



Git

Data: @Aug 26, 2020



Coluna de interpretação

- ▼ O que é git?
É um programa que auxilia versionamento de arquivos e facilita contribuição em projetos com mais de um participante.
- ▼ O que é um repositório?
- ▼ Qual a diferença de um repositório para um diretório comum?
- ▼ O que é um repositório local?
- ▼ O que é um repositório remoto?
- ▼ O que é um commit?
- ▼ O que é uma branch?
- ▼ O que é uma branch local?
- ▼ O que é uma branch remota?
- ▼ O que é o ambiente de preparação (*staging*)?
- ▼ O que são conflitos?
- ▼ Como posso evitar conflitos?



Coluna de anotação

Sumário

[Sumário](#)

["História"](#)

[Como a contribuição funciona?](#)

[Beleza, como eu começo?](#)

[Como transformo um projeto já existente em um repositório git?](#)

[Como copiar um repositório já existente?](#)

[É possível copiar mais de um repositório para meu repositório local?](#)

[Beleza, mas como o git funciona?](#)

[Como vejo os commits de um projeto?](#)

[O que são aqueles nomes entre parênteses ao lado do commit no topo?](#)

[Ok... Mas o que são essas ramificações?](#)

[Branches locais e remotas?](#)

[Como introduzir modificações ao projeto?](#)

[Massa, valeu. Entendi nada.](#)

[Como criar uma *branch* local para trabalhar nela?](#)

[Depois de modificar, o que devo fazer? Ou, o que é preparar modificações?](#)

[Então depois de preparar e revisar o que preparei finalmente posso confirmar?](#)

[Me satisfiz com as modificações, quero enviar pro repositório remoto. E agora?](#)

[Alguma modificação foi mesclada na *branch* principal do projeto \(possivelmente a que enviei\). Como atualizo a minha referência local?](#)

[Como insiro novidades de uma *branch* remota numa *branch* local?](#)

[Sobre conflitos.](#)

[Dicas](#)

"História"

O git foi criado para auxiliar o desenvolvimento do kernel linux. Como é um projeto muito grande, envolvendo muitos desenvolvedores, se viu necessária uma maneira de facilitar o compartilhamento de código e a introdução de alterações no projeto vinda de forma distribuída.

Como a contribuição funciona?

O git é um programa que transforma um diretório comum de arquivos (o projeto) em um diretório especial chamado "repositório". Esse repositório pode ser copiado para qualquer pessoa, fazendo com a mesma tenha acesso ao projeto inteiro em seu computador (chamado de repositório local). As pessoas interagem por meio de seus próprios repositórios, fazendo modificações ao projeto e então, quando satisfeitas com alguma modificação, pedem para quem mantém o repositório original "dar uma olhada" nas modificações que foram feitas na cópia e, se for desejável, introduzir as modificações no repositório original.

Beleza, como eu começo?

Primeiro é preciso instalar e configurar o git, informando quem é você (essa informação é relevante para que pessoas que vejam seu trabalho possam saber quem contatar).

Após a instalação, execute os comandos:

```
$ git config --global user.name "<seu nome>"
$ git config --global user.email "<seu endereço de email>"
```

▼ Como posso corrigir conflitos?

Como transformo um projeto já existente em um repositório git?

Para essa transformação é necessário executar o comando:

```
$ git init
```

Este comando deve ser executado no diretório do projeto que deseja utilizar o git. A partir desse comando, a única diferença pro diretório original será uma pasta `.git` que foi criada e que irá manter toda a arquitetura utilizada pelo git para versionar o projeto.

Este comando só precisa ser utilizado uma vez.

Como copiar um repositório já existente?

A cópia do repositório é feita através do comando:

```
$ git clone <endereço do repositório>
```

Com isto, um diretório com o nome do repositório que está sendo copiado (clonado) será criado no diretório atual no sistema de arquivos do computador.

Este comando só precisa ser utilizado uma vez.

É possível copiar mais de um repositório para meu repositório local?

De certa forma sim. Para visualizar os repositórios que o seu repositório local está "referenciando" execute o comando:

```
$ git remote -v
```

Algo do tipo deve aparecer como resultado:

```
~/work/projects/diario-oficial main* ↑ 23m 1s
> git remote -v
origin  git@github.com:giuliocc/diario-oficial.git (fetch)
origin  git@github.com:giuliocc/diario-oficial.git (push)
upstream  git@github.com:okfn-brasil/querido-diario.git (fetch)
upstream  git@github.com:okfn-brasil/querido-diario.git (push)
```

O `git clone` já cria o remoto de apelido "origin". Este é o apelido padrão do repositório de onde o seu repositório local foi copiado.

O repositório de apelido `upstream` foi adicionado manualmente, através do comando:

```
$ git remote add upstream <endereço do repositório>
```

E esta é a maneira que um repositório local pode "copiar" mais de um repositório, através do apelido que foi dado ao "repositório remoto". Sempre que quiser referenciar o repositório remoto (ex: ao introduzir no repositório local novidades do repositório remoto), o apelido dele deve ser usado para ter certeza a qual remoto você está se referindo.

Este comando só precisa ser utilizado uma vez para cada remoto que for adicionar.

Beleza, mas como o git funciona?

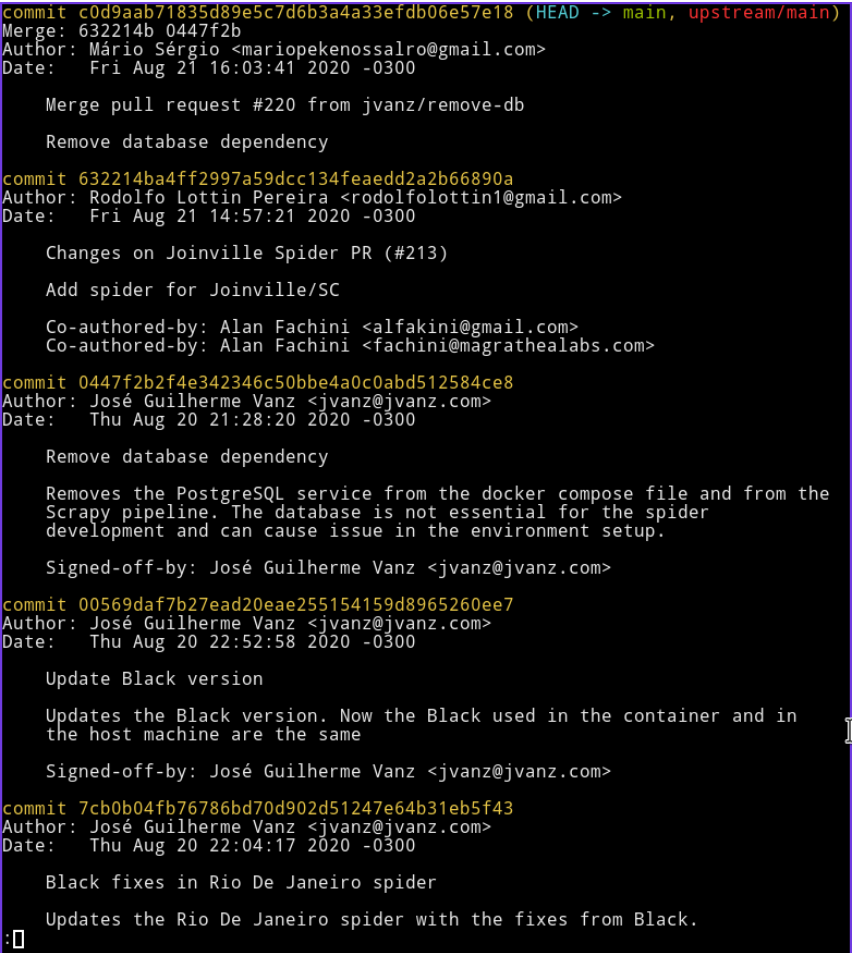
O git funciona através de confirmações (do inglês, *commits*). Cada modificação realizada pode ser confirmada. Não há um limite determinado para o tamanho da modificação, podendo virar o projeto do avesso ou apenas alterar um caractere em um arquivo. Porém, é indicado que cada *commit* tenha um propósito único, que seja um bloco lógico de modificação do projeto.

Como vejo os commits de um projeto?

Para isso, é necessário executar o comando:

```
$ git log
```

Algo do tipo pode ser visto como resultado:



Esta é uma sequência de *commits*, mais conhecida como "a história". Cada commit tem algumas características que aparecem ao executar o `git log` :

- Identificador único (ex: `c0d9aab71835d89e5c7d6b3a4a33efdb06e57e18`)
- Autor do *commit*
- Data de criação do *commit*
- Mensagem do *commit*

Destas, vale a pena destacar duas: o identificador único e a mensagem do *commit*.

O identificador único serve para que cada commit seja realmente único no projeto inteiro. Assim, cada modificação pode ser rastreada individualmente e jamais será perdida.

A mensagem de commit se divide em título e descrição. O título é correspondente à primeira linha e deve descrever o que é a modificação realizada, em outras palavras, "o que é esta alteração". A descrição é inserida após o título e de uma linha em branco e descreve as razões pelas quais aquela modificação foi introduzida, ou, "por que estou realizando esta alteração". O "como fiz esta alteração" é o próprio conteúdo da alteração 😊.

O que são aqueles nomes entre parenteses ao lado do commit no topo?

- `HEAD` : *commit* atual (versão do projeto que você está vendo)
- `main` : nome de uma ramificação (do inglês, *branch*) local
- `HEAD -> main` : significa que o commit atual é o mesmo commit que o último da *branch* `main`
- `upstream/main` : uma ramificação remota (*branch* `main` do repositório remoto de apelido `upstream`)
- `HEAD -> main, upstream/main` : significa que o *commit* atual é o mesmo *commit* que o último da *branch* `main` , assim como o último da *branch* `main` do remoto `upstream` , ou seja, tanto `main`

quanto `upstream/main` estão atualizadas entre si, pois o seu último *commit* é o mesmo.

Ok... Mas o que são essas ramificações?

Uma *branch* é uma variação do projeto. A convenção é manter uma *branch* principal como sendo a "história oficial do projeto" e usar outras *branches* pra testar variações e caso esteja satisfeito com a modificação realizada, tentar mesclar essa modificação na *branch* principal.

Normalmente, a *branch* principal é chamada de `main`.

Visualizar as *branches* no seu repositório local é possível através dos comandos:

```
$ git branch
$ git branch -r
```

Com o primeiro é possível visualizar as *branches* locais:

```
df_brasilia
doc-troubleshooting
fix_duplicate_key_error
local-user-id
* main
mg_itauna
pa_belem_spider
pb_joao_pessoa
pe-jaboatao
pe_recife
sigpub
(END)
```

O asterisco "*" denota a *branch* local atual.

E com o segundo as *branches* remotas:

```
origin/HEAD -> origin/master
origin/cuducos-belo-horizonte
origin/doc-troubleshooting
origin/es
origin/master
origin/pe_recife
origin/rs_caxias_do_sul
origin/sigpub
origin/splash_spider_example
upstream/main
upstream/master
(END)
```

Branches locais e remotas?

É... *branches* locais são *branches* existentes apenas no seu repositório local, já as remotas são referências locais (apenas permitida a leitura) às *branches* que existem nos remotos que você adicionou no seu projeto com o `git remote add`.

Como introduzir modificações ao projeto?

Para inserir modificações no projeto o fluxo a seguir é bem comum:

1. Criar uma *branch* local para trabalhar nela;
2. Realizar modificações;
3. Preparar modificações;
4. Confirmar modificações;
5. Mais modificações? Repetir a partir do passo 2;
6. Caso esteja satisfeita com as modificações, subir modificações para a versão remota da *branch* local;
7. Se o trabalho for avaliado e melhorias forem sugeridas, repetir a partir do passo 2 e continuar dali novamente.

Massa, valeu. Entendi nada.



Vamos passo-a-passo?

Como criar uma *branch* local para trabalhar nela?

É necessário criar uma *branch* a partir da sua *branch* atual. Desse jeito, você "parte do ponto" que aquela *branch* de onde você está partindo te deixou, e irá inserir modificações em cima do trabalho que foi realizado nela.

Para isso, é possível executar o comando:

```
$ git branch <nome da branch>
```

Após isso, é necessário alterar a sua *branch* atual para a *branch* que você acabou de criar.

Para isso, existe o comando:

```
$ git checkout <nome da branch>
```

Também existe um atalho para executar os dois comandos anteriores em um único comando:

```
$ git checkout -B <nome da branch>
```

Agora é a hora de modificar o que você quiser no projeto.

Depois de modificar, o que devo fazer? Ou, o que é preparar modificações?

Confirmar modificações é o ato de adicionar mais um *commit* àquela história que vimos antes com o `git log`. Mas, antes disso, o git te permite colocar as modificações em preparação (chamado de *staging*). Isto serve para você selecionar quais modificações você quer confirmar dentre todas as que você realizou. Isso é importante pois não necessariamente tudo que você modificou fazer parte de um mesmo bloco lógico que faça sentido entrar na história. E também, permite você revisar o que você quer confirmar antes de confirmar de fato.

Para preparar, é possível executar o comando:

```
$ git commit add <caminho do arquivo>
```

Para revisar o que você está preparando, é possível executar os comandos:

```
$ git diff --staged  
$ git status
```

O primeiro comando mostra o conteúdo que foi adicionado:

```
diff --git a/processing/data_collection/setup.py b/processing/data_collection
/setup.py
new file mode 100644
index 0000000..bbd17aa
--- /dev/null
+++ b/processing/data_collection/setup.py
@@ -0,0 +1,10 @@
+# Automatically created by: scrapy-deploy
+
+from setuptools import setup, find_packages
+
+setup(
+    name          = 'project',
+    version       = '1.0',
+    packages      = find_packages(),
+    entry_points  = {'scrapy': ['settings = gazette.settings']},
+)
~
~
~
~
```

Já o segundo dá uma visão geral sobre as modificações que foram realizadas (que foram ou não foram preparadas):

```
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   processing/data_collection/setup.py

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   docker-compose.yml
        modified:   processing/data_collection/scrapy.cfg

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        data/parsed/
        processing/Dockerfile-scrapy
        processing/data_collection/.pdbrc
        processing/data_collection/dbs/
        processing/data_collection/twistd.pid

~/work/projects/diario-oficial main* ↑
> █
```

Na imagem acima, as três seções correspondem (de cima para baixo) a:

- Arquivos preparados;
- Arquivos que estão na história da *branch* e foram modificados;
- Arquivos novos.

Então depois de preparar e revisar o que preparei finalmente posso confirmar?

Sim, agora é a hora de confirmar.

Para isso é possível executar o comando:

```
$ git commit
```

Após executar este comando, um editor será aberto e você poderá inserir a mensagem do *commit*.

Lembrando, um commit precisa ter um título, e, se for necessário, uma descrição (deixando uma linha em branco entre as duas seções).

Um atalho para confirmar as modificações preparadas e já colocar um título no *commit* sem que o editor seja aberto:

```
$ git commit -m "Mensagem do commit"
```

Agora, caso deseje realizar novas modificações, é só repetir os mesmos passos anteriores.

Me satisfiz com as modificações, quero enviar pro repositório remoto. E agora?

Para atualizar (ou criar) a *branch* remota com a história da *branch* local, é possível executar o comando:

```
$ git push <apelido do remoto> <nome da branch>
```

Caso deseje realizar novas modificações depois disso, é só repetir os mesmos passos anteriores.

Alguma modificação foi mesclada na *branch* principal do projeto (possivelmente a que enviei). Como atualizo a minha referência local?

Isso irá acontecer eventualmente, mas resolver isso é tranquilo.

Para atualizar as referências para *branches* remotas caso elas sejam atualizadas no repositório remoto, basta executar o comando:

```
$ git fetch <apelido do remoto> <nome da branch>
```

Ele irá baixar o conteúdo novo, caso haja, e atualizar a referência.

Como insiro novidades de uma *branch* remota numa *branch* local?

Após atualizar a referência para a *branch* remota com o `git fetch` você pode inserir as modificações na sua *branch* local atual (onde você pode trabalhar e está trabalhando atualmente) com o comando:

```
$ git merge <apelido do remoto> <nome da branch>
```

Para atualizar a *branch* local atual com o conteúdo da *branch* remota com apenas um comando (sem precisar executar `git fetch` e em seguida `git merge`), basta executar o comando,:

```
$ git pull <apelido do remoto> <nome da branch>
```

Lembrando, você deve estar trabalhando na *branch* que você quer atualizar para que esses `git merge` ou `git pull` tenham o efeito desejado.

Sobre conflitos.

É possível modificar *commits* previamente realizados, para isso são utilizados comandos como `git rebase` e `git reset`. Porém, com este poder nasce um problema: modificar *commits* prévios criar um ponto de divergência na história da *branch*. Em repositórios locais isso normalmente não deveria dar problemas e caso só você esteja utilizando aquela *branch* é até recomendado o uso para deixar a história mais elegante se quiser, porém, quando se trata de colaboração deve-se ter muito cuidado com o uso. Pois, outras pessoas podem acabar utilizando uma história e outras pessoas outra história pra uma mesma *branch* caso esta *branch* compartilhada tenha sua história alterada. E, como resultado disso, criaríamos o **conflito** quando os códigos de histórias divergentes fossem mesclados. Mas, mesmo assim, o git permite resolver esses conflitos de forma relativamente simples, só não iremos abordar isso aqui pois normalmente não é necessário para suas primeiras contribuições. Mas, caso aconteça, não se desespere pois existe bastante material sobre isso, a comunidade está aí para ajudar.

Dicas

- Por razões históricas, o título do *commit* não deve passar de 50 caracteres e consiste de apenas uma linha. Já a descrição pode ter várias linhas, mas nenhuma linha deve passar de 72 caracteres 😞. É a convenção mais comum e muitos programas dependem disso para renderizar adequadamente a mensagem do *commit*;

- Git não para por aí, a ferramenta tem um poder imenso e aprender ela vale muito a pena pois pode ser usada para quase todo tipo de projeto que envolva arquivos. Se decidir estudar um pouco mais, vai sentir o esforço recompensado, com certeza!
- Aprender git no terminal é legal pois nos acostumamos com os comandos e quando algum problema aparecer temos mais conhecimento sobre a ferramenta para poder pesquisar na internet 😊;
- As referências deste documento estão cheias de materiais super legais que complementam o conteúdo apresentado. Se quiser avançar mais ou ver outras abordagens sobre os mesmo assuntos, não vai se arrepender.



Resumo

Para configurar o git:

```
$ git config --global user.name "<seu nome>"  
$ git config --global user.email "<seu endereço de email>"
```

Para iniciar um repositório:

```
$ git init
```

Para copiar um repositório já existente:

```
$ git clone <endereço do repositório>
```

Para visualizar os remotos referenciados pelo repositório local:

```
$ git remote -v
```

Para adicionar a referência de um repositório remoto:

```
$ git remote add upstream <endereço do repositório>
```

Para visualizar a história:

```
$ git log
```

Para visualizar branches locais:

```
$ git branch
```

Para visualizar *branches* remotas:

```
$ git branch -r
```

Para criar uma *branch* local:

```
$ git branch <nome da branch>
```

Para alterar a sua *branch* atual para outra *branch*:

```
$ git checkout <nome da branch>
```


Para fazer o `git branch` e o `git checkout` em um único comando:

```
$ git checkout -B <nome da branch>
```

Para preparar arquivos para serem confirmados:

```
$ git add <caminho do arquivo>
```

Para revisar o conteúdo dos arquivos que está preparando:

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

Para ter uma visão geral sobre os arquivos que foram modificados no projeto (preparados ou não preparados):

```
$ git status
```

Para confirmar as modificações preparadas:

```
$ git commit
```

Para confirmar as modificações preparadas e já colocar um título no *commit*:

```
$ git commit -m "Mensagem do commit"
```

Para atualizar (ou criar) a *branch* remota com a história da *branch* local, é possível executar o comando:

```
$ git push <apelido do remoto> <nome da branch>
```

Para atualizar a referência para uma *branch* remota:

```
$ git fetch <apelido do remoto> <nome da branch>
```

Para atualizar uma *branch* local com o conteúdo de uma *branch* remota:

```
$ git merge <apelido do remoto> <nome da branch>
```

Para fazer o `git fetch` e o `git merge` em um único comando:


```
$ git pull <apelido do remoto> <nome da branch>
```



Referências

Downloads


Git comes with built-in GUI tools (git-gui, gitk), but there are several third-party tools for users looking for a platform-

 <https://git-scm.com/downloads>



Reference

Quick reference guides: GitHub Cheat Sheet | Visual Git Cheat Sheet

 <https://git-scm.com/docs>



Como instalar o git.

Referência oficial do git.

Book

The entire Pro Git book, written by Scott Chacon and Ben Straub and published by Apress, is available here. All content is available under the Creative Commons Attribution-ShareAlike license. <https://git-scm.com/book/pt-br/v2>

Pro Git

EVERYTHING YOU NEED TO KNOW ABOUT GIT

Livro Pro Git, ou, a referência extra-oficial do git (mais oficial do que extra)

Folha de Dicas de Git do GitHub (pt-BR)

GitHub fornece clientes desktop que incluem uma interface gráfica para as ações mais comuns em um repositório e atualiza automaticamente para a linha de comando do Git para cenários avançados. <https://windows.github.com>
https://training.github.com/downloads/pt_BR/github-git-cheat-sheet/

Dicas de comandos mais usados (complemento à seção "Resumo" acima).

git - guia prático

apenas um guia prático para começar com git. sem complicação ;)
https://rogerdudler.github.io/git-guide/index.pt_BR.html

Guia bem objetivo sobre como utilizar o git.

Conheça o Git: tutoriais, fluxos de trabalho e comandos Git

um branch representa uma linha independente de desenvolvimento. Os branches funcionam como uma abstração do processo de edição/estágio/commit discutido em Noções básicas de Git. <https://www.atlassian.com/br/git>

Guia que cobre vários pontos, desde o iniciante até o mais avançado.