

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
DO NORTE DE MINAS GERAIS - *CAMPUS* PIRAPORA

VITOR JOSÉ RIGOTTO GINO

**MINICURSO GIT E GITHUB**

Pirapora

2019

VITOR JOSÉ RIGOTTO GINO

## **MINICURSO GIT E GITHUB**

Material complementar do minicurso de Git e GitHub, contendo comandos, instruções e observações.

Pirapora

2019

## SUMÁRIO

1	INTRODUÇÃO . . . . .	3
1.1	O que é Git? . . . . .	3
1.2	O que é GitHub? . . . . .	3
2	INSTALANDO O GIT . . . . .	4
3	CRIANDO A CONTA NO GITHUB . . . . .	5
4	ÁREAS DE OPERAÇÃO . . . . .	7
5	CONFIGURANDO SUAS INFORMAÇÕES . . . . .	8
6	CONTROLANDO UM PROJETO . . . . .	9
7	CLONANDO UM PROJETO . . . . .	11
8	STATUS . . . . .	12
9	COMMIT . . . . .	13
10	PULL . . . . .	14
11	PUSH . . . . .	15
12	ALGUNS COMANDOS DO GIT . . . . .	16
12.1	Iniciando o Git . . . . .	16
12.2	Voltando Commits a versões anteriores . . . . .	16
12.3	Jogando o branch criado no branch master . . . . .	18
12.4	Grudando o branch criado no branch master sem o commit . . . . .	18
12.5	Clonando e puxando alterações de projetos . . . . .	18
12.6	Tags . . . . .	19
13	CONCLUSÃO . . . . .	21

## 1 INTRODUÇÃO

Este minicurso traz a você tudo que precisa saber para se tornar um desenvolvedor que possa dominar tanto o git, quanto o Github. O objetivo é trazer os conhecimentos necessários para que você possa, a partir do zero, dominar os conceitos gerais do git, e usar o github para “hospedar” seus projetos pessoais e acompanhar outros projetos de seu interesse.

### 1.1 O QUE É GIT?

Git é um sistema de controle de versão de arquivos. Através dele podemos desenvolver projetos na qual diversas pessoas podem contribuir simultaneamente no mesmo, editando e criando novos arquivos e permitindo que os mesmos possam existir sem o risco de suas alterações serem sobrescritas.

Se não houver um sistema de versão, imagine o caos entre duas pessoas abrindo o mesmo arquivo ao mesmo tempo. Uma das aplicações do Git é justamente essa, permitir que um arquivo possa ser editado ao mesmo tempo por pessoas diferentes. Por mais complexo que isso seja, ele tenta manter tudo em ordem para evitar problemas para nós desenvolvedores.

Outro fator importante do Git é a possibilidade de criar, a qualquer momento, vários snapshots do seu projeto, ou como chamamos mais “nerdmente”, branch. Suponha que o seu projeto seja um site html, e você deseja criar uma nova seção no seu código HTML, mas naquele momento você não deseja que estas alterações estejam disponíveis para mais ninguém, só para você. Isso é, você quer alterar o projeto (incluindo vários arquivos nele), mas ainda não quer que isso seja tratado como “oficial” para outras pessoas, então você cria um branch (como se fosse uma cópia espelho) e então trabalha apenas nesse branch, até acertar todos os detalhes dele. Após isso, você pode fazer um merge de volta do seu branch até o projeto original. Veja bem, se tudo isso que falei só ajudou a te confundir mais – respire fundo – e siga em frente. Na prática tudo fica melhor.

### 1.2 O QUE É GITHUB?

O Github é um serviço web que oferece diversas funcionalidades extras aplicadas ao git. Resumindo, você poderá usar gratuitamente o github para hospedar seus projetos pessoais. Além disso, quase todos os projetos/frameworks/bibliotecas sobre desenvolvimento open source estão no github, e você pode acompanhá-los através de novas versões, contribuir informando bugs ou até mesmo enviando código e correções. Se você é desenvolvedor e ainda não tem github, você está atrasado e essa é a hora de correr atrás do prejuízo.

## 2 INSTALANDO O GIT

O git é um programa que pode ser instalado usando o link *git-scm.com/download/win* para Windows. No link *git-scm.com/download/mac* para Mac. Ou então através do comando *sudo apt-get install git* para plataformas Linux/Debian, como o Ubuntu.

### 3 CRIANDO A CONTA NO GITHUB

Como já mencionado na seção 1.2, o GitHub é um serviço, sendo assim, não é necessário o processo de instalação. Sendo assim, para utilizá-lo é necessário fazer login em sua conta e, caso ainda não possua uma conta no github.com, está na hora de criá-la.

Figura 1 – Tela de Criação do Repositório

Owner: danieltableless / Repository name: site ✓

Great repository names are short and memorable. Need inspiration? How about **irksome-succotash**.

Description (optional): Site de teste

☒ **Public**  
Anyone can see this repository. You choose who can commit.

☐ **Private**  
You choose who can see and commit to this repository.

☒ **Initialize this repository with a README**  
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: None | Add a license: None ⓘ

**Create repository**

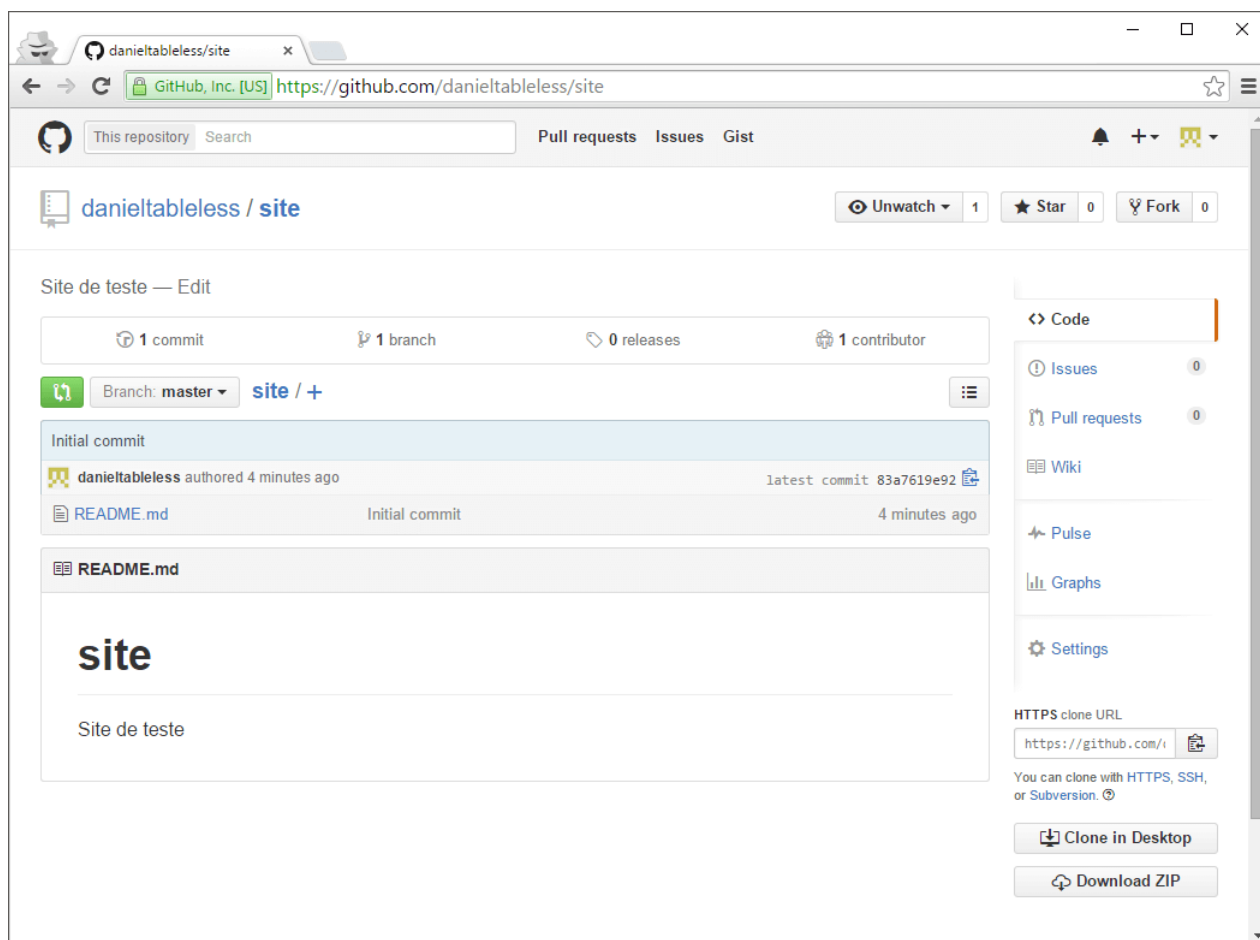
Fonte: [tableless.com.br/tudo-que-voce-queria-saber-sobre-git-e-github-mas-tinha-vergonha-de-perguntar](https://tableless.com.br/tudo-que-voce-queria-saber-sobre-git-e-github-mas-tinha-vergonha-de-perguntar)

Na imagem acima estamos criando um repositório cujo nome é site, de domínio público (podem ser criados reps privados pagando uma mensalidade), e com o arquivo *README.md* embutido, que contém uma descrição do seu projeto. Para que possamos começar a entender como o git funciona, é fundamental criar um rep como este para os nossos testes.

Após a criação do repositório, ele estará disponível no endereço `github.com/<username>/site`, onde *username* é o login que você usou para se cadastrar. Acessando esta url temos a seguinte resposta:

Temos muitas informações nesta tela, pois ela é a tela principal do seu projeto. Explicaremos algumas informações ao longo deste artigo, por enquanto repare apenas no botão *HTTPS Clone Url* na parte inferior à direita. Esta URL será necessária para que possamos “clonar” este projeto em nosso ambiente de estudo (sua máquina windows, mac, linux ou a vm). Clique no botão de copiar URL e perceba que a seguinte URL está na área de transferência:

Figura 2 – Tela Principal do Repositório



Fonte: [tableless.com.br/tudo-que-voce-queria-saber-sobre-git-e-github-mas-tinha-vergonha-de-perguntar](https://tableless.com.br/tudo-que-voce-queria-saber-sobre-git-e-github-mas-tinha-vergonha-de-perguntar)

`https://github.com/<username>/site.git`

## 4 ÁREAS DE OPERAÇÃO

Os locais de operação são as áreas onde os arquivos irão transitar enquanto estão sendo editados e modificados. São 3: Working Directory, Stage Area, Git directory.

O *Git Directory* é onde o Git guarda os dados e objetos do seu projeto. Ele é o diretório mais importante do Git e é ele que será copiado quando alguém clonar (clonar é copiar o projeto para a sua máquina) o projeto.

O *Work Directory* é onde você vai trabalhar. Os arquivos ficam aí para poderem ser usados e alterados quantas vezes quiser para você. É basicamente sua pasta de arquivos dos projeto.

Quando você faz uma alteração em algum arquivo, ele vai para a *Staging Area*, que é uma área intermediária. Basicamente a Staging Area contém o Git Directory com os arquivos modificados, onde ele guarda as informações sobre o que vai no seu próximo commit.



## 5 CONFIGURANDO SUAS INFORMAÇÕES

A primeira coisa que você deve fazer depois de instalar o Git é definir seu username e email. Isso é importante por que os seus commits usarão essas informações para identificar o autor das mudanças. Pois é... Se alguém fizer alguma merda no projeto e quebrar todo o sistema, é possível saber quem, quando e qual linha foi o autor do apocalipse.

É simples, no terminal escreva:

```
git config --global user.name "Seu Nome"
```

```
git config --global user.email seu@email
```

## 6 CONTROLANDO UM PROJETO

Pelo terminal mesmo, entre na pasta do projeto que você quer iniciar o controle e use o comando:

```
git init
```

Esse comando vai criar um diretório invisível dentro do projeto chamado *.git*. Ele contém todos os arquivos necessários do seu repositório. Aqui, neste ponto, nada dos seus arquivos ainda estão sendo controlados. Você apenas criou um “lugar” (branch) para o Git colocar os arquivos.

O próximo comando vai inserir os arquivos que você quer controlar. Normalmente a gente controla TUDO o que está no projeto. Mas isso tem que ser combinado com a equipe antes. Em um projeto que envolve um CMS com o WordPress, por exemplo, é normal controlar tudo, até os arquivos do WordPress. Mas se em um projeto você guarda pastas de layouts, pastas de wireframes, protótipos e etc, é interessante não colocar isso no Git. Mas aí vai de equipe para equipe, de projeto pra projeto.

O comando para adicionar os arquivos é:

```
git add arquivo.formato
```

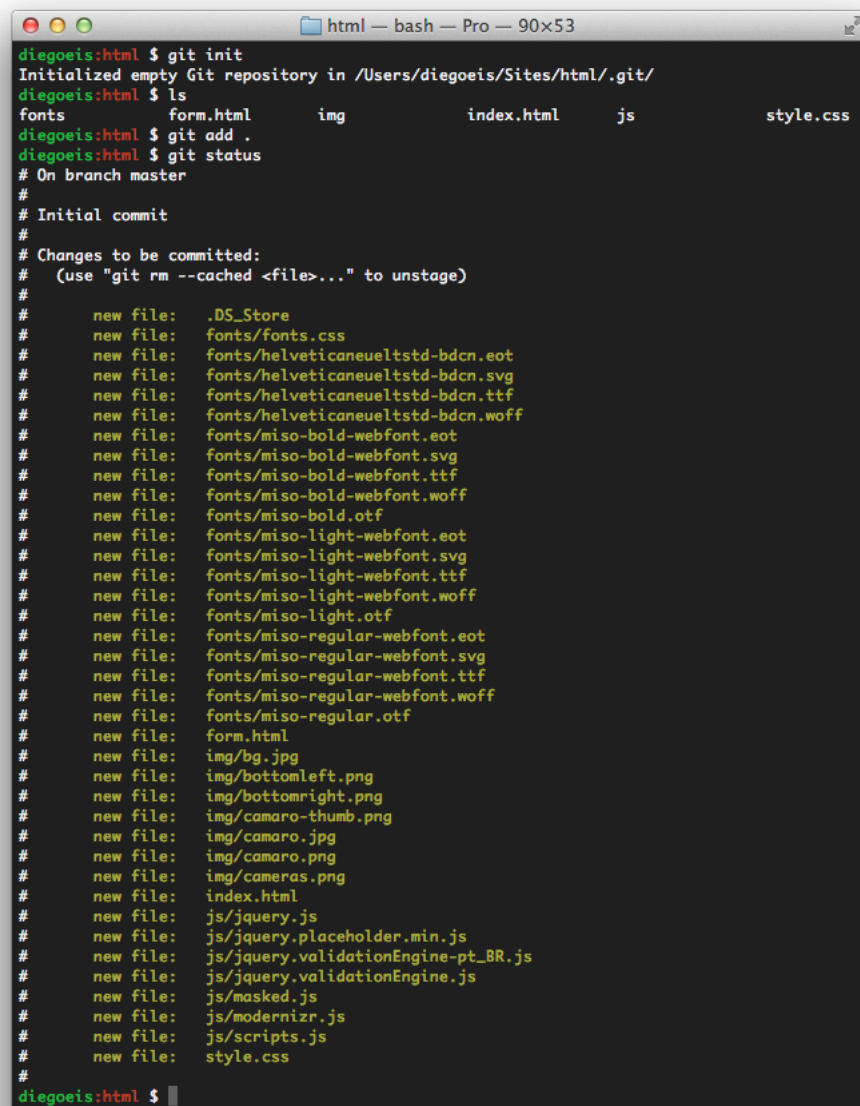
O comando acima é utilizado para adicionar arquivos individualmente do Work Directory para a Staging Area, agora se você deseja adicionar mais de um arquivo, basta utilizar a mesma sintaxe, porém separando os arquivos utilizado a barra de espaço. Observe o exemplo abaixo:

```
git add arquivo.formato1 arquivo.formato2
```

Ou, para adicionar *todos* os arquivos que estão no Work Directory para a Staging Area, basta utilizar o seguinte comando:

```
git add .
```

Para você ver o status, use o comando *git status*, aí você verá tudo o que foi incluído no projeto. Veja o screenshot abaixo para ter uma ideia:

Figura 3 – Comando *git status*A terminal window titled 'html — bash — Pro — 90x53' showing the output of the 'git status' command. The window has a dark background with light green text. The output shows that a new Git repository has been initialized in '/Users/diegoeis/Sites/html/.git/'. The current branch is 'master'. The status shows an initial commit with changes to be committed. A list of new files is displayed, including font files, images, and JavaScript files.

```
diegoeis:html $ git init
Initialized empty Git repository in /Users/diegoeis/Sites/html/.git/
diegoeis:html $ ls
fonts          form.html      img            index.html     js             style.css
diegoeis:html $ git add .
diegoeis:html $ git status
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#       new file:   .DS_Store
#       new file:   fonts/fonts.css
#       new file:   fonts/helveticanueultstd-bdcn.eot
#       new file:   fonts/helveticanueultstd-bdcn.svg
#       new file:   fonts/helveticanueultstd-bdcn.ttf
#       new file:   fonts/helveticanueultstd-bdcn.woff
#       new file:   fonts/miso-bold-webfont.eot
#       new file:   fonts/miso-bold-webfont.svg
#       new file:   fonts/miso-bold-webfont.ttf
#       new file:   fonts/miso-bold-webfont.woff
#       new file:   fonts/miso-bold.otf
#       new file:   fonts/miso-light-webfont.eot
#       new file:   fonts/miso-light-webfont.svg
#       new file:   fonts/miso-light-webfont.ttf
#       new file:   fonts/miso-light-webfont.woff
#       new file:   fonts/miso-light.otf
#       new file:   fonts/miso-regular-webfont.eot
#       new file:   fonts/miso-regular-webfont.svg
#       new file:   fonts/miso-regular-webfont.ttf
#       new file:   fonts/miso-regular-webfont.woff
#       new file:   fonts/miso-regular.otf
#       new file:   form.html
#       new file:   img/bg.jpg
#       new file:   img/bottomleft.png
#       new file:   img/bottomright.png
#       new file:   img/camaro-thumb.png
#       new file:   img/camaro.jpg
#       new file:   img/camaro.png
#       new file:   img/cameras.png
#       new file:   index.html
#       new file:   js/jquery.js
#       new file:   js/jquery.placeholder.min.js
#       new file:   js/jquery.validationEngine-pt_BR.js
#       new file:   js/jquery.validationEngine.js
#       new file:   js/masked.js
#       new file:   js/modernizr.js
#       new file:   js/scripts.js
#       new file:   style.css
diegoeis:html $
```

Fonte: <https://tableless.com.br/iniciando-no-git-parte-1/>

Feito isso você vai precisar inserir seu primeiro commit. Use o seguinte comando:

*git commit -m "Primeiro commit - Inserindo os arquivos iniciais do projeto"*

Ao executar o comando anterior, você acaba de mandar uma alteração para o Git.

## 7 CLONANDO UM PROJETO

Pode ser que já exista um projeto no Git criado e você só precise clonar para seu computador. Para isso você vai usar o comando *git clone*.

Quando você clona um projeto, o Git recebe a cópia de todos os dados que tem no servidor. Cada versão de cada arquivo da história inteira do projeto é puxada quando você roda o comando *git clone*.

Para clonar um projeto você precisa ter a URL do Git daquele projeto em específico. O comando completo fica mais ou menos assim:

*git clone **url-do-projeto***

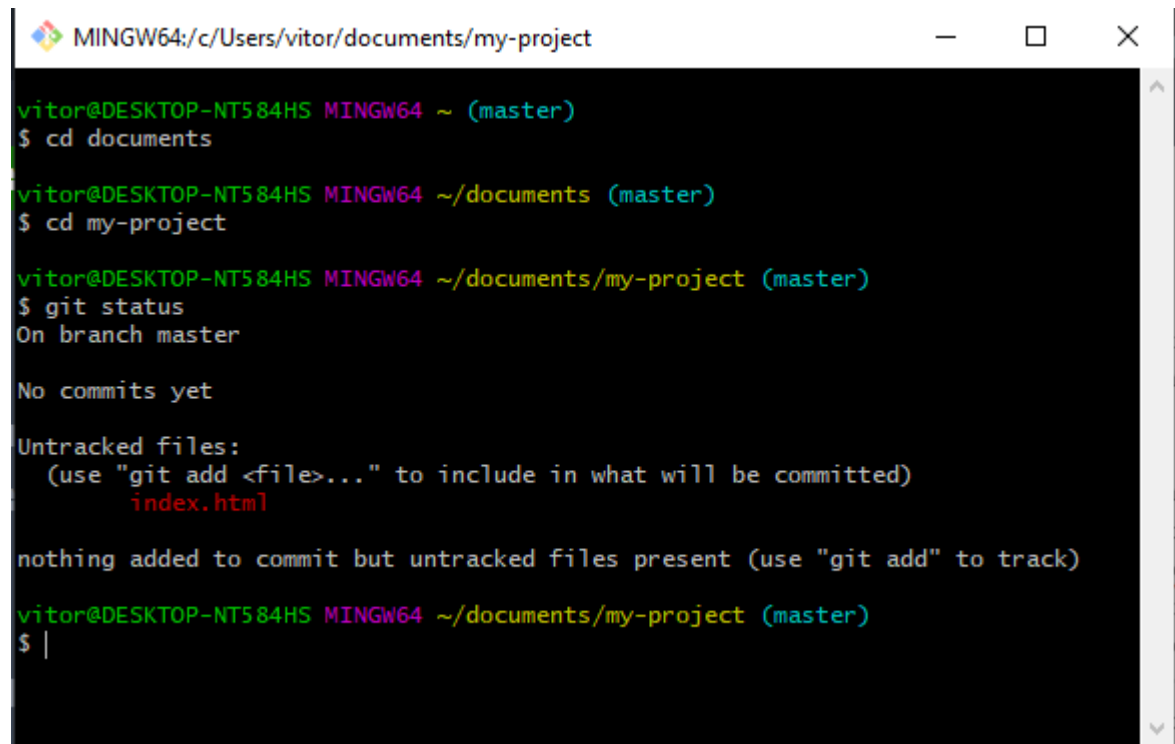
## 8 STATUS

Antes de tudo você precisa entender em qual status os arquivos se encontram. Você pode modificar um arquivo, mas não commita-lo.

Você já clonou ou iniciou seu projeto no Git e agora vai fazer modificações nos arquivos e enviar essas modificações para o repositório. Lembre-se que os arquivos em seu Work Directory podem estar tracked ou untracked. Vou manter os termos em inglês para você se familiarizar melhor. Arquivos com status tracked são arquivos que já estão inseridos no repositório. Eles podem ser unmodified (que não foram modificados por você), modified (que foram modificados por você) ou staged (que são os arquivos que acabaram de ser mudados).

Esse ciclo é repetido diversas e diversas vezes. Veja abaixo um exemplo:

Figura 4 – *git status*



```
MINGW64:/c/Users/vitor/documents/my-project
vitor@DESKTOP-NT584HS MINGW64 ~ (master)
$ cd documents
vitor@DESKTOP-NT584HS MINGW64 ~/documents (master)
$ cd my-project
vitor@DESKTOP-NT584HS MINGW64 ~/documents/my-project (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        index.html

nothing added to commit but untracked files present (use "git add" to track)
vitor@DESKTOP-NT584HS MINGW64 ~/documents/my-project (master)
$ |
```

Fonte: Autor

## 9 COMMIT

Suponha que você resolveu um bug no projeto. É hora de commitar suas modificações. Essas modificações serão inseridas no histórico do projeto e ficarão disponíveis para que os outros integrantes da equipe.

Ao commitar você escreve uma descrição sobre o que foi feito ali. Assim essa modificação não fica perdida e todo mundo sabe do que se trata aquela mudança. Você documenta essa mudança. É mais ou menos isso que é o commit.

Quando você commita uma modificação, os arquivos editados saem do status staged e voltam para o status unmodified. Claro, por que teoricamente aquela alteração já foi feita e agora os arquivos voltam com o status de sem modificações.

O comando é este:

```
git commit -m "Resolvendo bug da modal sobreposta na página de pagamentos."
```

Se você fizer um *git log* no projeto, você consegue visualizar uma lista de todos os commits enviados para o projeto, seus commits e commits de outros integrantes.

## 10 PULL

Não é só você que está fazendo modificações nos arquivos, mas também sua equipe. Por isso é importante que você deixe o projeto sempre atualizado. Para isso você precisa trazer as modificações que eles fizeram e commitaram para o seu repositório local. Você vai usar o comando pull para trazer essas modificações:

```
git pull
```

Feito isso vai até o servidor buscar todas as modificações a partir da versão do seu repositório local, ele vai baixar essas modificações e fará um merge automático nos arquivos necessários que foram modificados. Coisa linda... alguém deve ter modificado o mesmo arquivo que você, o Git vai entender isso e vai juntar seu código com o dele, automaticamente... Claro que pode ser que de conflitos caso vocês tenham modificado a mesma linha, mas aí é outra história, vemos mais pra frente como resolver isso. Se quiser se adiantar, procure sobre o comando diff.

## 11 PUSH

Você modificou os arquivos, commitou descrevendo o que fez exatamente naquela modificação e agora precisa enviar tudo isso para o servidor. O comando *git push* empurra as suas modificações para o servidor, incluindo-as no histórico do projeto. Quando os outros integrantes da equipe fizerem um *git pull*, essas modificações serão baixadas e incluídas no repositório local da pessoa.

*git push*

O Git Push só pode ser feito se você executou o *git pull* antes. Isso é uma forma de você ter o seu repositório atualizado e também para evitar possíveis conflitos no projeto. Quando você faz o pull, se der algum conflito de código, você deverá resolvê-los para depois enviar o novo código novamente.

Há algumas outras opções tanto no Pull e no Push que podemos utilizar para especificar o branch para onde iremos empurrar ou buscar atualizações. Mas isso fica para outra hora.

A documentação do Git é muito fácil de ler e entender. É bem objetiva e não perde tempo. Recomendo que você leia e entenda melhor como utilizar o git nos seus projetos. Nada de FTP, SFTP e outras coisas... Isso é coisa do passado.



## 12 ALGUNS COMANDOS DO GIT

As seções abaixo irão mostrar os comandos mais utilizados no Git.

### 12.1 INICIANDO O GIT

Entre no diretório que deseja controlar a versão e inicie o Git assim:

```
git init
```

Feito isso, seus arquivos ainda não estão sendo versionados, mas eles estão esperando para serem adicionados no estágio de controle. Para fazer isso digite o comando

```
git add nome-do-arquivo-incluindo-extensão
```

Se você precisa adicionar todos os arquivos do diretório, basta digitar:

```
git add .
```

Saber o status do projeto é importante. Com o comando abaixo você consegue ver quais arquivos estão fora do controle, quais foram modificados e estão esperando por uma descrição de modificação etc:

```
git status
```

Voltando ao estágio anterior do adiconamento:

```
git reset HEAD nome-do-arquivo
```

Commit – Comitando:

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

Adicionando e comitando ao mesmo tempo:

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

### 12.2 VOLTANDO COMMITS A VERSÕES ANTERIORES

Voltar um commit: *git reset HEAD 1*

Voltar dois commits:

*git reset HEAD 2*

Voltando um commit e deixando o arquivo no estagio anterior:

*git reset HEAD 1 --soft*

Voltando um commit e excluindo o arquivo, deixando no estágio anterior:

*git reset HEAD 1 --hard*

Verificando o histórico de commits:

*git log*

Verificando o que foi mudado, diferença entre um arquivo e outro:

*git log -p*

Verificando os 2 últimos commits:

*git log -p -2*

Mostrando as estatísticas de todos os commits:

*git log --stat*

Mostrando todos os commits, cada um em uma linha:

*git log --pretty=oneline*

Mostrando todos os commits dos últimos 2 dias até o momento atual *git log --since=2.days*

Criando um branch – uma ramificação

*git checkout -b nome-do-branch*

Verificando em que branch você está

*git branch*

Voltando para o branch master

*git checkout master*

### 12.3 JOGANDO O BRANCH CRIADO NO BRANCH MASTER

Entre como branch master:

*git merge nome-do-branch-que-foi-criado*

### 12.4 GRUDANDO O BRANCH CRIADO NO BRANCH MASTER SEM O COMMIT

Somente localmente – localhost, entre como branch master:

*git rebase nome-do-branch-que-foi-criado*

Removendo um branch:

*git branch -D nome-do-branch*

Vendo branches remotos:

*git branch -a*

Mostrando o início do hash, quem comitou, quanto tempo atrás, mensagem: descrição do commit:

*git log --pretty=format: "%h - %an, %ar : %s"*

Deletando arquivos:

*git rm nome-do-arquivo*

Deletando todos os arquivos removidos ao mesmo tempo:

*git ls-files --deleted | xargs git rm*

### 12.5 CLONANDO E PUXANDO ALTERAÇÕES DE PROJETOS

Clonando um projeto remoto:

*git clone url-do-projeto*

Fazendo um clone de outros branches:

```
git checkout -b nome-do-branch origin/ nome-do-branch
```

Trazendo, puxando as alterações feitas por outros usuários:

```
git pull origin master
```

Sincronizando tudo que está no repositório remoto:

```
git pull
```

Enviando o(s) projeto(s), arquivo(s) para o repositório:

```
git push origin master
```

Enviando um branch para o repositório:

```
git push origin nome-do-branch
```

## 12.6 TAGS

As tags servem para marcar uma etapa. Imagine que você vai lançar uma versão, que resolve uma série de problemas. Você pode marcar aquela etapa criando uma tag. Assim fica simples de fazer qualquer rollback do projeto para uma tag específica em vez de voltar para um commit. Você sabe que tudo o que foi feito até aquela tag está funcionando.

Criando tags:

```
git tag versão-da-tag
```

Listando tags:

```
git tag -l
```

Enviando a tag para o repositório

```
git push origin master -tags
```

Removendo as tags criadas localmente:

```
git tag -d versão-da-tag
```

Removendo tag no repositório remoto:

```
git push origin :refs/tags/versão-da-tag
```

## 13 CONCLUSÃO

Se você quer continuar ou iniciar seus estudos com Git indico o livro Pro Git, escrito por Scott Chacon, é um ótimo começo. Se tiver problemas com o inglês, encontrará várias versões em português.

A CodeSchool juntamente com o GitHub fizeram uma página exclusivamente para ensinar Git na prática.

Há também a documentação do Git que é bastante completa, direta e muito fácil de entender.

É isso aí, agora é só sentar na frente do PC, pegar uma garrafa de café e começar a praticar.

```
public class Fim_do_Material {  
public static void main(String[] args) {  
System.out.println("Bons Estudos, coder!");  
}  
}
```