

GIT



@prof.felipeassuncao

mentorama.

APRESENTAÇÃO



mentorama.

Apresentação do módulo

- Parte 1 – O que é o GIT
- Parte 2 – Instalação
- Parte 3 – Começando com o GIT
- Parte 4 – Conceitos e Arquitetura
- Parte 5 – Alterando Arquivos
- Parte 6 – Repositório remoto com o Github

Recursos e materiais

- Disponibilização de comandos básicos
- Fontes de informação relevantes
- Exercícios

1. O QUE É GIT?



Perguntas

- Você deseja desenvolver em equipe?
- Como compartilhar código e colaborar em um conjunto de artefatos comuns?
- Como controlar os arquivos desenvolvidos por cada equipe?

Perguntas

- E se você deseja:
 - criar pontos na história de produção?
 - verificar alterações?
 - reverter alterações?
 - comparar alterações?
 - escolher a melhor das alterações?
 - juntar todos os artefatos do projeto?

O que é GIT?

- É um sistema de controle de versão distribuído, moderno e mais utilizado no mundo atualmente.
- É um projeto de código aberto maduro e com manutenção ativa.
- Funciona bem em uma ampla variedade de sistemas operacionais e IDEs (Ambientes de Desenvolvimento Integrado).



A HISTÓRIA POR TRÁS DO GIT



mentorama.

A história por trás do GIT

- Desenvolvido em 2005 por Linus Torvalds (criador do Kernel e do Linux)
- Controle de versão distribuído
- Mais rápido que outros SCM (até 100x mais)



A história por trás do GIT

■ Quais empresas utilizam o GIT?

- Adobe
- Apache Software Foundation
- Atlassian (JIRA, Confluence)
- BlackBerry
- Globo.com
- Google
- Petrobras
- Rede Globo
- Oracle



Prós e Contrás - GIT

▪ **Prós Git**

- Ótimo para quem odeia o **CVS/SVN**;
- Aumento dramático na velocidade de operação;
- Árvore de histórico completo disponível *off-line*;
- Modelo distribuído

▪ **Contrás Git**

- Maior curva de aprendizado para aqueles usados no **SVN**;
- Não é ideal para desenvolvedores únicos;
- Suporte limitado do **Windows**, comparado ao **Linux**.

CONTROLE DE VERSÃO DISTRIBUÍDO



mentorama.

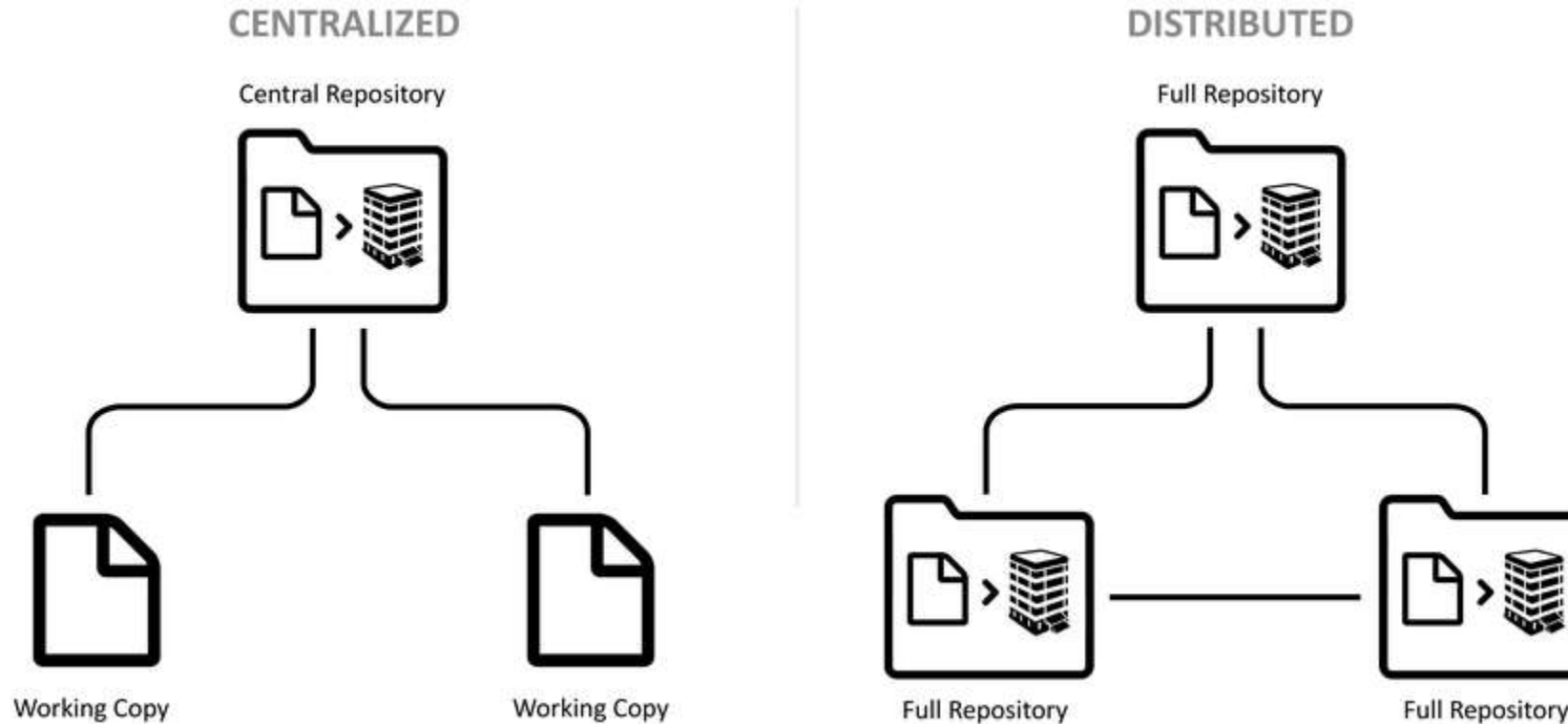
Sistemas de Controle de Versão

V · T · E			Version control software	[hide]
Years, where available, indicate the date of first stable release. Systems with names <i>in italics</i> are no longer maintained or have planned end-of-life dates.				
Local only	Free/open-source	RCS (1982) · SCCS (1972)		
	Proprietary	PVCS (1985) · QVCS (1991)		
Client–server	Free/open-source	CVS (1986, 1990 in C) · CVSNT (1998) · QVCS Enterprise (1998) · Subversion (2000)		
	Proprietary	AccuRev SCM (2002) · Azure DevOps (Server (via TFVC) (2005) · Services (via TFVC) (2014)) · ClearCase (1992) · CMVC (1994) · Dimensions CM (1980s) · DSEE (1984) · Endevor (1980s) · Integrity (2001) · Panvalet (1970s) · Perforce Helix (1995) · SCLM (1980s?) · Software Change Manager (1970s) · StarTeam (1995) · Surround SCM (2002) · Synergy (1990) · Team Concert (2008) · Vault (2003) · Visual SourceSafe (1994)		
Distributed	Free/open-source	ArX (2003) · BitKeeper (2000) · Codeville (2005) · Darcs (2002) · DCVS (2002) · Fossil (2007) · Git (2005) · GNU arch (2001) · GNU Bazaar (2005) · Mercurial (2005) · Monotone (2003)		
	Proprietary	Azure DevOps (Server (via Git) (2013) · Services (via Git) (2014)) · TeamWare (1992) · Code Co-op (1997) · Plastic SCM (2006)		
Concepts	Baseline · Branch · Changeset · Commit · Data comparison · Delta compression · Fork (Gated commit) · Interleaved deltas · Merge · Monorepo · Repository · Tag · Trunk			
Category · Comparison · List				

The diagram shows a sequence of numbered nodes (1-10) representing version control states. Node 1 is a green square labeled 'Trunk'. Node 2 is a yellow square labeled 'Branch'. Node 3 is a yellow square. Node 4 is a green square labeled 'Merge'. Node 5 is a yellow square. Node 6 is a yellow square. Node 7 is a yellow square. Node 8 is a yellow square. Node 9 is a green square labeled 'Tag'. Node 10 is a yellow square labeled 'Discontinued development branch'. Arrows indicate the flow: 1 to 2, 2 to 3, 3 to 4, 4 to 5, 5 to 6, 6 to 7, 7 to 8, 8 to 9, 9 to 10. A red arrow points from 4 to 9. A blue box labeled 'T1' is next to node 4, and a blue box labeled 'T2' is next to node 9.



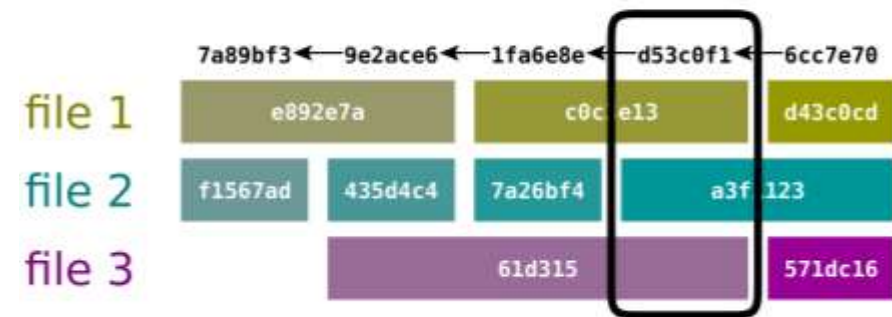
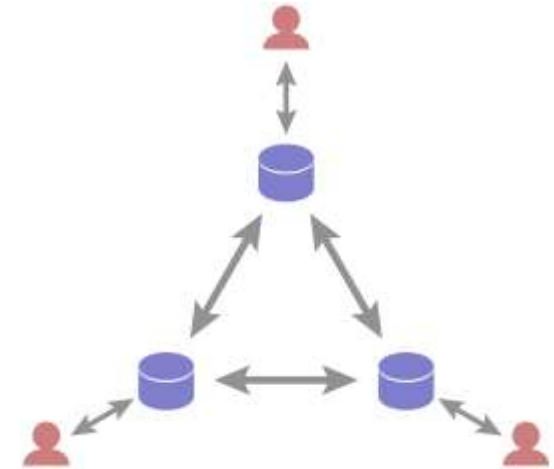
Diferenças entre controles de versão



Controle de versão distribuído

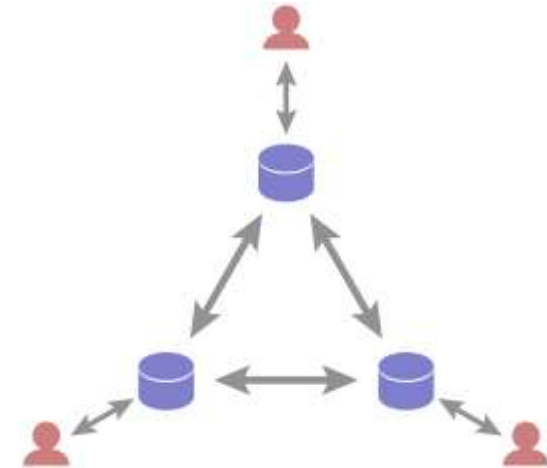
Características principais:

- Salva seu histórico
- Você pode desenvolver versões diferentes
- Programar em paralelo



Controle de versão distribuído

- Todo o desenvolvedor tem uma cópia local do projeto em que está trabalhando, ou seja, não é necessário acesso a internet para criar seu histórico de alterações.
- Permite melhor controle do código, das versões e independência no desenvolvimento.

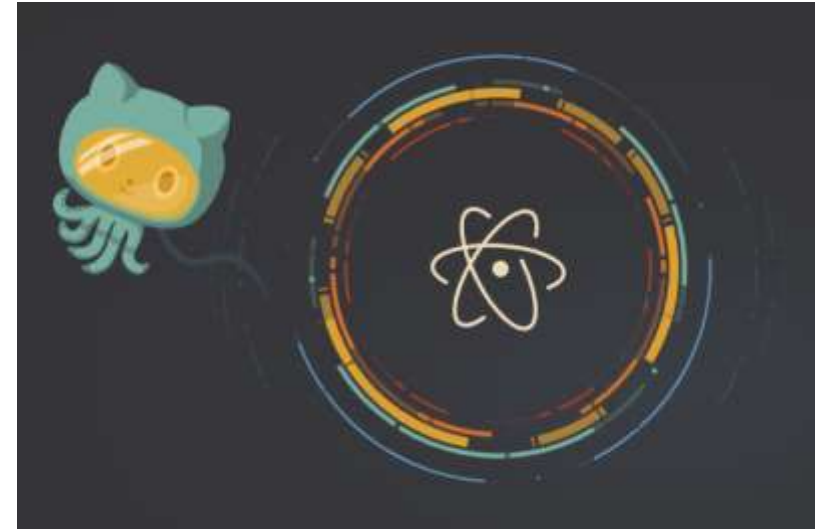


2. INSTALAÇÃO



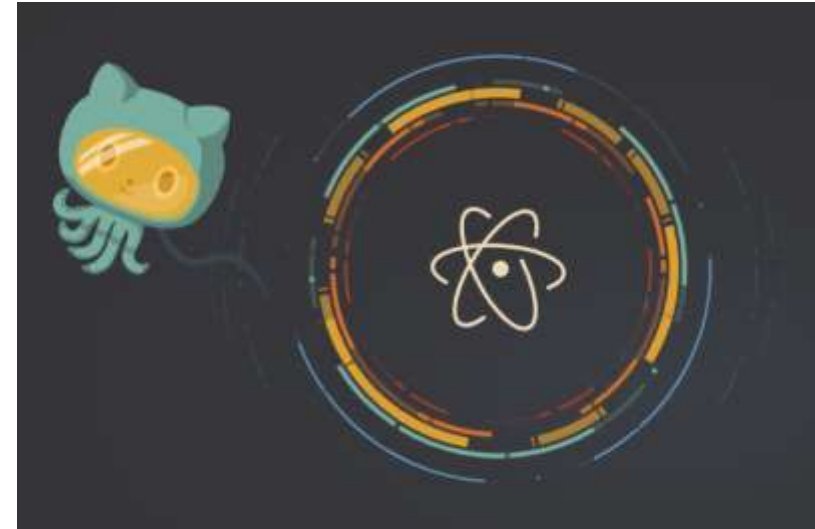
Instalando o Atom

- **Atom é um editor** de texto de **código aberto**
- Linux, macOS e Windows, desenvolvido pelo GitHub sob a licença MIT.
- Moderno
- Ampla comunidade
- Diversos pacotes e funcionalidades
- Versão beta em 25 de junho de 2015.



Instalando o Atom

- <https://atom.io/>
- Windows
- Siga as instruções e aperte em avançar em todas as etapas



Instalando o GIT

- <https://git-scm.com/>
- Windows, Linux, MacOS
- Siga as instruções e aperte em avançar em todas as etapas



CONFIGURAÇÕES BÁSICAS



Onde as configurações são armazenadas?

- Sistema (ou globais)
Path de instalação\etc\gitconfig
- Usuário
Diretório de usuário\.gitconfig
- Projeto
Diretório do projeto\.git\config

Configurações Básicas

- Sistema
git config --system
- Usuário
git config --global
- Projeto
git config

Configurações Básicas

Abra a linha de comando:

1. Configure seu nome de usuário:
`git config --global user.name "FIRST_NAME LAST_NAME"`
2. Configure seu nome de email:
`git config --global user.email "MY_NAME@example.com"`
3. Configure seu editor de texto:
`git config --global core.editor vim`
`git config --global core.editor "c:\users\<nome-do-usuário>\AppData
\Local\atom\atom.exe"`

Testando suas configurações

- Vamos checar nossas informações pessoais?

```
git config --list
```

GIT

HELP



mentorama.

Git Help

- Mostra informações de ajuda sobre o git
- Abra a linha de comando e digite
 - git help

3. COMEÇANDO COM O GIT



Os três estados do Git

1. Você modifica arquivos no seu diretório de trabalho (working).
2. Você prepara os arquivos, adicionando imagens (snapshots) deles à sua área de preparo (staging index).
3. Você faz **commit**, que leva os arquivos como eles estão na área de preparo, e armazena essas imagens de forma permanente para o diretório do Git (repository).

Começando com o Git

- Criar um diretório
- Entrar no diretório
- Iniciar o Git Bash
- Inicialize o **diretório** local como um **repositório Git**
git init



Começando com o Git

- Criar um arquivo no repositório
touch README.md
- Verificar o status atual dos arquivos
git status
- Adicionar o arquivo
git add README.md
git status

Seu primeiro commit

- Execute um “commit” com as mudanças para o repositório com uma mensagem

```
git commit -m “minha mensagem”
```

```
git status
```

MENSAGEM COMMIT

mentorama.



Escrevendo uma mensagem commit

- Como escrever boas mensagens de commit?
 - Seja objetivo
 - Menos que 50 palavras
 - Menos que 72 palavras (considere os mais diversos dispositivos)
 - Escreva sempre no presente e não no passado ou futuro.
 - Escreva informações relevantes para identificação

Visualizando um commit log

git log

git help log

git log --since=2020-01-01

git log --since=2022-01-01

git log --until=2020-01-01

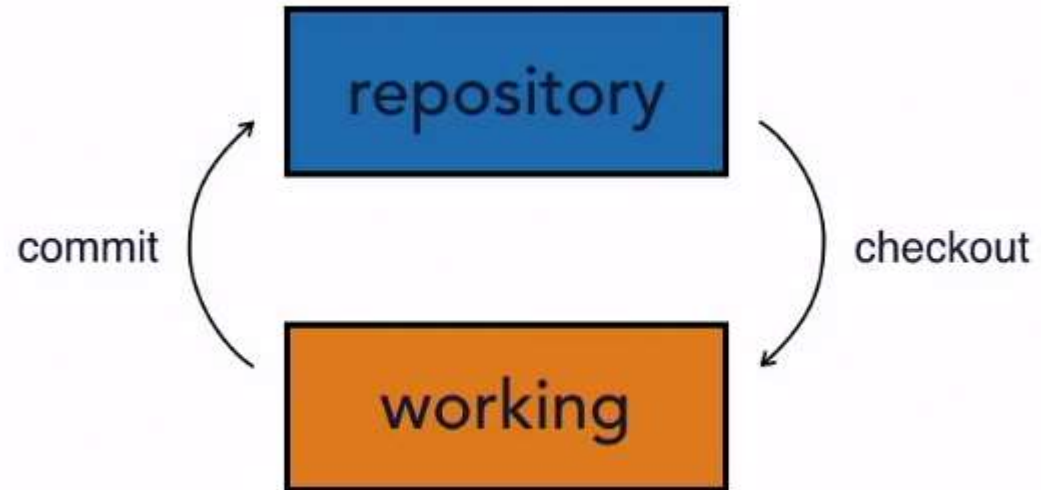
git log --author="feli"

git log --grep="Init"

4. CONCEITOS E ARQUITETURA GIT



Modelo SVN



Modelo GIT

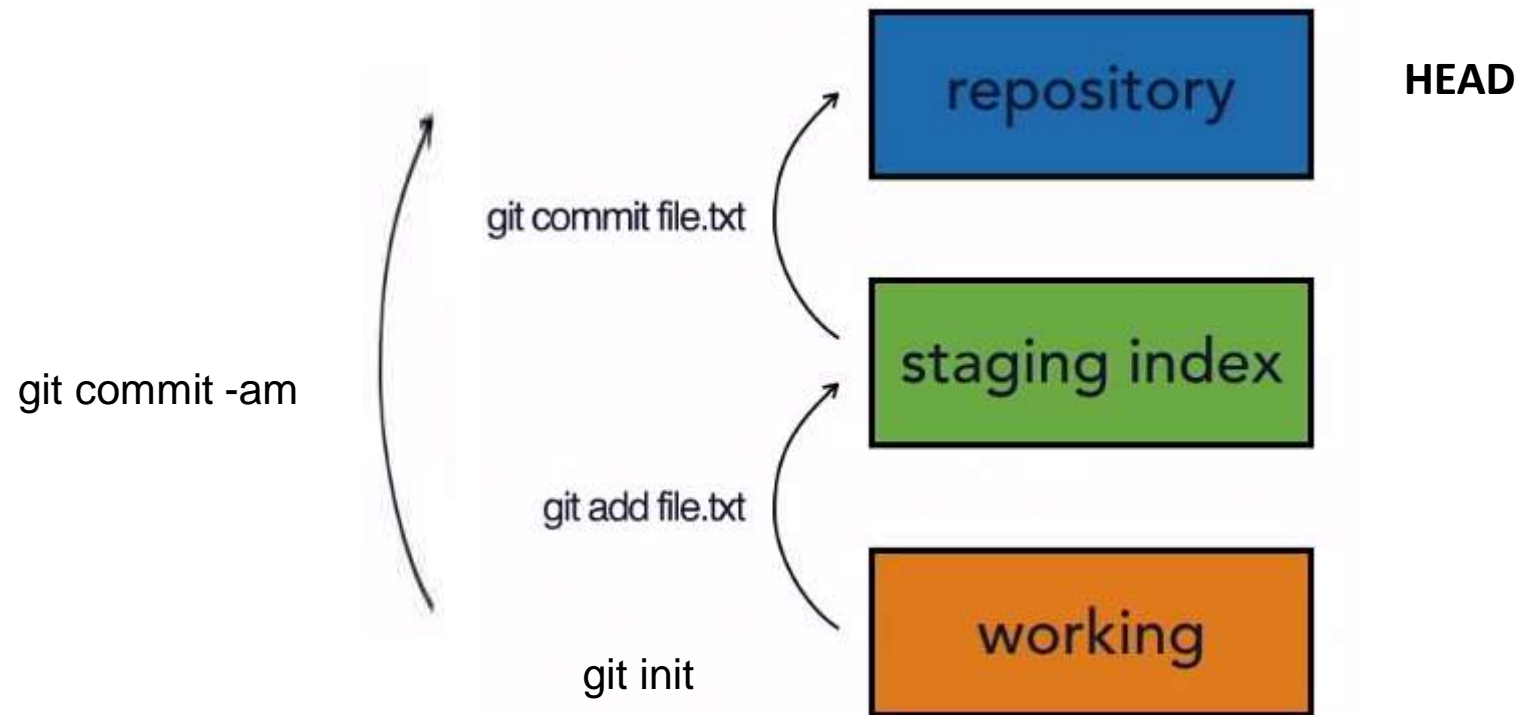
Os três estados:

repository

staging index

working

Modelo Git



Workflow

- Controlar a sequencia dos eventos como peça do trabalho.
- Não há uma solução fixa, mas um fluxo adequado envolve o bom uso de ferramentas, processos e pessoas.
- Flexibilidade



Workflow

Algumas perguntas:

- O fluxo de trabalho se adapta ao tamanho da equipe?
- É fácil desfazer erros com este fluxo de trabalho?
- O fluxo de trabalho impõe alguma nova sobrecarga cognitiva desnecessária à equipe?



5. ALTERANDO OS ARQUIVOS



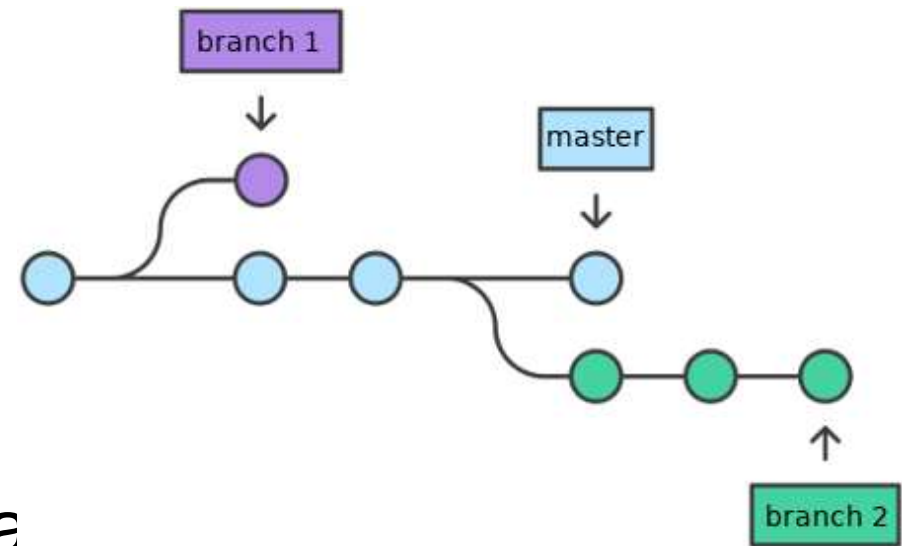
CRIANDO BRANCHES



mentorama.

Branches

- Branches ("ramos") são utilizados para desenvolver funcionalidades isoladas umas das outras.
- O branch master é o branch "padrão" quando você cria um repositório.
- Podemos usar outras branches para desenvolver e mesclar (merge) ao branch master após a conclusão.



Branches

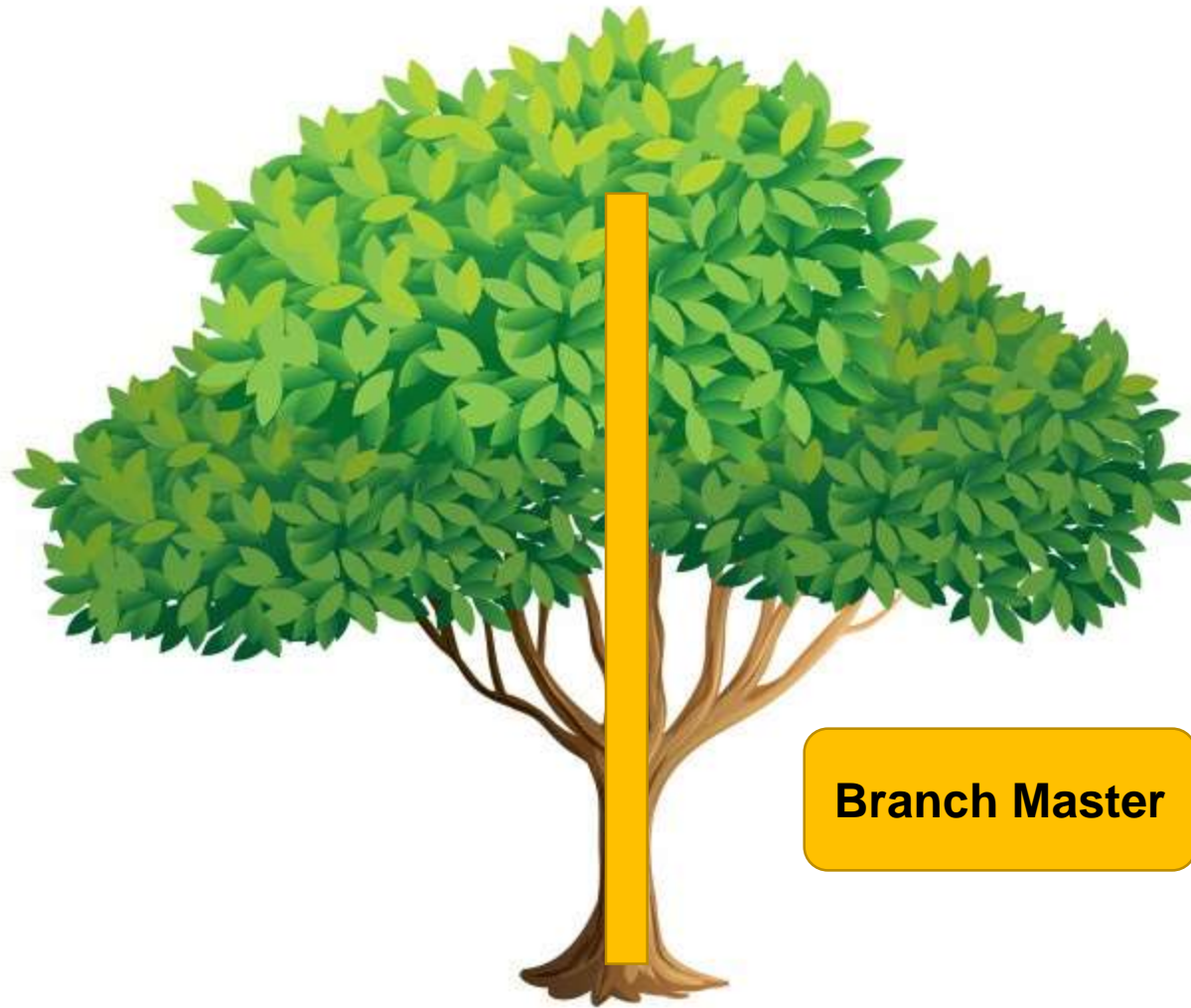
- Branches de longa duração e branches temáticas
- Projetos com branches principais mais estáveis
- Projetos com branches com features diversas
- Criação de branches temporárias para implementações pontuais, criação de tópicos para resolução dos problemas

mentorama.

Branches

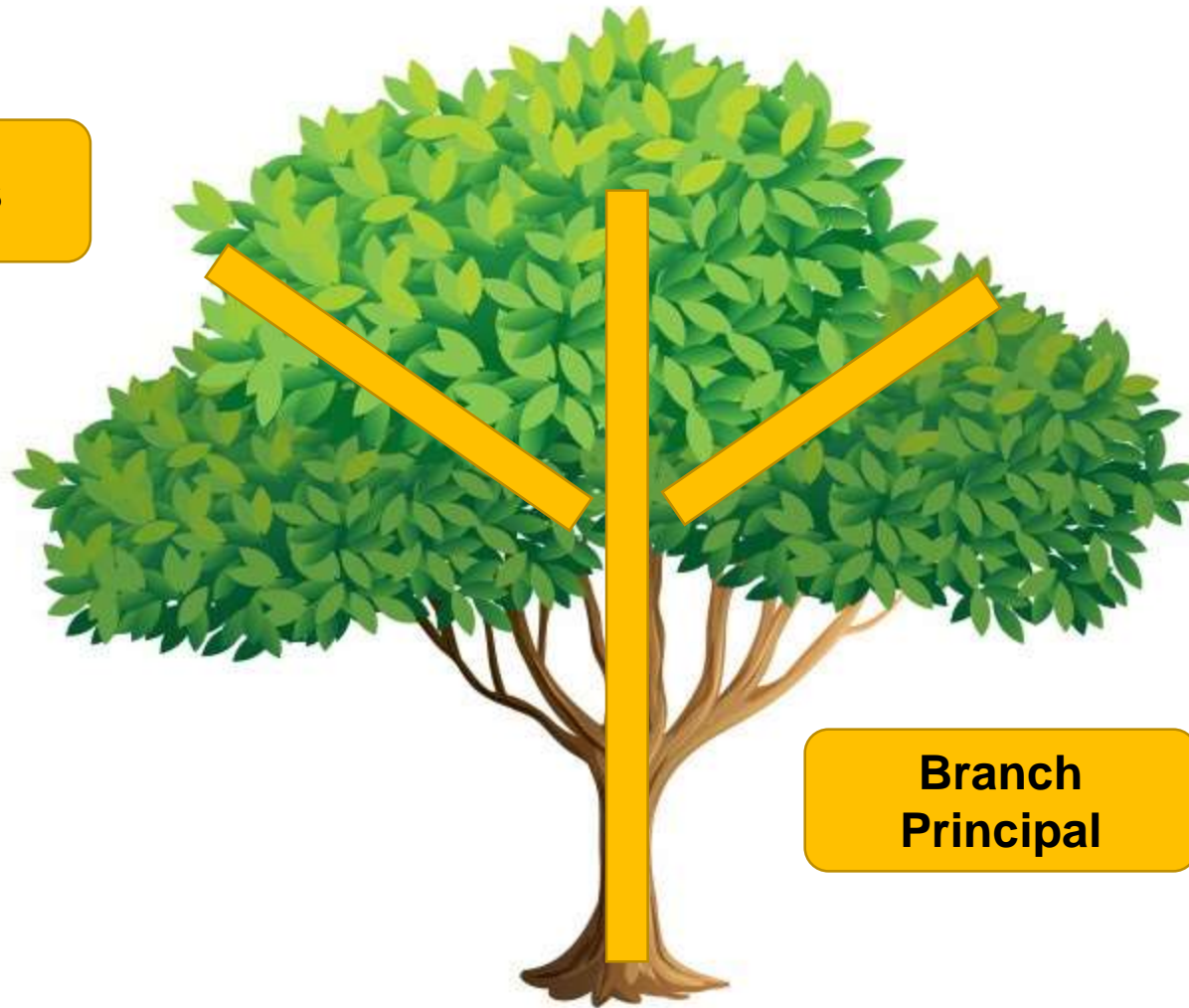


Branches



Branches

Branches



Branch
Principal

Criando um branch

`git branch <name>`

`git checkout -b <name>`

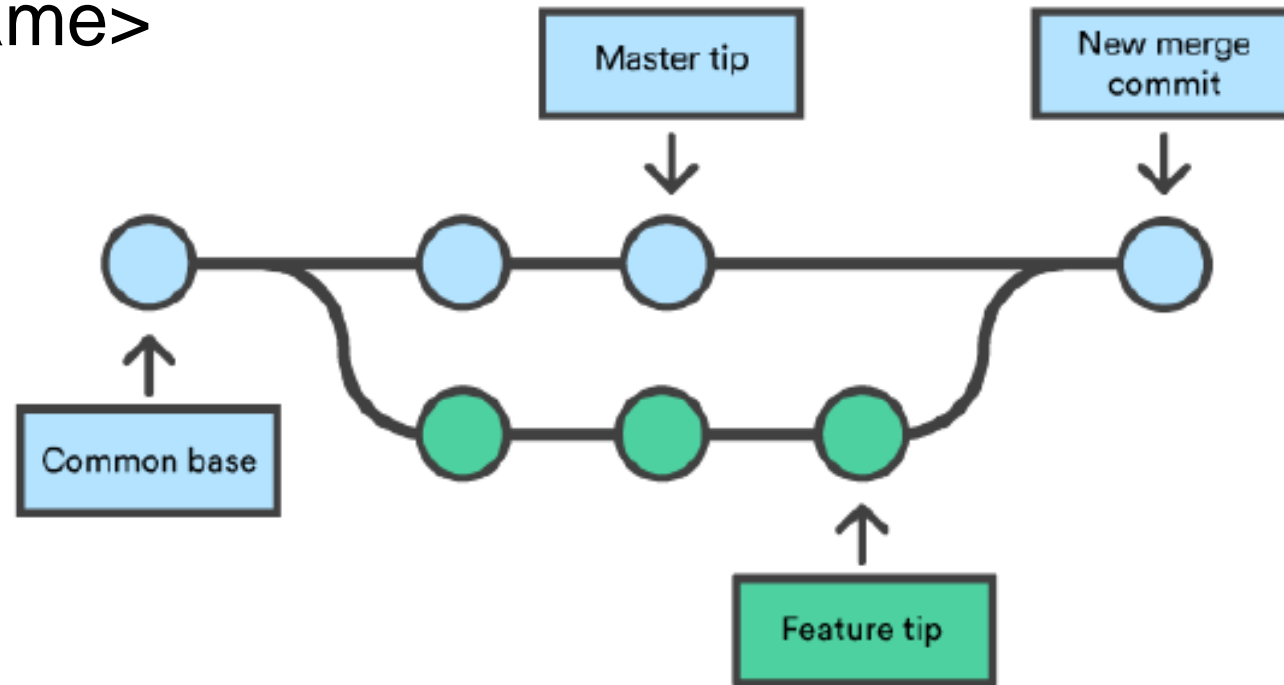
`git status` (avisa que estou no branch)



Git merge

git checkout master

git merge <branch name>



Excluindo um branch

```
git branch -d <branch name>
```

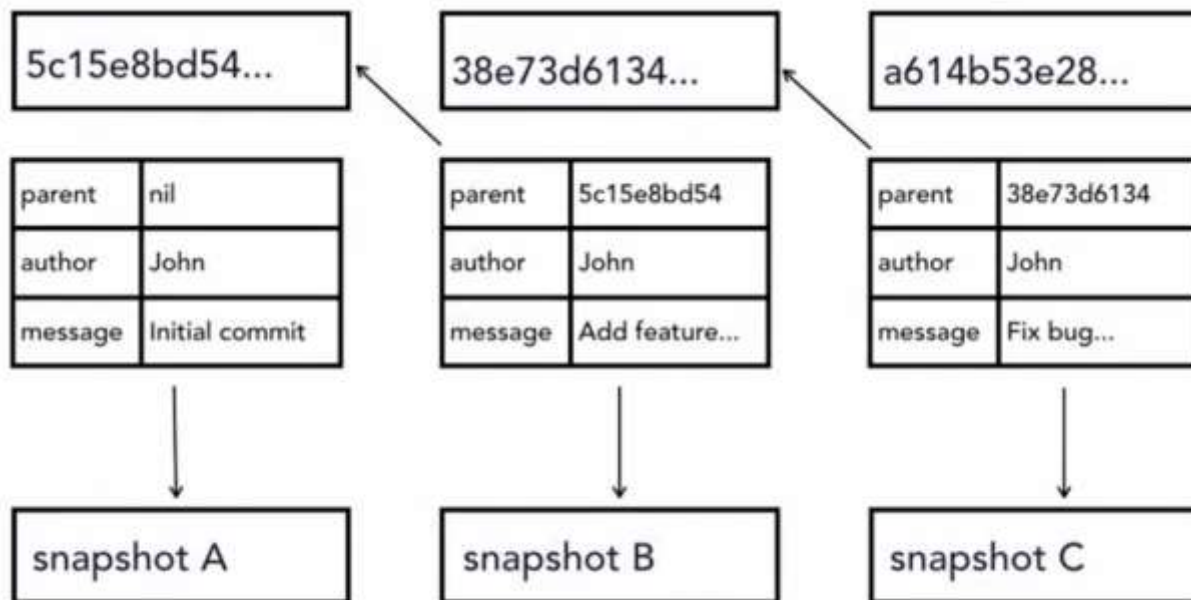
Alterações e remoções diversas

Comandos como:

```
git checkout <hash-number>
```

Valores SHA

- Podemos chamar o git log e visualizar alguns commits



O ponteiro HEAD

- O HEAD (ponteiro) sempre aponta para a ponta de nosso branch atual no repositório.
- Você pode imaginar o HEAD como o “branch atual comprometido”.

`cat .git/head`

[mentorama.](#)



TAGS E RELEASES

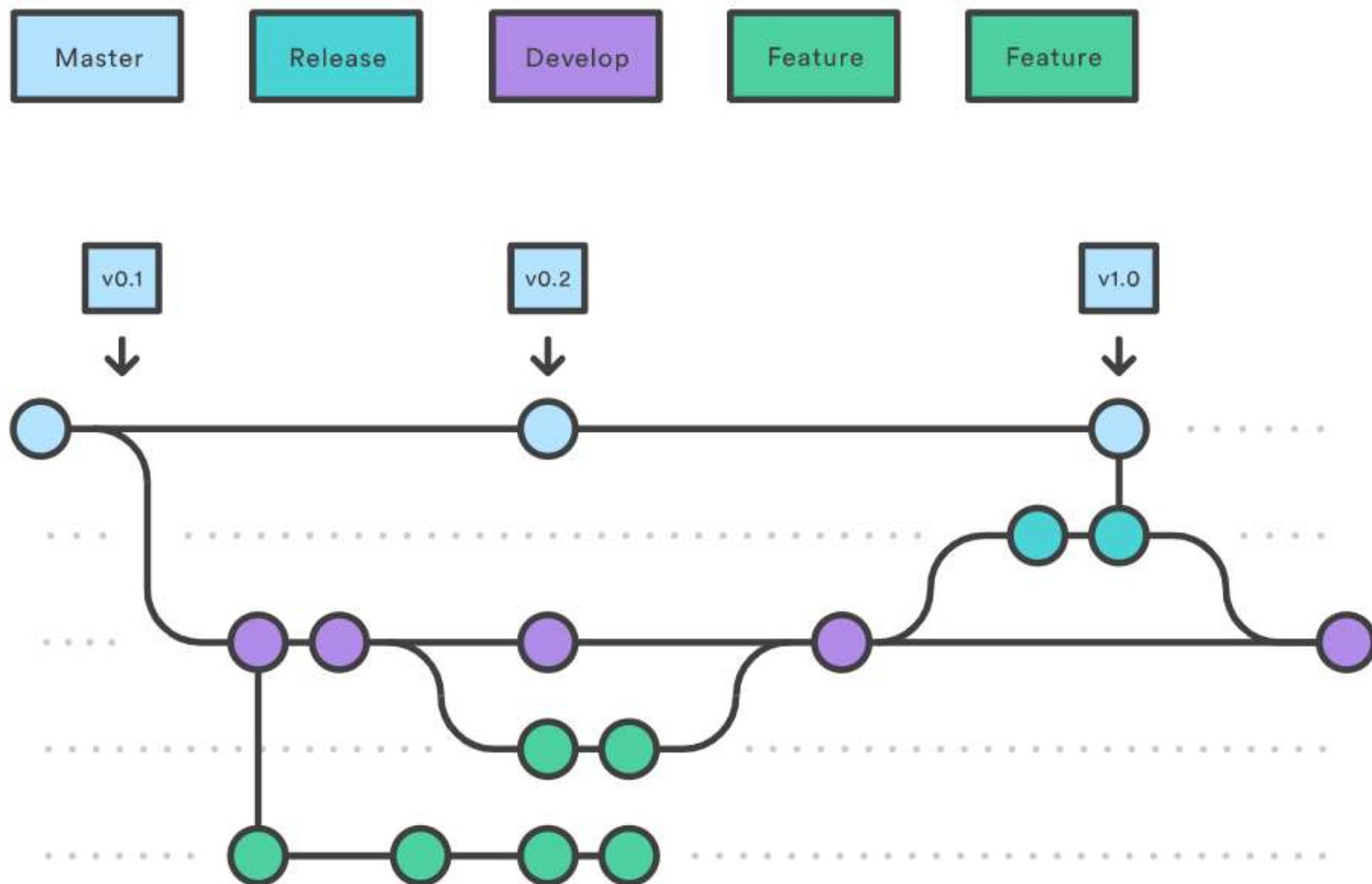


mentorama.

Tags e releases



mentorama.



Criando tags do tipo leve

`git tag <nome da tag>`




Criando tags anotadas

`git tag -a v1.4 -m "minha versão 1.4"`



Criando tags posteriormemente

```
$ git log --pretty=oneline
15027957951b64cf874c3557a0f3547bd83b3ff6 Merge branch 'experiment'
a6b4c97498bd301d84096da251c98a07c7723e65 beginning write support
0d52aaab4479697da7686c15f77a3d64d9165190 one more thing
6d52a271eda8725415634dd79daabbc4d9b6008e Merge branch 'experiment'
0b7434d86859cc7b8c3d5e1dddfed66ff742fcbc added a commit function
4682c3261057305bdd616e23b64b0857d832627b added a todo file
166ae0c4d3f420721acbb115cc33848dfcc2121a started write support
9fceb02d0ae598e95dc970b74767f19372d61af8 updated rakefile
964f16d36dfccde844893cac5b347e7b3d44abbc commit the todo
8a5cbc430f1a9c3d00faaeffd07798508422908a updated readme
```



`git tag -a v1.2 9fceb02`

[mentorama.](#)

COMPARANDO ARQUIVOS



Alterando arquivos

- Vamos alterar um arquivo já enviado para o Stage (index)

`git status`

`git add <nome do arquivo>`

Vamos comparar as diferenças?

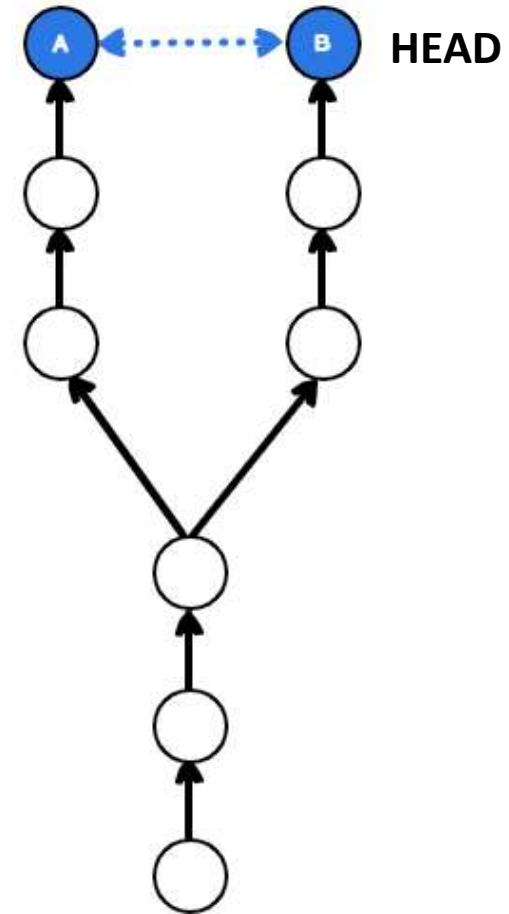
- Compara as diferenças em relação as modificações no arquivo

`git diff <nome do arquivo>`

Vamos comparar as diferenças?

- Compara as diferenças entre arquivos de duas branches

`git diff branch1..branch2`



IGNORE FILES



mentorama.

Usando o arquivo .gitignore

- O objetivo dos arquivos .gitignore é garantir que certos arquivos não rastreados pelo Git permaneçam não rastreados.
- Para parar de rastrear um arquivo que está sendo rastreado, use
`git rm --cached <nome do arquivo> .`

Usando o arquivo .gitignore

- Um arquivo .gitignore especifica arquivos intencionalmente não rastreados que o Git deve ignorar.
- Os arquivos já rastreados pelo Git não são afetados;
- Cada linha em um arquivo .gitignore especifica um padrão.

Padrões .gitignore

- Exemplos:
Fotos-[0-9]-*.jpg
*.jar
[Tt]mp/
bin/



Como não ignorar uma pasta vazia?

- Por padrão o Git ignora pastas vazias
- Como não ignorar?
 - Crie um arquivo com o nome “.gitkeep”
 - Confira com “git status” o reconhecimento da pasta na linha de comando
 - Ou crie o arquivo com o seguinte comando:
touch temp\.gitkeep

6. REPOSITÓRIO REMOTO COM GITHUB



mentorama.

Diferenças entre Git e Github



git



github
SOCIAL CODING

Vamos relembrar nosso workflow?

remoto

repository

staging index

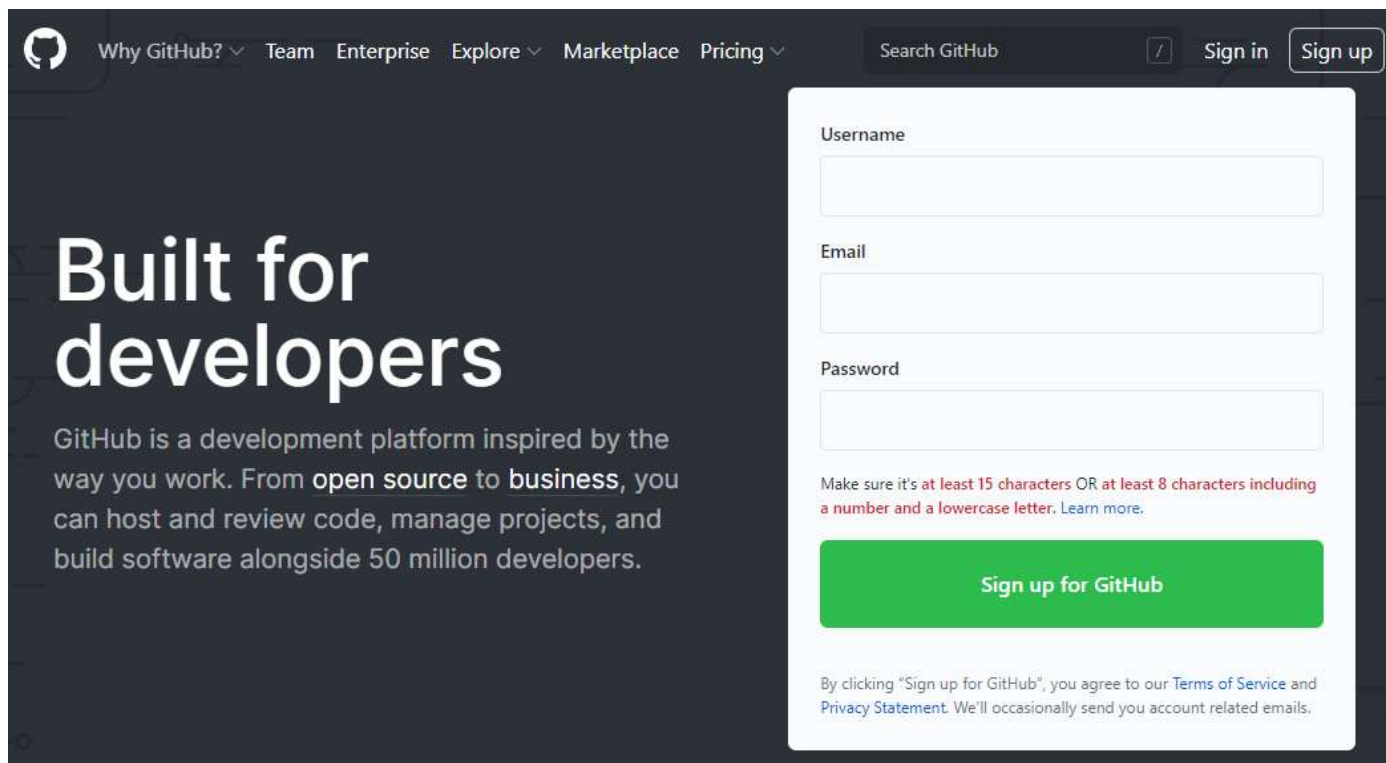
working

Comunicação remota com Github

1. Criar um repositório local e fazer um push no Github
2. Criar um repositório no Github e clonar para o repositório local



Criando uma conta no Github



The image shows the GitHub sign-up page. On the left, there's a dark sidebar with the GitHub logo and navigation links: 'Why GitHub?', 'Team', 'Enterprise', 'Explore', 'Marketplace', and 'Pricing'. The main content area has a dark background with the text 'Built for developers' and a description of GitHub as a development platform. On the right, there's a white sign-up form with fields for 'Username', 'Email', and 'Password'. Below the password field, there's a note about password requirements: 'Make sure it's at least 15 characters OR at least 8 characters including a number and a lowercase letter. Learn more.' A green button labeled 'Sign up for GitHub' is at the bottom of the form. At the very bottom, there's a small disclaimer: 'By clicking "Sign up for GitHub", you agree to our Terms of Service and Privacy Statement. We'll occasionally send you account related emails.'

Why GitHub? Team Enterprise Explore Marketplace Pricing Search GitHub / Sign in Sign up

Built for developers

GitHub is a development platform inspired by the way you work. From open source to business, you can host and review code, manage projects, and build software alongside 50 million developers.

Username

Email

Password

Make sure it's at least 15 characters OR at least 8 characters including a number and a lowercase letter. [Learn more.](#)

Sign up for GitHub

By clicking "Sign up for GitHub", you agree to our [Terms of Service](#) and [Privacy Statement](#). We'll occasionally send you account related emails.

mentorama.

Criando uma conta no Github

Create your account

Username *

Email address *

Password *

Note (or if): at least 15 characters OR at least 8 characters including a number and a lowercase letter. [Learn more](#).

Email preferences

☐ Send me occasional product updates, announcements, and offers.

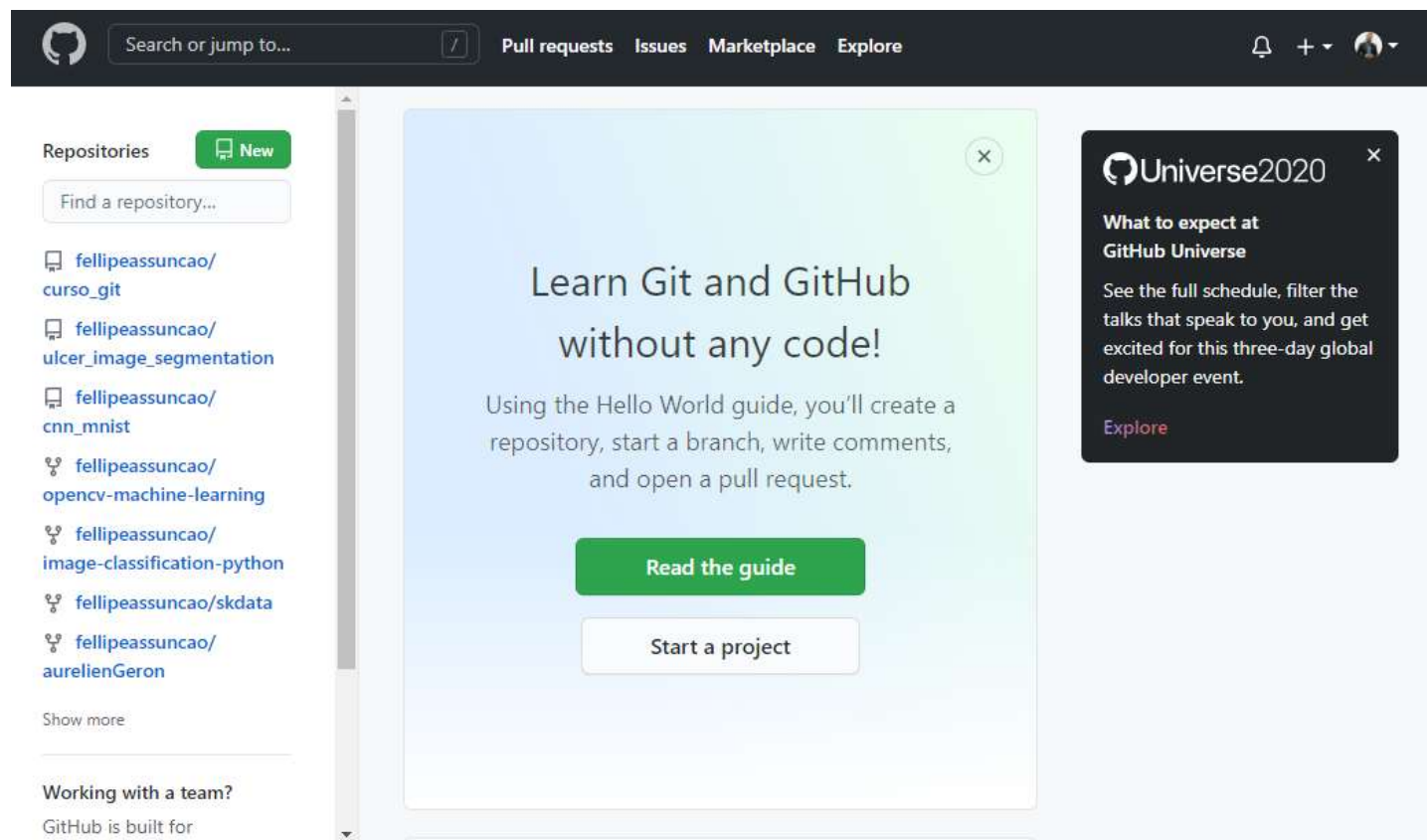
Verify your account

Resolva este enigma para sabermos que
você é uma pessoa de verdade.

Verificar

Create account

Criando uma conta no Github



Criando um novo repositório

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner * / Repository name *

Great repository names are short. cursoGit is available. Need inspiration? How about [supreme-rotary-phone?](#)

Description (optional)

- ☒  **Public**
Anyone on the internet can see this repository. You choose who can commit.
- ☐  **Private**
You choose who can see and commit to this repository.

Initialize this repository with:

Skip this step if you're importing an existing repository.

- ☐ **Add a README file**
This is where you can write a long description for your project. [Learn more.](#)
- ☐ **Add .gitignore**
Choose which files not to track from a list of templates. [Learn more.](#)
- ☐ **Choose a license**
A license tells others what they can and can't do with your code. [Learn more.](#)

Create repository

mentorama.

Criando um novo repositório

[Code](#) [Issues](#) [Pull requests](#) [Actions](#) [Projects](#) [Wiki](#) [Security](#) [Insights](#) [Settings](#)

Quick setup — if you've done this kind of thing before

[Set up in Desktop](#) or [HTTPS](#) [SSH](#)

Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

...or create a new repository on the command line

```
echo "# cursoGit" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/fellipeassuncao/cursoGit.git
git push -u origin main
```

...or push an existing repository from the command line

```
git remote add origin https://github.com/fellipeassuncao/cursoGit.git
git branch -M main
git push -u origin main
```

Criando uma conexão remota

- Isso cria uma conexão remota chamada “origin” apontando para o seu repositório GitHub que você acabou de criar.


```
git remote add origin <endereço Github>
```

```
git remote
```

Enviando os arquivos remotamente

<> Code ⓘ Issues 🔗 Pull requests ▶ Actions 📁 Projects 📖 Wiki 🛡 Security 📈 Insights ⚙ Settings

Quick setup — if you've done this kind of thing before


 Set up in Desktop

 or

HTTPS

SSH


https://github.com/fellipeassuncao/cursoGit.git



Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).


...or create a new repository on the command line

```
echo "# cursoGit" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/fellipeassuncao/cursoGit.git
git push -u origin main
```



...or push an existing repository from the command line

```
git remote add origin https://github.com/fellipeassuncao/cursoGit.git
git branch -M main
git push -u origin main
```



Como puxar arquivos?

- Crie um arquivo no Github
- Retorne a linha de comando e digite

ls

git pull origin

Como clonar arquivos?

- Crie uma nova pasta dentro do seu computador
- Clone os arquivos do Github para dentro dessa pasta

`git clone <endereço do repositório remoto>`

EXERCICIOS



Exercícios

- ETAPA 1 >> ETAPA 2 >> ETAPA 3
- Relatório de entrega com os prints de cada uma das etapas e observações que você julgar necessário (logs, etc).

CONCLUSÃO E PRÓXIMOS PASSOS

