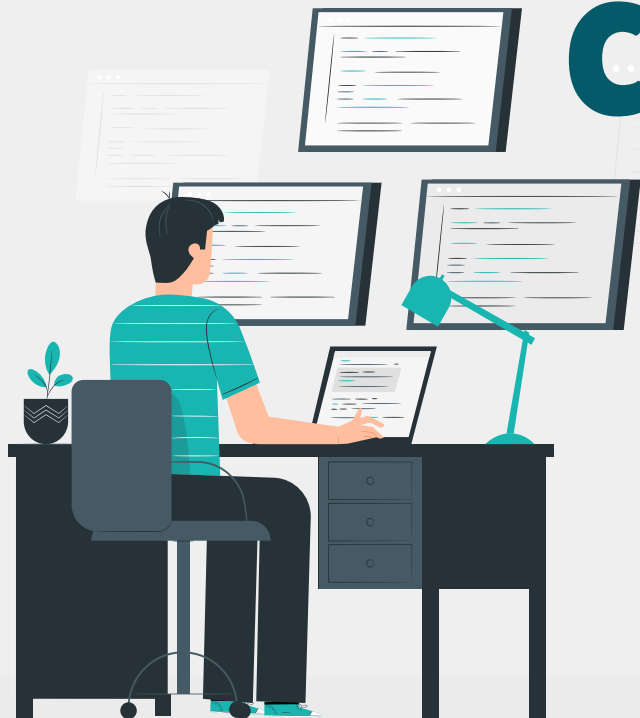


# Controle de versão com Git



Leila Rodrigues  
Engenharia de Software  
UFC - 2020.1

# "FINAL".doc



FINAL.doc!



FINAL\_rev.2.doc



FINAL\_rev.6.COMMENTS.doc



FINAL\_rev.8.comments5.  
CORRECTIONS.doc



FINAL\_rev.18.comments7.  
corrections9.MORE.30.doc



FINAL\_rev.22.comments49.  
corrections.10.##\$%WHYDID  
ICOMETOGRADSCHOOL?????.doc



## O que é controle de versão?

# O que é controle de versão?

O controle de versão é um sistema que registra alterações em um arquivo ou conjunto de arquivos ao longo do tempo, para que você possa recuperar versões específicas mais tarde.



**Desenvolvedor PHP Junior**  
Deloitte Brasil · [São Paulo, São Paulo, Brasil](#)  
Anunciada há 3 semanas · 713 visualizações

[Salvar](#) [Candidatura simplificada](#)

**Experiência Profissional Anterior:**

- Experiência em implementação de arquitetura de microserviço
- PHP
- Banco de dados SQL Server / Mongo DB
- **GIT (GitLab/GitHub)**
- Framework Yii
- Conhecimentos em Node.js
- Desenvolvimento Android (Java Nativo)

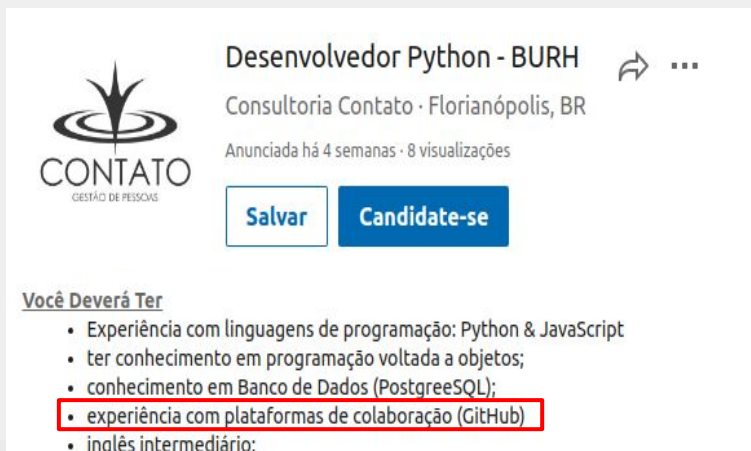


**Full Stack Developer (React Native)**  
Serasa · [São Paulo, São Paulo](#)  
Anunciada há 1 semana · 487 visualizações

[Salvar](#) [Candidate-se](#)

**Requisitos e qualificações:**

- Domínio em React/React Native
- Domínio em Redux
- Domínio em APIs REST
- **Domínio em Git**
- Conhecimento em Programação orientada a objetos
- Experiência com aplicativos mobile (híbrido ou nativo)
- Conhecimento em Metodologia ágil
- Conhecimento em Testes unitários
- Conhecimento em Padrões de projeto



**Desenvolvedor Python - BURH**  
Consultoria Contato · Florianópolis, BR  
Anunciada há 4 semanas · 8 visualizações

[Salvar](#) [Candidate-se](#)

**Você Deverá Ter**

- Experiência com linguagens de programação: Python & JavaScript
- ter conhecimento em programação voltada a objetos;
- conhecimento em Banco de Dados (PostgreSQL);
- **experiência com plataformas de colaboração (Github)**
- inglês intermediário;



**Cientista de Dados**  
Cielo · [Barueri, BR](#)  
Anunciada há 2 meses · 798 visualizações

[Salvar](#) [Candidate-se](#)

**Diferencial**

- **Ter publicações/portfolio no Github;**
- ter participado de **competições**, como por exemplo no **Kaggle**;

# Sistema de Gerenciamento de Versões

- Soluções Comerciais
  - SourceSafe
  - TFS
  - ClearCase
- Soluções Open-source
  - Concurrent Version System (CVS)
  - Subversion (SVN)
  - Git
  - Mercurial

# Por que usar?



Armazena histórico

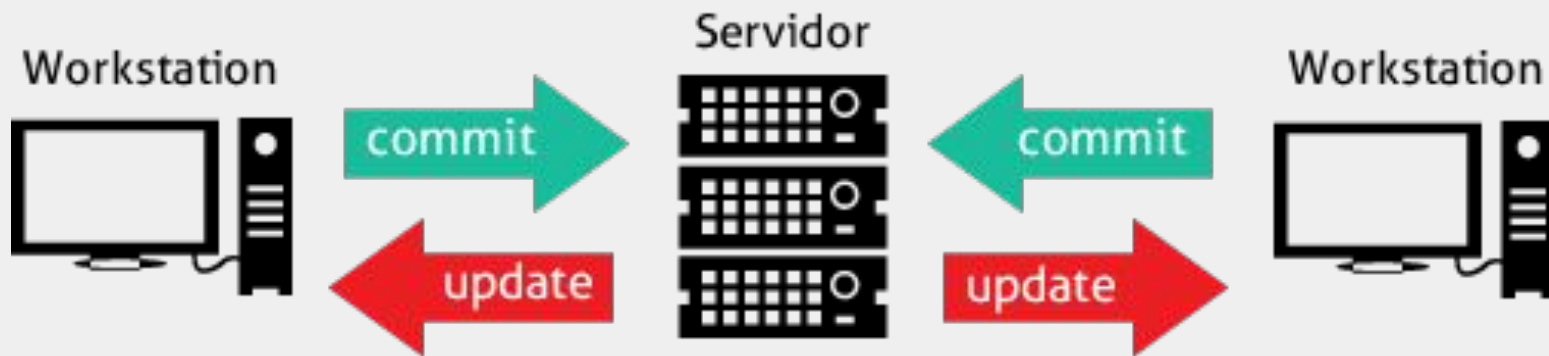


Desenvolver versões diferentes em paralelo (ramificações)



Desenvolvedores trabalhando no mesmo código

# Controle de Versão Centralizado

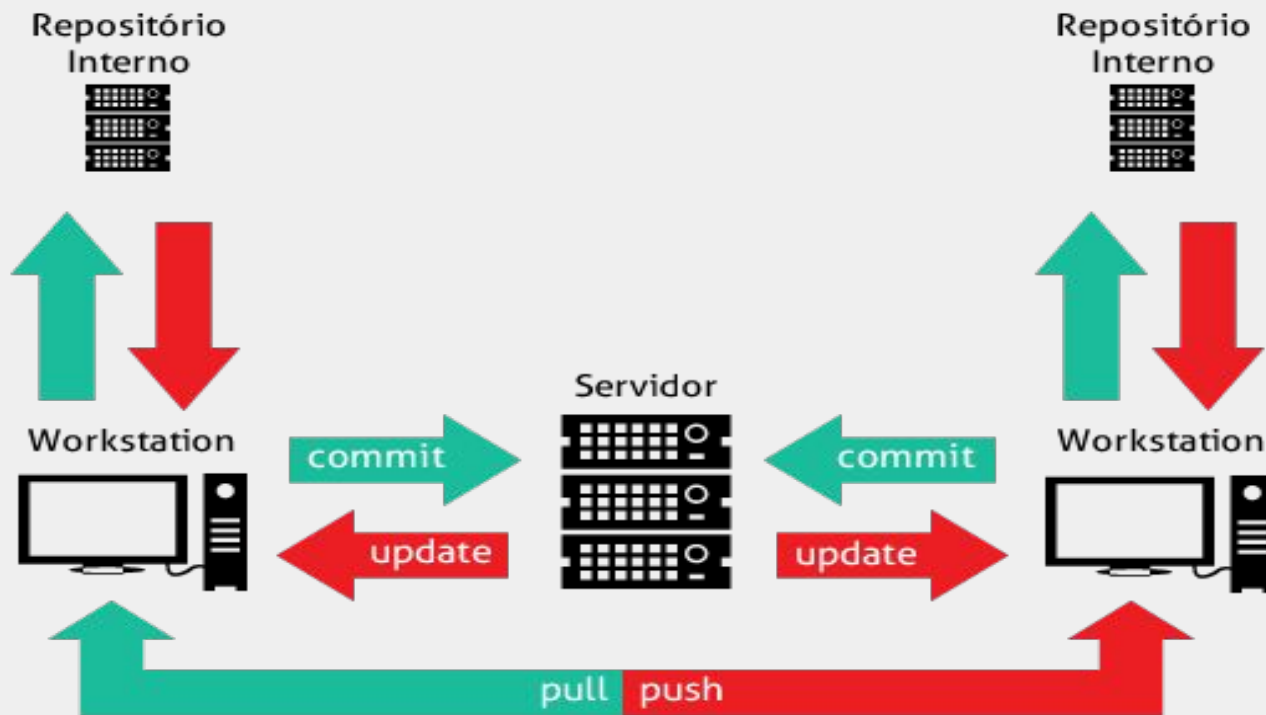


Commit: envia o conjunto de arquivos alterados ao servidor, gerando um novo histórico de atualização.

Update: processo inverso do commit. Envia a última versão do servidor para o repositório local.



# Controle de Versão Distribuído



Pull: Atualiza o repositório local (destino) com todas as alterações feitas em outro repositório (origem).

Push: Envia as alterações do repositório local (origem) para um outro repositório (destino).

# Contexto histórico

- O BitKeeper era utilizado por Linus para gerenciar o código de suas versões do Linux.
- Ocorreu um problema com a licença do BitKeeper.
- 05/04/2005: 1ª versão do Git
- 15/06/2005: Git é usado para controle de versão do kernel do Linux.

# Objetivos iniciais do Git

- Rapidez
- Design simples
- Forte suporte ao desenvolvimento não linear (milhares de ramos paralelos)
- Totalmente distribuído
- Capaz de lidar com grandes projetos como o kernel Linux com eficiência (velocidade e tamanho dos dados)

# Ferramentas populares



GitHub



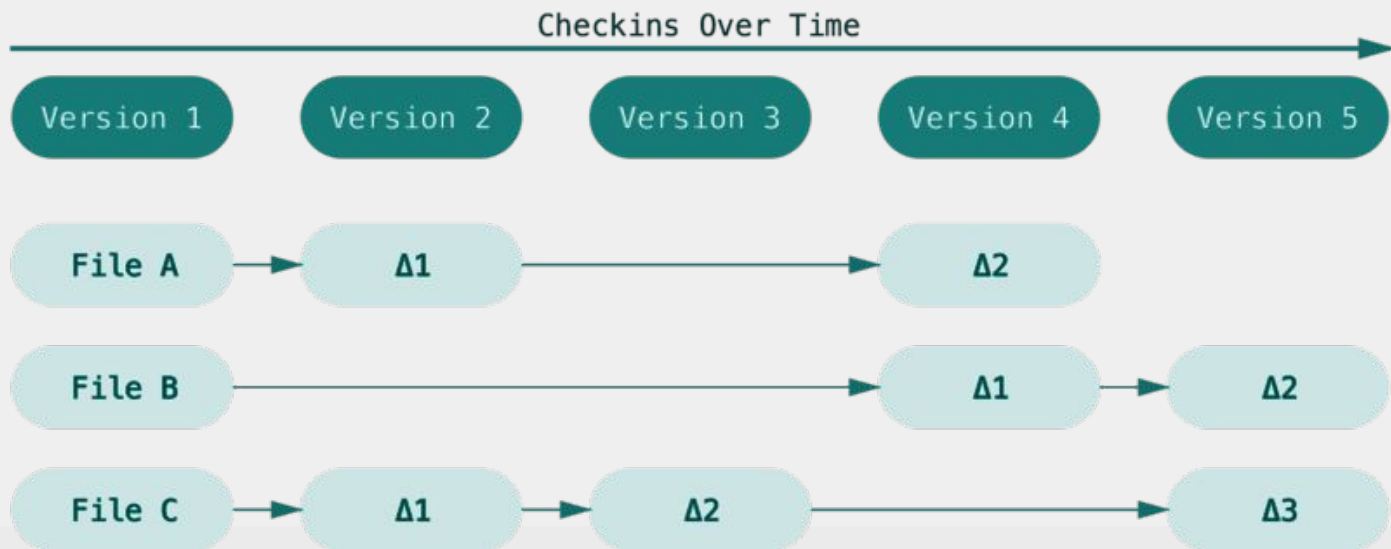
Gitlab



Bitbucket

# Snapshots

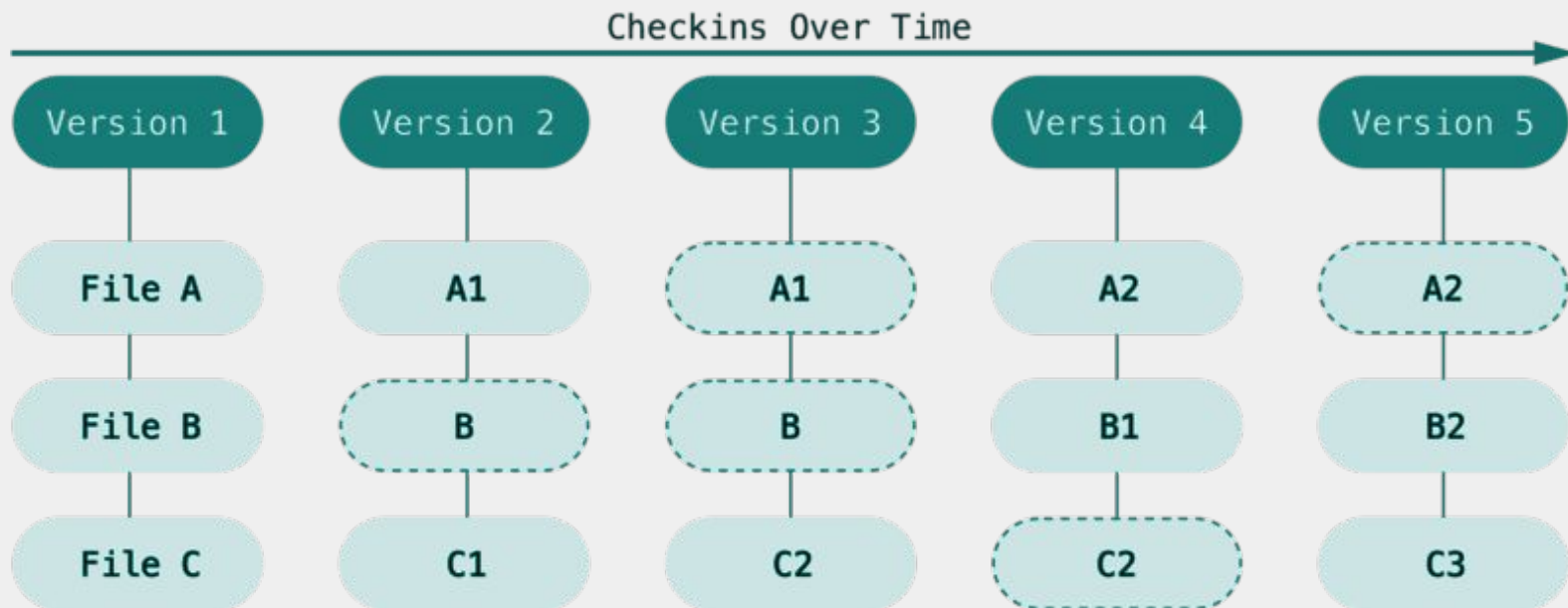
- Outros sistemas armazenam as informações como uma lista de alterações baseadas em arquivos.
- Controle de versão baseado em delta.



# Snapshots

- O Git registra “momentos”, os quais são chamados de snapshots, contendo uma espécie foto dos arquivos.
- A cada commit, o Git basicamente tira uma foto da aparência de todos os seus arquivos naquele momento e armazena uma referência a essa captura.
- O Git pensa em seus dados mais como um fluxo de snapshots.

# Snapshots





# Operações locais

- Quase todas operações do Git são feitas de forma local.
- Você tem todo o histórico do projeto no seu disco local, a maioria das operações parece quase instantânea.
- Isso significa que há muito pouco que você não pode fazer se estiver offline.

# Integridade

- A integridade das informações de cada projeto são asseguradas por um checksum (chave de verificação de integridade).
- Tudo no Git é referido por checksum.
- Isso significa que é impossível alterar o conteúdo de qualquer arquivo ou diretório sem o Git saber disso.

# Integridade

- Para realizar checksum, o Git usa uma implementação de hash SHA-1.

```
24b9da6552252987aa493b52f8696cd6d3b00373
```

- Git armazena tudo em seu banco de dados não pelo nome do arquivo, mas pelo valor de hash do seu conteúdo.

# Três estados

- **Modificado**: alterou arquivo, mas ainda não foi confirmado no banco de dados.
- **Preparado**: marcou um arquivo modificado na versão atual para entrar no próximo snapshot de confirmação.
- **Confirmado**: os dados foram armazenado com segurança no banco de dados local.

# Armazenamento

- Pasta .git
  - Apenas no diretório raiz do projeto.
  - Contém todos os objetos, commits e configurações do projeto.
  - .git/config: arquivo com configurações específicas do repositório.
- .gitignore
  - Arquivo texto que indica os arquivos que devem ser ignorados

# Instalação

- Linux
  - Para Debian, Ubuntu e derivados:
    - **apt-get install git**
- Windows
  - <https://gitforwindows.org>

# Configurações iniciais

- O git config é a ferramenta do Git usada para configurações.
- Um dos passos iniciais é identificar o nome de usuário e e-mail para o git.

```
$ git config --global user.name "seunome"
```

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

# Configurações iniciais

- Se desejar que essas informações sejam diferentes para projetos específicos.

```
$ git config --local user.name "seunome"
```

```
$ git config --local user.email "seunome@email.com"
```



# Repositório Git

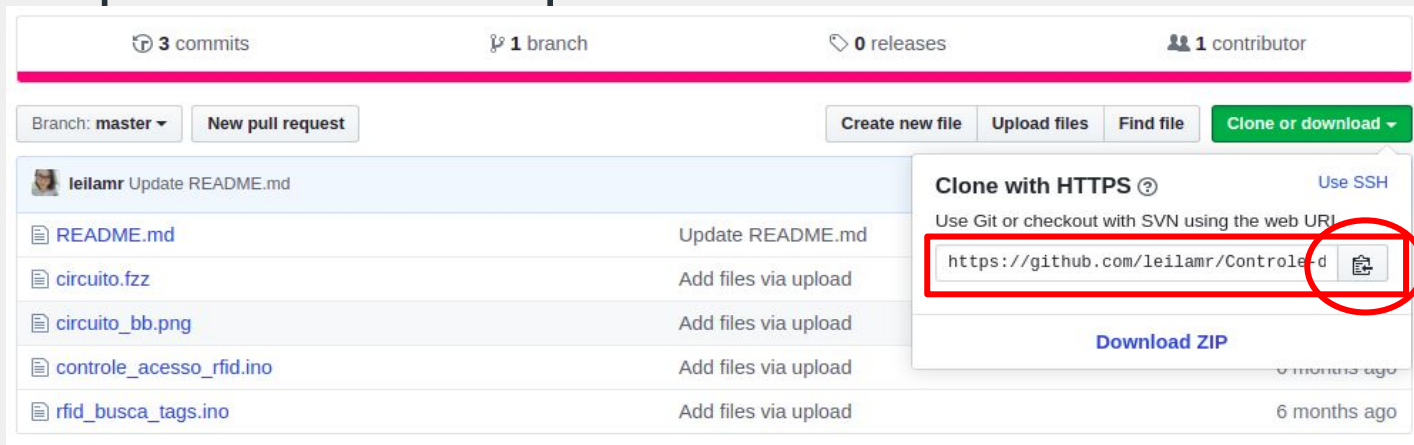
Transformar um diretório local que atualmente não está sob o controle de versão:

- Linux:  
\$ cd /home/user/project  
\$ git init
- Windows:  
\$ cd C:\Users\nomeusuario\Desktop  
\$ git init

# Repositório Git

Clonar um repositório Git existente

- Copie a URL do repositório



- Digite o código abaixo no terminal:  
`$ git clone <url-do-repositório>`

# Adicionar arquivos à lista de commit

- Adicionar um ou mais arquivos:  
\$ git add <nome-arquivo>  
\$ git add \*.extensão  
\$ git add \*
- Verificar alterações  
\$ git status

# Ignorar arquivos à lista de commit

- Aplicar commit apenas arquivos fonte e ignore arquivos derivados
- Por exemplo, ignore executáveis, arquivos gerados durante a compilação e arquivos durante a execução dos testes de código.

# Ignorar arquivos à lista de commit

- Arquivo `.gitignore`

```
# Byte-compiled / optimized / DLL files
__pycache__/
*.py[cod]
*$py.class

# C extensions
*.so

# Distribution / packaging
.Python
build/
develop-eggs/
dist/
downloads/
eggs/
.eggs/
lib/
lib64/
parts/
sdist/
var/
wheels/
*.egg-info/
.installed.cfg
*.egg
MANIFEST
```

```
# ignore all .a files
*.a

# but do track lib.a, even though you're ignoring .a files above
!lib.a

# only ignore the TODO file in the current directory, not subdir/TODD
/TODO

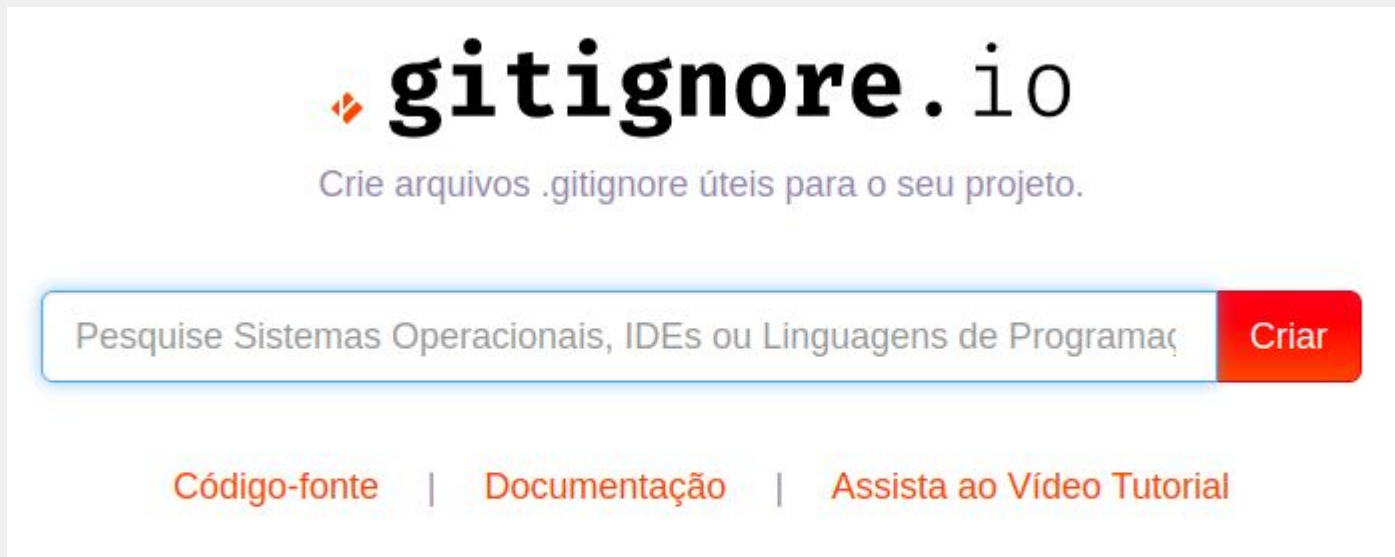
# ignore all files in any directory named build
build/

# ignore doc/notes.txt, but not doc/server/arch.txt
doc/*.txt

# ignore all .pdf files in the doc/ directory and any of its subdirectories
doc/**/*.pdf
```

# Ignorar arquivos à lista de commit

- <https://gitignore.io/>



# Commit

- Confirmar alterações  
\$ git commit  
\$ git commit -m “sua mensagem
- Desfazendo um commit específico  
\$ git revert 58cffab

# Remover arquivos

- Para remover arquivos presentes no seu repositório .git, é preciso removê-lo não só do diretório em que ele se encontra, mas também é removê-lo da área de seleção e então efetuar o commit.

```
$ git rm arquivo.extensão
```

```
$ git commit
```



# Histórico

- Para verificar o histórico dos commits efetuados em determinado repositório

\$ git log

```
leila@leila-rodrigues:~/Documentos/mathematical-morphology$ git log
commit c857f5858c9596fb95d35ec70df6f30745f95355 (HEAD -> master, origin/master,
origin/HEAD)
Author: leila <leilarodrigues@alu.ufc.br>
Date:   Fri Oct 11 23:13:08 2019 -0300
```

Outputs

```
commit 3caad5ed8a85d0812ea779b3cffffa7da7869afe5
Author: leila <leilarodrigues@alu.ufc.br>
Date:   Fri Oct 11 15:29:25 2019 -0300
```

etapa 2

```
commit f7ae55c9bc253ae5fd511ff9552c247c9662f703
Author: leilarodrigues <leilarodrigues@alu.ufc.br>
Date:   Mon Oct 7 11:13:02 2019 -0300
```

Inputs

# Repositórios remotos

- Quando se deseja compartilhar um diretório Git com outras pessoas, é necessário entender e aprender como gerenciar um repositório remoto do Git.
- Para ver quais servidores remotos você configurou, você pode executar o comando:  
`$ git remote`

# Repositórios remotos

- Enviar modificações  
`$ git push origin master`
- Pegar modificações feitas em um repositório  
`$ git fetch origin master` (pega as atualizações e coloca em outro branch)  
`$ git pull origin master` (pega as atualizações e já faz merge)

# Ramificações

- Ramificação significa que você diverge da linha principal de desenvolvimento e continua a trabalhar sem mexer nessa linha principal.
- A ramificação isso cria um novo ponteiro para movimentação no repositório Git.

`$ git branch <nome>`

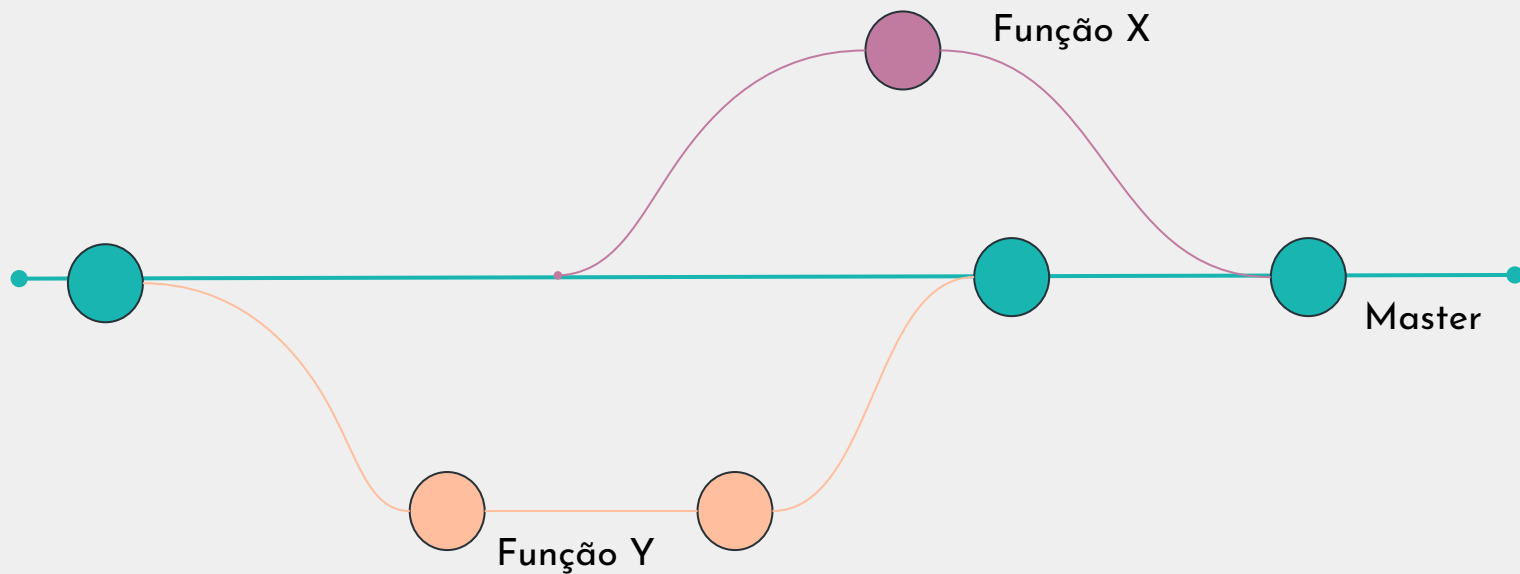
- Alterando a ramificação

`$ git checkout <nome>`

- Visualiza branches

`$ git branch`

# Ramificações



# Merge

- Mesclar a ramificação criada com o ramo master  
\$ git checkout master  
\$ git merge <nome>

# Problemas

- Problemas (bugs) ou sugestões de tarefas para o código?

The screenshot shows a GitHub issue page for the repository 'suhas-nithyanand / Image-Segmentation-using-Region-Growing'. The issue is titled 'amazing job #1' and is in the 'Issues' tab. It was opened by 'iamyourdaddy' on 20 Jun 2017 and has 1 comment. The issue is marked as 'Open'.

Comments:

- ghost** commented on 20 Jun 2017: "can you explain what is find\_region() doing?"
- akkshita** commented on 27 Jun: "Traceback (most recent call last):  
File 'region\_growing.py', line 100, in  
if arr\_out[i][j] > 25:  
IndexError: index 481 is out of bounds for axis 0 with size 481"

On the right side, there are sections for 'Assignees' (No one assigned), 'Labels' (None yet), 'Projects' (None yet), 'Milestone' (No milestone), and 'Notifications' (You're not receiving notifications from this thread). At the bottom right, it says '1 participant' with the user 'akkshita' listed.

At the bottom, there is a 'Write' section with a 'Preview' tab, a text area for 'Leave a comment', and a 'Comment' button.

# Licenças



## Eu quero uma licença simples e permissiva

A **Licença MIT** é uma licença permissiva que é concisa e vai direto ao ponto. Ela permite que as pessoas façam o que quiserem com seu código, desde que forneçam uma atribuição de volta para você e não lhe responsabilize.

**jQuery** e **Rails** usam a Licença MIT.



## Eu estou preocupado com patentes

A **Licença Apache** é uma licença permissiva, similar à Licença MIT, mas que também provê uma concessão expressa de direitos de patente dos contribuintes para os usuários.

**Apache**, **SVN**, e **NuGet** usam a Licença Apache.



## Eu me preocupo em compartilhar melhorias

A **GPL (V2 ou V3)** é uma licença "copyleft" que exige que quem distribui o seu código ou uma obra derivada deve disponibilizar o fonte sob os mesmos termos.

**Linux**, **Git**, e **WordPress** usam GPL.



# README.md

README.md

## Dataset Iris plant

Iris flower classification with MLP using MATLAB.

### Attribute informations

1. sepal length in cm
2. sepal width in cm
3. petal length in cm
4. petal width in cm
5. class: Iris Setosa, Iris Versicolour and Iris Virginica.

### Class coding

- Setosa = [1 0 0]
- Versicolor = [0 1 0]
- Virginica = [0 0 1]

### Network settings:

```
Train = 70%, Validation = 15% and Testing = 15%
Number hidden of nodes = 4
Epochs = 1000
Trainng Function = trainlm
Transfer Function (layer 1) = tansig
Trasnfer Function (layer 2) = purelin

Accuracy = 99.3%
```

# Referências

- Documentação oficial:
  - <https://book.git-scm.com/book/en/v2/>
- Guia rápido:
  - [https://rogerdudler.github.io/git-guide/index.pt\\_BR.html](https://rogerdudler.github.io/git-guide/index.pt_BR.html)
- Guia rápido sobre licenças:
  - <http://escolhaumalicensa.com.br/>



# Indicações

