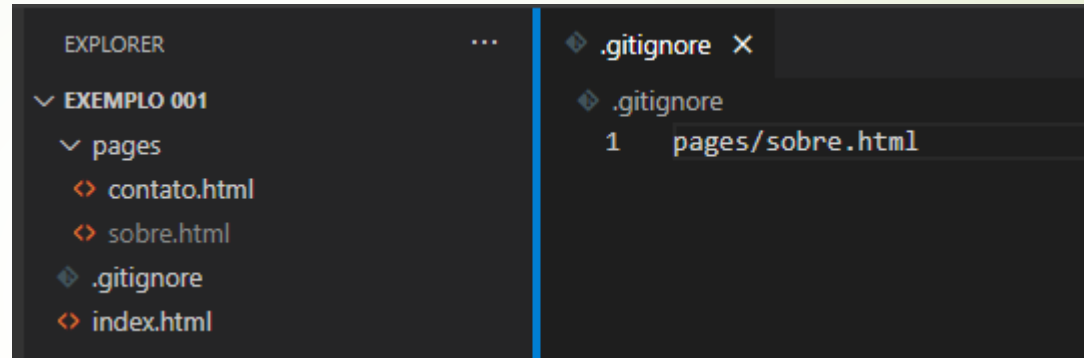
### 8. Ignorando arquivos

.gitignore

<filename>

<dir/filename>

- git add .gitignore
- git commit -m "Adicionando .gitignore"



The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar displays a file tree for a project named 'EXEMPLO 001'. It contains a 'pages' directory with files 'contato.html' and 'sobre.html', and a root-level '.gitignore' file. On the right, the Editor pane shows the content of the '.gitignore' file, which contains the line 'pages/sobre.html'.

```
EXPLORER
└─ EXEMPLO 001
   └─ pages
      ├── contato.html
      └── sobre.html
  .gitignore
  index.html
```

```
.gitignore
1  pages/sobre.html
```



# Git

Começando com o git Passo-a-Passo...

## Quando devo commitar?

1. Nunca commite algo que não está funcionando!
2. Commite pequenas funcionalidades concluídas.

# Git

## Recuperando uma versão...

### 1. Verificando as alterações

- `git log`
- `git log --oneline`
- `git log --graph`

\*\* Outros formatos de log: <https://devhints.io/git-log>

### 2. Posicionando o HEAD em um commit específico

- `git ckeckout <id commit>`

### 3. Verificando a branch

- `git branch`

### 4. Criando uma nova branch a partir do commit selecionado

- `git branch <nomeNovaBranch>`

### 5. Voltando para a branch master

- `git checkout master`

# Git

## Desfazendo alterações antes do ADD .

1. Verificando diferenças
  - `git diff`
2. Verificando o Status
  - `git status`
3. Descartando as alterações do diretório de trabalho
  - `git checkout <file>`
  - ou
  - `git restore <file>`

# Git

## Desfazendo alterações já adicionadas ao Stage

### 1. Verificando o Status

- `git status`

### 2. Resetando as alterações de volta ao Unstage

- `git reset HEAD <file>`

ou

- `git restore --staged <file>`

### 3. Verificando as diferenças

- `git diff`

\*\* Perceba que após adicionar as alterações ao Stage não é mais possível verificar diferenças

### 4. Verificando o Status

- `git status`

### 5. Descartando as alterações do diretório de trabalho

- `git checkout <file>`

ou

- `git restore <file>`



# Git

## Excluindo um commit

1. Verificando o log
  - `git log --oneline`
2. Removendo o commit
  - `git reset --hard <id commit>`
3. Verificando o log
  - `git log --oneline`



# Git

## Criando uma nova Branch

1. Verificando em qual branch o git está
  - `git branch`
  - `git branch -a` (lista todas as branches remotas e locais)
2. Criando uma nova branch
  - `git branch <nomeNovaBranch>`
3. Alternando de uma branch para outra
  - `git checkout <nomeNovaBranch>`
4. Criando e alternando para uma nova branch
  - `git checkout -b <nomeNovaBranch>`
5. Verificando a nova Branch
  - `git log --oneline`
6. Verificando o grafo de commits após alterações nas branches
  - `git log --oneline --graph --all`
  - `git log --graph --all`
7. Renomeando uma Branch
  - `git branch -m <novoNomeDaBranch>`
8. Deletando uma Branch
  - `git branch -D <nomeDaBranch>` (Força a deleção)
  - `git branch -d <nomeDaBranch>` (Só permite deletar branches já mergeadas ou sincronizadas com o repositório remoto)

\*\*Precisa estar "Logado" em outra branch

# Git

## Mesclando Branches

### 1. Unindo duas branches

➤ `git merge <nomeDaBranch>`

\*\* Executar dentro da branch que receberá as alterações

➤ `git merge --no-ff <nomeDaBranch>`

\*\* Faz o merge e cria um novo commit

### 2. Resolvendo conflitos

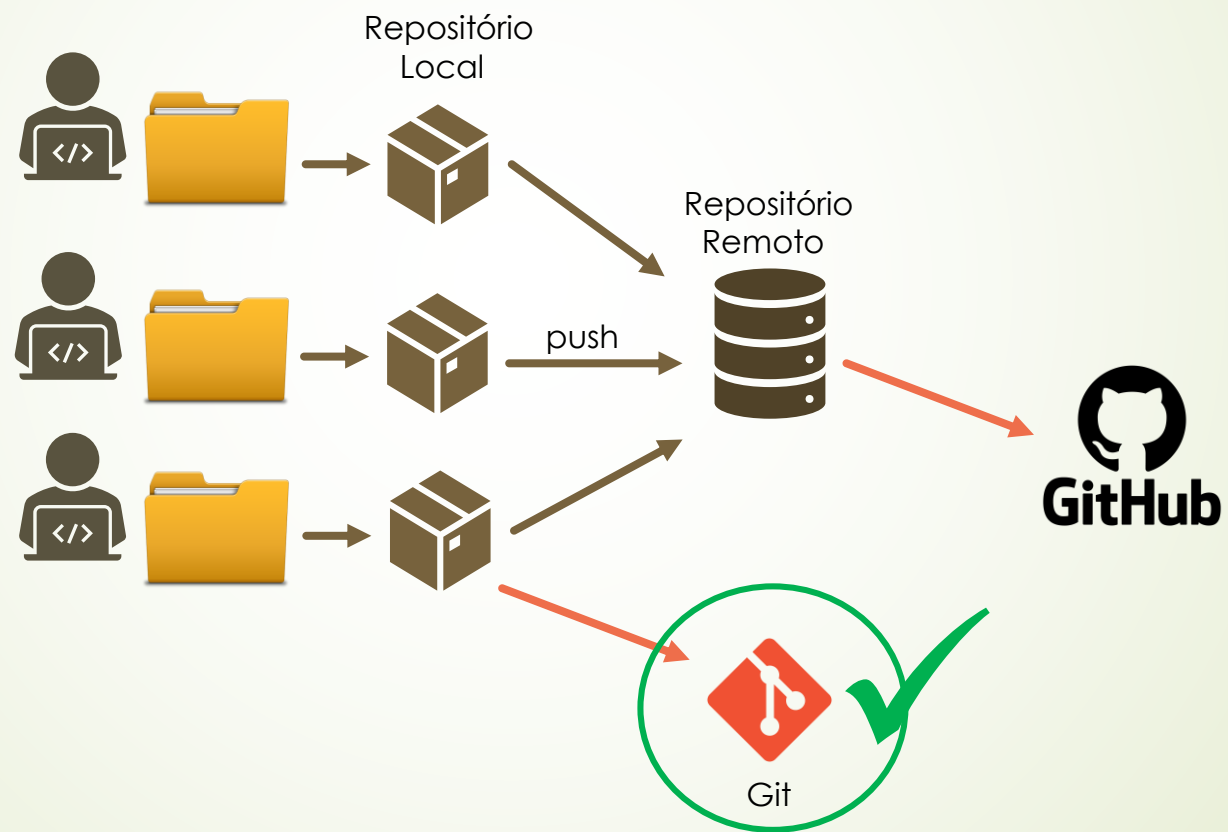
a) Mesclar os arquivos no editor (opção Accept Both Changes)

b) `git status`

c) `git add .`

d) `git commit -m "unindo branches..."`

# Git e GitHub

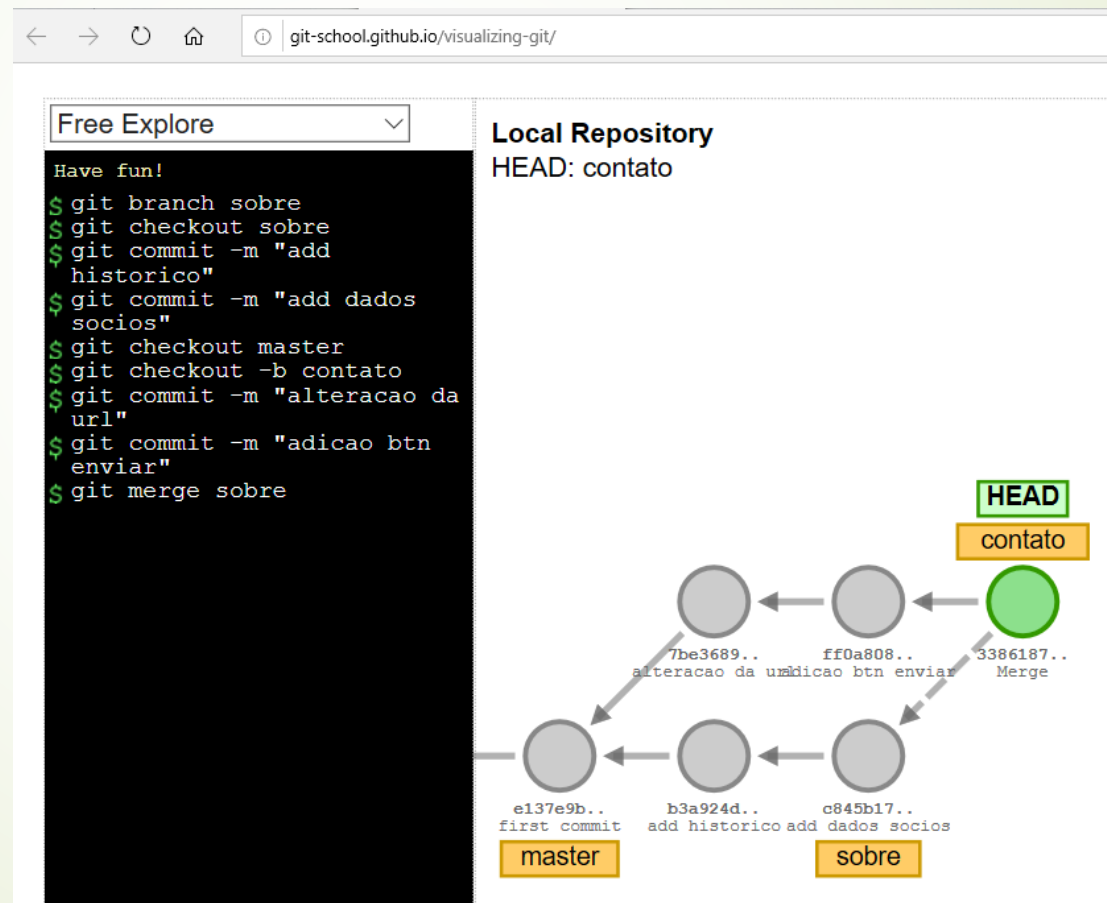


# Git e GitHub

## Trabalhando em equipe...

### Branches

➤ Visualizing Git



# Git e GitHub

## Trabalhando em equipe...

### 1. Clonando um repositório remoto

- `git clone <url do repositório remoto>`
- `git clone <url do repositório remoto> <nome_Novo_Diretório>`

### 2. Trabalhando com Branch

- `git branch -a`  
\*\* Lista todas as Branchs (locais e remotas)
- `git branch <nome da Nova Branch>`  
\*\* Cria nova branch
- `git checkout <nome da Branch>`  
\*\* Seleciona uma Branch

# Git e GitHub

## Trabalhando em equipe...

### 3. Enviando as alterações do repositório local para o remoto

- `git push`

\*\* Envia o código da Branch atual para a Branch correspondente no Repositório Remoto apontado pela Origin

- `git push <REMOTE_NAME>`

\*\* Envia o código da Branch atual para a Branch correspondente no Repositório Remoto apontado por REMOTE\_NAME

- `git push <REMOTE_NAME> <BRANCH_NAME_LOCAL>`

\*\* Envia o código da Branch apontada por BRANCH\_NAME\_LOCAL para a Branch correspondente no Repositório Remoto apontado por REMOTE\_NAME

- `git push <REMOTE_NAME> <BRANCH_NAME>: <BRANCH_NAME_REMOTE>`

\*\* Envia o código da Branch apontada por BRANCH\_NAME\_LOCAL para a Branch BRANCH\_NAME\_REMOTE no Repositório Remoto apontado por REMOTE\_NAME

# Git e GitHub

## Trabalhando em equipe...

### 4. Deletando Branches

➤ `git branch -D <BRANCH_NAME>`

\*\* Deleta a Branch Local

➤ `git push -D <REMOTE_NAME> <BRANCH_NAME>`

\*\* Deleta a Branch Remota

### 5. Atualizando o repositório Local (com fetch)

A. `git fetch` ou `git fetch <REMOTE_NAME>`

\*\* Baixa as atualizações do repositório remoto para o local

B. `git merge origin/master`

\*\* Mescla as atualizações baixadas com a Branch master local



# Git e GitHub

## Trabalhando em equipe...

5. Atualizando o repositório Local (com pull)
  - A. `git pull`
  - B. `git pull <REMOTE_NAME>`
  - C. `git pull <REMOTE_NAME> <BRANCH_NAME_LOCAL>`
  - D. `git pull <REMOTE_NAME> <BRANCH_NAME_LOCAL> : <BRANCH_NAME_REMOTE>`



# Git e GitHub

## Trabalhando em equipe...

### 6. Erro ao tentar fazer um git push

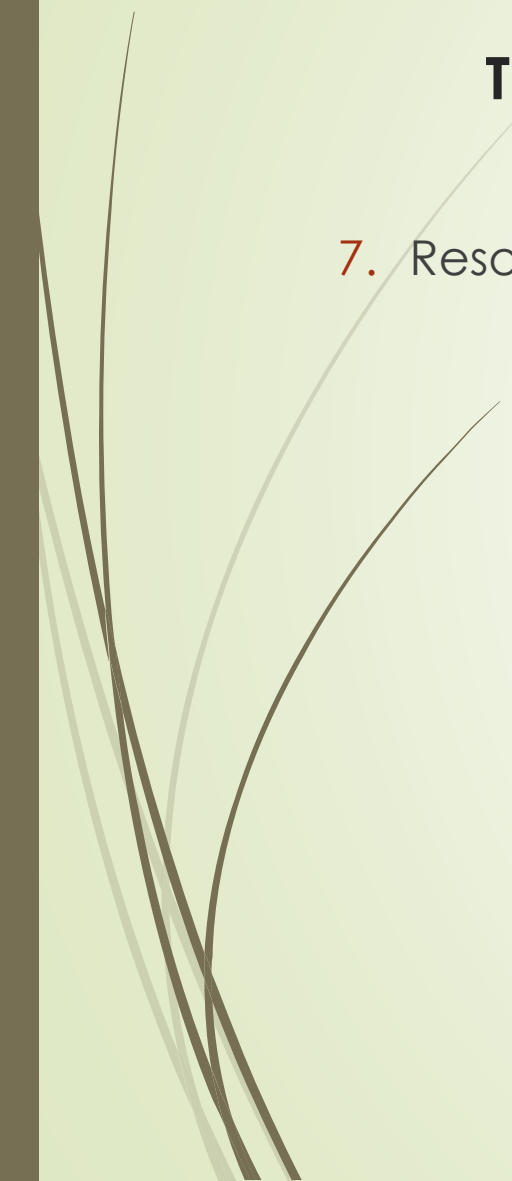
\*\* Não é possível fazer um git push se os repositórios não estão sincronizados. É necessário fazer um git pull primeiro, resolvendo eventuais conflitos.



# Git e GitHub

## Trabalhando em equipe...

### 7. Resolvendo conflitos





# Git e GitHub

## Fluxos de Trabalho...

### 1. GitHub Flow

- A. Criar uma branch baseada na Master para a feature que será desenvolvida
- B. Trabalhar normalmente nessa branch, fazendo commits
- C. Abrir uma pull request
- D. Discussão e revisão das alterações
- E. Deploy em ambiente de produção a partir da Branch em trabalho
- F. Finalizar o merge

# Git e GitHub

## Fluxos de Trabalho...

### 2. Open Source Flow/Forking Flow

- A. Fazer um fork do repositório de interesse
- B. Clonar o fork para o ambiente de desenvolvimento
- C. Configurar o remote origin para que ele aponte para o nosso fork
- D. Configurar o remote upstream para que ele aponte para o repositório oficial
- E. Criar uma branch para desenvolvimento / Seguir GitHub Flow

Mais Fluxos de trabalho:

<https://www.atlassian.com/br/git/tutorials/comparing-workflows>