

Controle de Versão e Introdução git

Felipe Timóteo

GISIS & DOT-UFF

February 9, 2019



Outline

1 Introdução

- Motivação
- Controle de versão
- git
- github & gitlab

2 Git Workflow Básico

- Configurando o git
- Criando repositório
- Fazendo alterações
- Exemplo - primeiros passos
- Colaborando e Sincronizando
- Exemplo - primeiros passos

3 Git Workflow Avançado [não finalizado]

- Branches
- Fork
- Trabalhando em equipe

4 Links úteis e referências

Outline

1 Introdução

- Motivação
- Controle de versão
- git
- github & gitlab

2 Git Workflow Básico

- Configurando o git
- Criando repositório
- Fazendo alterações
- Exemplo - primeiros passos
- Colaborando e Sincronizando
- Exemplo - primeiros passos

3 Git Workflow Avançado [não finalizado]

- Branches
- Fork
- Trabalhando em equipe

4 Links úteis e referências

Introdução

Motivação

- ✓ Ajuda na organização do projeto de forma eficiente
- ✓ Documentação e histórico do desenvolvimento mais robusta
- ✓ Facilita a colaboração de pequenos ou grandes grupos
- ✓ Economia de tempo durante o desenvolvimento
- ✓ Facilita debug de programas complexos
- ✓ Muito popular entre os desenvolvedores de softwares
- ✓ Evita problemas comum aos desenvolvedores (ou escritores)
- ✓ Evita as várias versões diferentes durante a etapa de desenvolvimento
- ✓ Evita tarefas chatas e repetitivas



Introdução

Motivação

- ✓ Ajuda na organização do projeto de forma eficiente
- ✓ Documentação e histórico do desenvolvimento mais robusta
- ✓ Facilita a colaboração de pequenos ou grandes grupos
- ✓ Economia de tempo durante o desenvolvimento
- ✓ Facilita debug de programas complexos
- ✓ Muito popular entre os desenvolvedores de softwares
- ✓ Evita problemas comum aos desenvolvedores (ou escritores)
- ✓ Evita as várias versões diferentes durante a etapa de desenvolvimento
- ✓ Evita tarefas chatas e repetitivas



Introdução

Motivação

- ✓ Ajuda na organização do projeto de forma eficiente
- ✓ Documentação e histórico do desenvolvimento mais robusta
- ✓ Facilita a colaboração de pequenos ou grandes grupos
- ✓ Economia de tempo durante o desenvolvimento
- ✓ Facilita debug de programas complexos
- ✓ Muito popular entre os desenvolvedores de softwares
- ✓ Evita problemas comum aos desenvolvedores (ou escritores)
- ✓ Evita as várias versões diferentes durante a etapa de desenvolvimento
- ✓ Evita tarefas chatas e repetitivas



Introdução

Motivação

- ✓ Ajuda na organização do projeto de forma eficiente
- ✓ Documentação e histórico do desenvolvimento mais robusta
- ✓ Facilita a colaboração de pequenos ou grandes grupos
- ✓ Economia de tempo durante o desenvolvimento
- ✓ Facilita debug de programas complexos
- ✓ Muito popular entre os desenvolvedores de softwares
- ✓ Evita problemas comum aos desenvolvedores (ou escritores)
- ✓ Evita as várias versões diferentes durante a etapa de desenvolvimento
- ✓ Evita tarefas chatas e repetitivas



GESIS

Geologic Information
and Imaging Group

Introdução

Motivação

- ✓ Ajuda na organização do projeto de forma eficiente
- ✓ Documentação e histórico do desenvolvimento mais robusta
- ✓ Facilita a colaboração de pequenos ou grandes grupos
- ✓ Economia de tempo durante o desenvolvimento
- ✓ Facilita debug de programas complexos
- ✓ Muito popular entre os desenvolvedores de softwares
- ✓ Evita problemas comum aos desenvolvedores (ou escritores)
- ✓ Evita as várias versões diferentes durante a etapa de desenvolvimento
- ✓ Evita tarefas chatas e repetitivas



Introdução

Motivação

- ✓ Ajuda na organização do projeto de forma eficiente
- ✓ Documentação e histórico do desenvolvimento mais robusta
- ✓ Facilita a colaboração de pequenos ou grandes grupos
- ✓ Economia de tempo durante o desenvolvimento
- ✓ Facilita debug de programas complexos
- ✓ Muito popular entre os desenvolvedores de softwares
- ✓ Evita problemas comum aos desenvolvedores (ou escritores)
- ✓ Evita as várias versões diferentes durante a etapa de desenvolvimento
- ✓ Evita tarefas chatas e repetitivas



Introdução

Motivação

- ✓ Ajuda na organização do projeto de forma eficiente
- ✓ Documentação e histórico do desenvolvimento mais robusta
- ✓ Facilita a colaboração de pequenos ou grandes grupos
- ✓ Economia de tempo durante o desenvolvimento
- ✓ Facilita debug de programas complexos
- ✓ Muito popular entre os desenvolvedores de softwares
- ✓ Evita problemas comum aos desenvolvedores (ou escritores)
- ✓ Evita as várias versões diferentes durante a etapa de desenvolvimento
- ✓ Evita tarefas chatas e repetitivas



Introdução

Motivação

- ✓ Ajuda na organização do projeto de forma eficiente
- ✓ Documentação e histórico do desenvolvimento mais robusta
- ✓ Facilita a colaboração de pequenos ou grandes grupos
- ✓ Economia de tempo durante o desenvolvimento
- ✓ Facilita debug de programas complexos
- ✓ Muito popular entre os desenvolvedores de softwares
- ✓ Evita problemas comum aos desenvolvedores (ou escritores)
- ✓ Evita as várias versões diferentes durante a etapa de desenvolvimento
- ✓ Evita tarefas chatas e repetitivas



Introdução

Motivação

- ✓ Ajuda na organização do projeto de forma eficiente
- ✓ Documentação e histórico do desenvolvimento mais robusta
- ✓ Facilita a colaboração de pequenos ou grandes grupos
- ✓ Economia de tempo durante o desenvolvimento
- ✓ Facilita debug de programas complexos
- ✓ Muito popular entre os desenvolvedores de softwares
- ✓ Evita problemas comum aos desenvolvedores (ou escritores)
- ✓ Evita as várias versões diferentes durante a etapa de desenvolvimento
- ✓ Evita tarefas chatas e repetitivas

Introdução

Controle de versão | <https://guides.github.com/introduction/git-handbook/>

What's a version control system?

A version control system, or VCS, tracks the history of changes as people and teams collaborate on projects together. As the project evolves, teams can run tests, fix bugs, and contribute new code with the confidence that any version can be recovered at any time. Developers can review project history to find out:

- Which changes were made?
- Who made the changes?
- When were the changes made?
- Why were changes needed?



Introdução

Controle de versão | <https://guides.github.com/introduction/git-handbook/>

What's a distributed version control system?

Git is an example of a distributed version control system (DVCS) commonly used for open source and commercial software development. DVCSs allow full access to every file, branch, and iteration of a project, and allows every user access to a full and self-contained history of all changes. Unlike once popular centralized version control systems, DVCSs like Git don't need a constant connection to a central repository. Developers can work anywhere and collaborate asynchronously from any time zone

Without version control, team members are subject to redundant tasks, slower timelines, and multiple copies of a single project. To eliminate unnecessary work, Git and other VCSs give each contributor a unified and consistent view of a project, surfacing work that's already in progress. Seeing a transparent history of changes, who made them, and how they contribute to the development of a project helps team members stay aligned while working independently.

Introdução

Why Git? | <https://guides.github.com/introduction/git-handbook/>

According to the latest Stack Overflow developer survey, more than 70 percent of developers use Git, making it the most-used VCS in the world. Git is commonly used for both open source and commercial software development, with significant benefits for individuals, teams and businesses.

- Git lets developers see the entire timeline of their changes, decisions, and progression of any project in one place. From the moment they access the history of a project, the developer has all the context they need to understand it and start contributing.
- Developers work in every time zone. With a DVCS like Git, collaboration can happen any time while maintaining source code integrity. Using branches, developers can safely propose changes to production code.
- Businesses using Git can break down communication barriers between teams and keep them focused on doing their best work. Plus, Git makes it possible to align experts across a business to collaborate on major projects.



Introdução

git | github ou gitlab



GitLab



1 Introdução

- Motivação
- Controle de versão
- git
- github & gitlab

2 Git Workflow Básico

- Configurando o git
- Criando repositório
- Fazendo alterações
- Exemplo - primeiros passos
- Colaborando e Sincronizando
- Exemplo - primeiros passos

3 Git Workflow Avançado [não finalizado]

- Branches
- Fork
- Trabalhando em equipe

4 Links úteis e referências

Git Workflow

Preparando o ambiente pela 1^a vez

Defina o user name

```
$ git config --global user.name "[nome]"
```

Defina o email

```
$ git config --global user.email "[endereço-de-email]"
```

Melhore a visualização usando cores

```
$ git config --global color.ui auto
```

Confirme as configuras com:

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

- ★ Essa configurações irão valer para todos os seus repositórios locais



Git Workflow

Criando o repositório

Cria um novo repositório

```
$ git init
```



Clona um repositório existente

```
$ git clone [url]
```



Sincroniza com um repositório existente

```
$ git remote add origin [url]  
$ git pull
```



Git Workflow

Fazendo alterações

Lista os arquivo novos ou modificados

```
$ git status
```

Prepara o arquivo para o versionamento | *staging area*

```
$ git add [nome do arquivo]
```

Mostra a diferença entre os arquivos

```
$ git diff ou $ git diff --staged
```

Grava modificações no histórico de versão

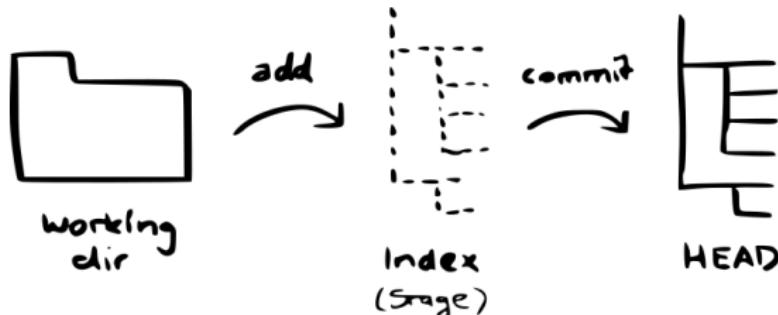
```
$ git commit -m "[mensagem descriptiva]"
```

Verifique o histórico de versões

```
$ git log
```

Git Workflow

Fazendo alterações | Entendendo as áreas de trabalho



- ✓ Seus repositórios locais consistem em três "árvores" mantidas pelo git.
- ✓ Working Directory - contém os arquivos vigentes.
- ✓ Index (Stage) - funciona como uma área temporária
- ✓ HEAD - aponta para o último commit (confirmação) que você fez.



Git Workflow

Exemplo prático | Primeiros passos com git

- ① crie um novo diretório e prepare [ou clone] o repositório

```
$ git init
```

- ② adicione [ou modifique] os arquivos de trabalho

- ③ verifique os arquivos alterados

```
$ git status
```

- ④ adicione os arquivos desejados a *staging area*

```
$ git add
```

- ▶ + alterações? | verifique | *staging area*

```
$ git status | $ git diff | $ git add
```

- ⑤ grave as modificações no histórico de versão

```
$ git commit -m "[mensagem descritiva]"
```

- ⑥ verifique o histórico de versões

```
$ git log"
```

Git Workflow

Colaborando e Sincronizando | Criando as contas no github e gitlab



Github



Gitlab

Após criar a conta...

crie um novo repositório



Git Workflow

Colaborando e Sincronizando | Configurando o caminho



Define o caminho até um repositório na nuvem

```
$ git remote add <origin> [url]
```

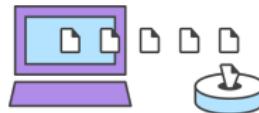
verifique o caminho

```
$ git remote -v
```



Git Workflow

Colaborando e Sincronizando | repo nuvem = repo local



Sincronize o histórico de versão na nuvem com o repositório local

```
$ git fetch <origin>
```

Verifique o histórico de versão sincronizado

```
$ git log <origin>/master
```

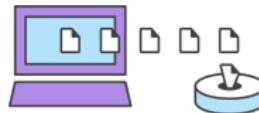
Combine os repositórios

```
$ git merge <origin>/master
```



Git Workflow

Colaborando e Sincronizando | repo nuvem = repo local



Sincronização e combinação de uma vez

```
$ git pull <origin>
```

*Se tiver certeza que a sincronização e combinação pode ser feita, agilize o processo com esse comando único

Sincronize o histórico de versão local com o repositório na nuvem

```
$ git push <origin> master
```

Git Workflow

Exemplo prático | Sincronizando com a nuvem

- ① Adicione o caminho do repositório na nuvem e confirme o caminho

```
$ git remote add <origin> [url]  
$ git remote -v
```

- ② Sincronize o repositório da nuvem com o repositório local

```
$ git fetch <origin>  
$ git log <origin>/master  
$ git merge <origin>/master
```

ou

```
$ git pull <origin>
```

- ③ verifique os arquivos alterados

```
$ git status
```

- ④ Faça alterações no seu repositório local

- ⑤ grave as modificações no histórico de versão e verifique

- ⑥ Sincronize o histórico de versão local com o repositório na nuvem



1 Introdução

- Motivação
- Controle de versão
- git
- github & gitlab

2 Git Workflow Básico

- Configurando o git
- Criando repositório
- Fazendo alterações
- Exemplo - primeiros passos
- Colaborando e Sincronizando
- Exemplo - primeiros passos

3 Git Workflow Avançado [não finalizado]

- Branches
- Fork
- Trabalhando em equipe

4 Links úteis e referências

Links úteis e referências

- [Git hub handbook](#)
- [Git hub glossary](#)
- [GitHub by Chris Grandin and Andrew Edwards](#)
- [Resources to learn Git](#)
- [Atlassian](#)
- [git - guia prático](#)