



Leila Rodrigues
Engenharia de Software
UFC - 2019

"FINAL".doc



FINAL.doc!



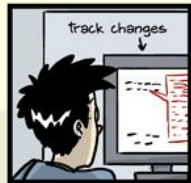
FINAL_rev.2.doc



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



JORGE CHAM © 2012

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.

Sistema de Gerenciamento de Versões

→ Soluções Comerciais

- ◆ SourceSafe
- ◆ TFS
- ◆ ClearCase

→ Soluções Open-source

- ◆ Concurrent Version System (CVS)
- ◆ Subversion (SVN)
- ◆ Git
- ◆ Mercurial

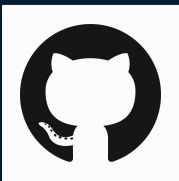
Por que usar?

- Facilidade para verificar as mudanças no código entre versões
- Facilidade em restaurar versões
- Permite unir alterações feitas por diversos desenvolvedores
- Evita acumular arquivos numerados (ou com data da versão)
- Histórico das alterações
 - ◆ Responsável pelas modificações

Sobre o git

- Sistema open-source de gerenciamento de versões
- Git foi desenvolvido e projetado por Torvalds em 2005
 - ◆ auxiliar no desenvolvimento do kernel do Linux
 - ◆ ênfase em velocidade e praticidade
- Pode ser usado para controlar versões de qualquer formato (geralmente arquivos de texto)

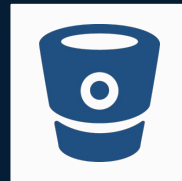
Sistemas populares



Github



Gitlab



Bitbucket

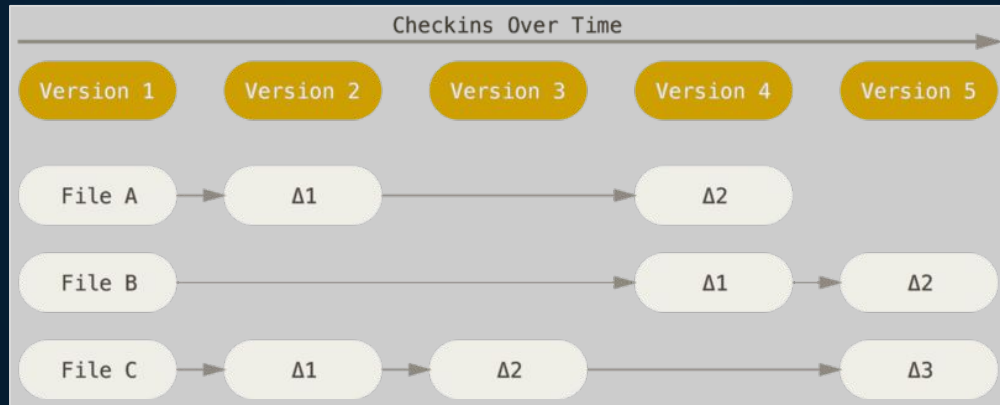


Características do Git

- Principais características que difere o Git de outros controladores de versão.

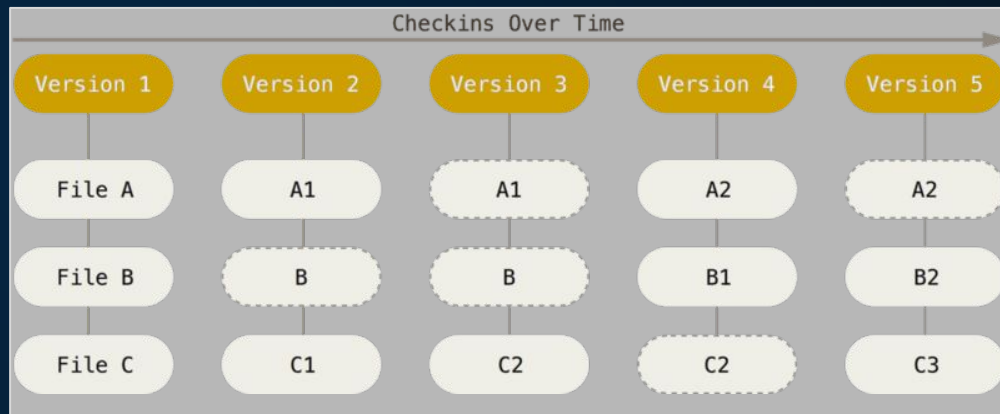
Snapshots

- 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.



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.
- 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

- O Git possui três estados principais nos quais seus arquivos podem residir: **modificado**, **preparado** e **confirmado**.
- 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.

Linux:

- Para Debian, Ubuntu e derivados:
 - ◆ `apt-get install git`

Windows

- <https://gitforwindows.org>

Instalação

- Disponível para: linux, windows e mac os

Configurações iniciais

- O git config é a ferramenta do Git usada para configurações. Através dela você pode alterar configurações pré-definidas no momento em que desejar.
- 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
```

- Se desejar que essas informações sejam diferentes para projetos específicos, é necessário apenas digitar o mesmo comando sem a diretiva --global no diretório em que foi criado um projeto .Git.

Configurações iniciais

- Comando para listar todas as definições que o Git pode encontrar nesse momento:

```
$ git config --list
```

- Encontrar nome de usuário armazenado (o mesmo vale para o email):

```
$ git config user.name
```



Comandos básicos

- Principais comandos

Repositório git

- Obtenção de repositório:
 - a. Transformar um diretório local que atualmente não está sob o controle de versão
 - b. Clonar um repositório Git existente

Repositório git

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

→ linux:

```
$ cd /home/user/project  
$ git init
```

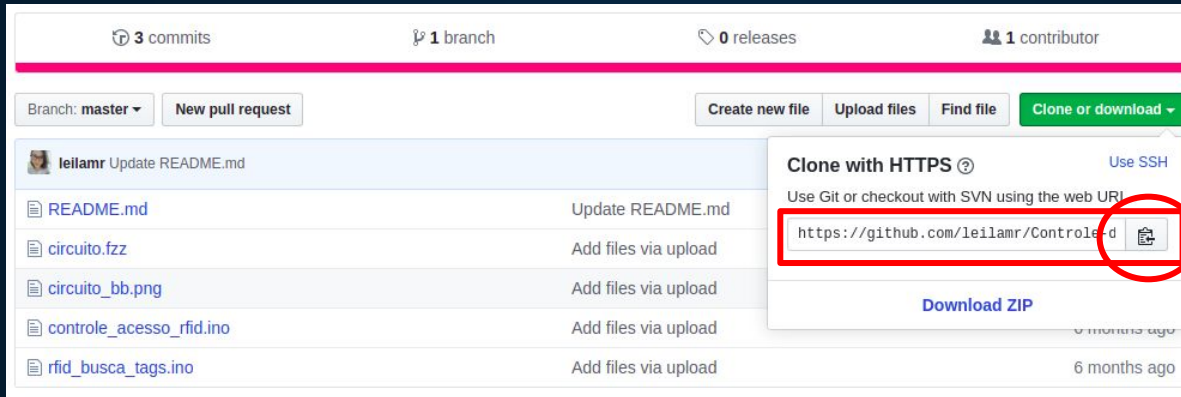
→ windows:

```
$ cd C:\Users\nomeusuario\Desktop  
$ git init
```

Repositório git

Clonar um repositório Git existente

→ Copie a URL do repositório



→ Digite o código abaixo:

```
$ git clone <url-do-repositório>
```

Adicionar arquivos

→ Adicionar um ou mais arquivos

```
$ git add <nome-arquivo>  
$ git add *.extensão  
$ git add *
```

→ Verificar estado de arquivos

```
$ git status
```

→ Confirmando alterações

```
$ git commit  
$ git commit -m "sua mensagem"
```

Ignorando arquivos

- Commite 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.
- Use o arquivo .gitignore para indicar arquivos que devem ser ignorados.

```
# 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
```

Removendo 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  
$ git commit
```

Histórico

- Para verificar o histórico dos commits efetuados em determinado repositório

```
$ git log
```

- Lista os commits feitos nesse repositório em ordem cronológica reversa; isto é, os commits mais recentes aparecem primeiro.

```
leilarodrigues@alu:~/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 que entender e aprender como gerenciar um repositório remoto do Git.
- Alguns dos conhecimentos necessários para conseguir gerenciar um repositório remoto são:
 - ◆ adicionar um repositório remoto, remover repositórios remotos inválidos, gerenciar ramificações
- Para ver quais servidores remotos você configurou, você pode executar o comando:

```
$ git remote
```


Repositórios remotos

- Quando o projeto está em um ponto que se deseja compartilhar, é necessário enviá-lo a montante. O comando para isso é simples: `git push <remote> <branch>`

```
$ git push origin master
```

- Buscar os dados do servidor de onde foi feito o clone originalmente e automaticamente tenta fazer o merge dele no código que está sendo trabalhado no momento.

```
$ git pull origin master
```

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>
```

Merge

- Mesclar a ramificação criada com o ramo master

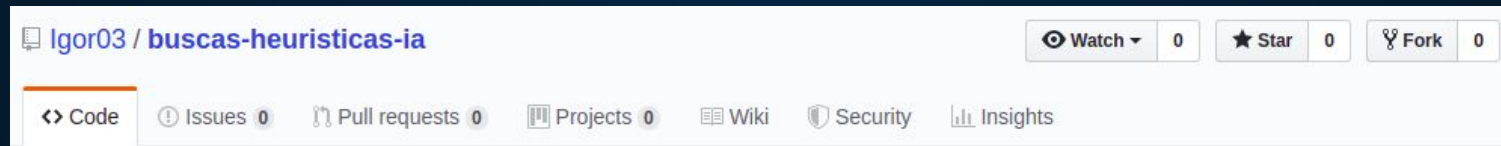
```
$ git checkout master  
$ git merge <nome>
```

- Excluindo branch

```
$ git branch -d <nome>
```

Bifurcações

- Contribuir para um projeto existente para o qual não se tem acesso por push, pode ser feito uma bifurcação do projeto.
- Quando você "bifurca" um projeto, o Git fará uma cópia do projeto que é inteiramente sua.



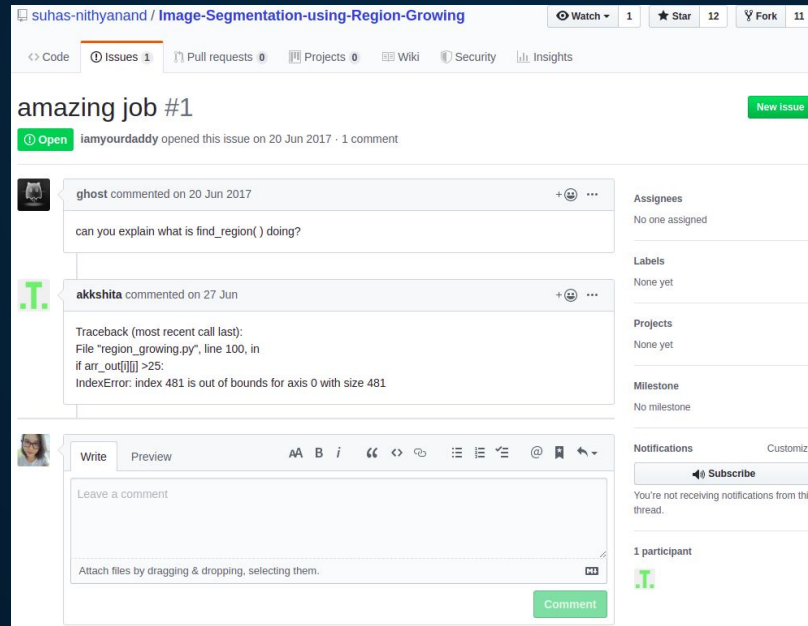
Reveter commits

→ Desfazendo um commit específico

```
$ git revert 58cffab
```

Reveter commits

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



The screenshot shows a GitHub repository page for 'suhas-nithyanand / Image-Segmentation-using-Region-Growing'. The 'Issues' tab is active, displaying an issue titled 'amazing job #1' which was opened by 'iamyourdaddy' on June 20, 2017. The issue is currently 'Open' and has one comment. The comment, made by 'ghost' on June 20, 2017, asks for an explanation of the 'find_region()' function. A second comment by 'akshita' on June 27, 2017, provides a traceback from a Python script, indicating an 'IndexError: index 481 is out of bounds for axis 0 with size 481'. The right sidebar shows that the issue has no assignees, labels, projects, milestones, or notifications. A 'Subscribe' button is visible, and it notes that the user is not currently receiving notifications for this thread. At the bottom, there is a '1 participant' section with the user 'akshita' listed.

Licenças

LuckyMallari / HABPanel-keypad

Watch 1 Star 2 Fork 0

Code Issues 0 Pull requests 0 Projects 0 Wiki Security Insights

HABPanel Universal Keypad

6 commits 1 branch 0 releases 1 contributor MIT

Branch: master New pull request

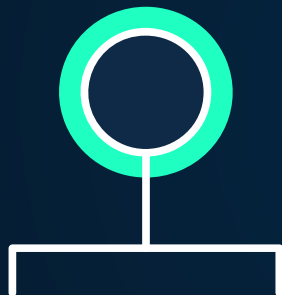
Create new file Upload files Find file Clone or download

LuckyMallari Fixed link Latest commit 6dc66db on 21 Jan 2018

img	Initial Commit	2 years ago
Keypad.widget.json	Fixes #1	2 years ago
LICENSE	Initial commit	2 years ago
README.md	Fixed link	2 years ago
keypad.html	Fixes #1	2 years ago

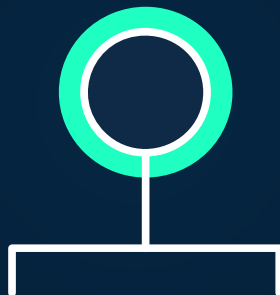
Licença

→ <http://escolhaumalicensa.com.br>



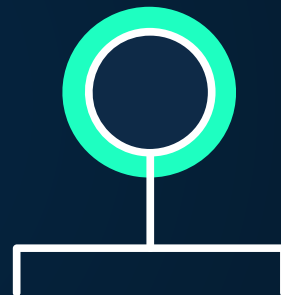
Licença MIT

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.



Licença Apache

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



GPL (V2 ou V3)

licença "copyleft" que exige que quem distribui o seu código ou uma obra derivada deve disponibilizar o fonte sob os mesmos termos.

README.md

README.md

Gracefulbot

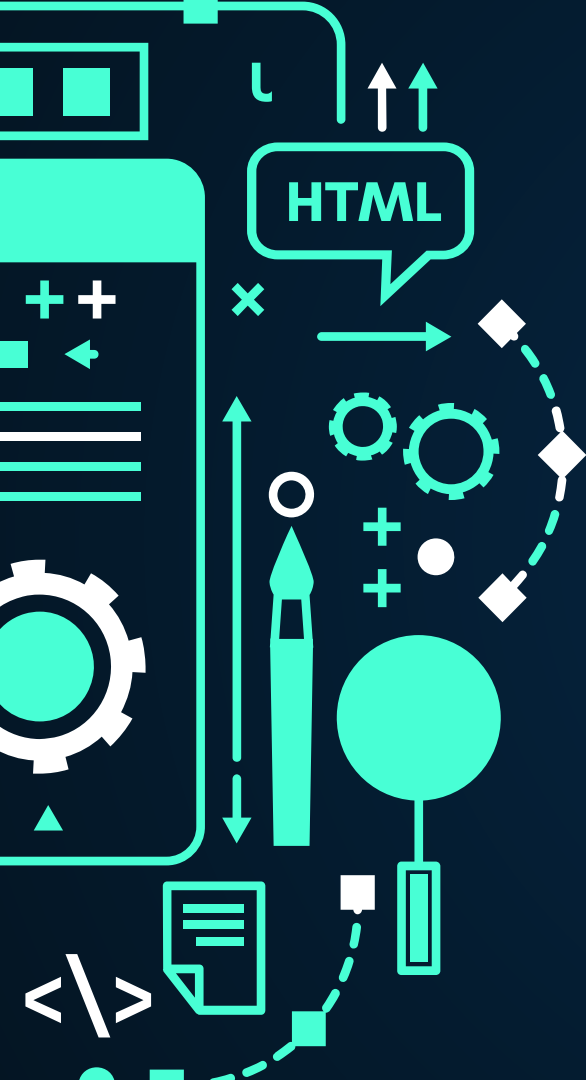
Minibot com reconhecimento de saudações, agradecimentos e despedidas usando TensorFlow integrado ao Telegram (pt-br)(python)

Requisitos

- Python 3.5
- Bibliotecas importantes:
 - NLTK
 - Numpy
 - Tflern
 - Telebot

Funcionamento

- O projeto é dividido em três passos:
 - Transformar intenções de conversação em um modelo TensorFlow
 - Criar uma estrutura de chatbot para processar respostas
 - Conectar o nosso modelo ao Telegram



Links

- **Documentação oficial:** <https://book.git-scm.com/book/en/v2/>
- **Guia rápido:** https://rogerdudler.github.io/git-guide/index.pt_BR.html
- <http://coral.ufsm.br/pet-si/wp-content/uploads/2017/02/Consultório-de-Software-Git.pdf>
- <https://woliveiras.com.br/posts/introdução-a-versionamento-de-código-e-conhecendo-o-git/#IntroduoaversionamentodecdigoeGit>