

# **MANUAL SIMPLIFICADO GIT**



**Versão 1.0**

## Principais comandos GIT

"git init" para iniciar o diretório (repositório) desejado do projeto;

"git status" para verificar a situação atual do projeto

"git add" para pegar atualizações e adicionar novos arquivos.

"git add ." caso queira atualizar todos os arquivos de uma só vez.

"git checkout -- nomedoarquivo" para descartar mudanças no diretório de trabalho, caso eu tenha feito alguma alteração por engano e queira voltar a versão anterior.

"git rm" que vai deletar o arquivo trackeado no servidor o

"git rm -f" para de fato deletar o arquivo com a opção force.

"git commit" para enviar os arquivos e atualizações para o servidor e commitar na branch.

### Dica:

Para facilitar o trabalho de identificação do versionamento - podemos comentar o commit da seguinte maneira:

*git commit -m 'alteração que foi feita'*

Dessa forma eu já estou informando no próprio commit a alteração que foi feita.

## Git no servidor - gerando sua chave pública SSH

Muitos servidores Git se autenticam usando chaves públicas SSH.

Para fornecer uma chave pública, cada usuário em seu sistema deve gerar uma se ainda não a tiver.

Esse processo é semelhante em todos os sistemas operacionais. Primeiro, verifique se você ainda não possui uma chave.

Por padrão, as chaves SSH de um usuário são armazenadas no ~/.sshdiretório desse usuário .

Você pode verificar facilmente se já possui uma chave acessando esse diretório e listando o conteúdo:

```
$ cd ~/.ssh
$ ls
authorized_keys2  id_dsa      known_hosts
config           id_dsa.pub
```

Estamos procurando um par de arquivos chamado algo como **id\_dsa** ou **id\_rsa** e um arquivo correspondente com uma .pubextensão.

O .pubarquivo é sua chave pública e o outro arquivo é a chave privada correspondente.

Se você não possui esses arquivos (ou não possui um .sshdiretório), é possível criá-los executando um programa chamado "ssh-keygen -o", fornecido com o pacote SSH nos sistemas Linux / macOS e fornecido com o Git for Windows:

```
$ ssh-keygen -o
Generating public/private rsa key pair.
Enter file in which to save the key (/home/schacon/.ssh/id_rsa):
Created directory '/home/schacon/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/schacon/.ssh/id_rsa.
Your public key has been saved in /home/schacon/.ssh/id_rsa.pub.
The key fingerprint is:
d0:82:24:8e:d7:f1:bb:9b:33:53:96:93:49:da:9b:e3 schacon@mylaptop.local
```

Primeiro, ele confirma onde você deseja salvar a chave ( `.ssh/id_rsa`) e, em seguida, solicita uma frase-senha duas vezes, que você pode deixar em branco se não quiser digitar uma senha ao usar a chave.

No entanto, se você usar uma senha, adicione a opção; ele salva a chave privada em um formato mais resistente à quebra de senha de força bruta do que o formato padrão.

Você também pode usar a `ssh-agent-tools` para evitar ter que digitar a senha a cada vez.

Agora, cada usuário que fizer isso deve enviar sua chave pública para você ou para quem estiver administrando o servidor Git (supondo que você esteja usando uma configuração de servidor SSH que exija chaves públicas).

Tudo o que eles precisam fazer é copiar o conteúdo do `.pub`arquivo e enviá-lo por e-mail. As chaves públicas são mais ou menos assim:

```
$ cat ~/.ssh/id_rsa.pub
```

```
ssh-rsa
AAAAB3NzaC1yc2EAAAABIwAAAQEaklOUpkDHrfHY17SbrmTIpNLTGK9Tjom/
BWDSU
GPl+nafz1HDTYW7hdI4yZ5ew18JH4JW9jbhUFrviQzM7xlELEVf4h9lFX5QV
kbPppSwg0cda3
```

```
Pbv7kOdJ/  
MTyBlWXFCR+HAo3FXRitBqxiXlnKhXpHAZsMciLq8V6RjsNAQwdsdMFvSlVK  
/7XA  
  
t3FaoJoAsncMlQ9x5+3V0Ww68/  
eIFmb1zuUFljQJKprX88XypNDvjYNby6vw/Pb0rwert/En  
  
mZ+AW4OZPnTPI89ZPmVMLuayrD2cE86Z/  
il8b+gw3r3+lnKatmIkjn2sold0lQraTlMqVSsbx  
  
NrRFi9wrf+M7Q== schacon@mylaptop.local
```

## 1-Inserindo configurações de identificação GIT

Para realizar o commit inicial do repositório:

```
git config --global user.email "meu@email"
```

```
git config --global user.name "meunome"
```

## 2- Realizando a conexão com o GITHUB

Para testar se já existe alguma conexão:

```
git remote -v
```

### Para se conectar:

```
git remote add origin (vou até o projeto criado no git e copio a  
url do projeto)
```

### **3- Realizando o push do projeto que desejo enviar e de qual branch quero puxar:**

*git push -set-upstream origin master*

### **4- Realizando push de arquivos do ambiente local para o ambiente remoto**

Após a criação do diretório local e preparação para o ambiente remoto digito o comando a seguir para que os ambientes fiquem equiparados:

#### **Dica:**

Caso seja apresentado algum erro ou mensagem impossibilitando o envio do arquivos em decorrência de diferentes versões de "branches" locais e remotas, podemos tentar executar o comando abaixo, a fim de informar ao GIT que queremos desconsiderar a diferença entre versões nos ambientes;

Navego até o diretório onde criei o meu projeto e digito os seguintes comandos:

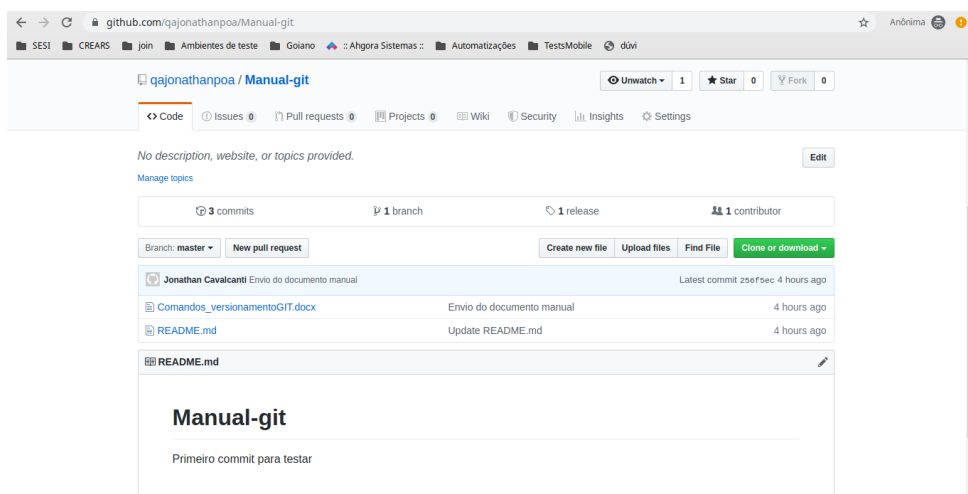
- *git init*
- *git add (nome do arquivo)*
- *git commit -m (nome do commit)*
- *git pull origin master*
- *git push origin master*

- *git checkout master*
- *git merge origin/master --allow-unrelated-histories*

## 5- Realizando o versionamento tags/releases através do GitHub

O processo de criação de tags pode ser realizado tanto por terminal quanto pelo painel do GitHub. No painel de nosso repositório criado no Github basta clicar sobre:

*nome do repositório → release → Botão nova release/tag.*



Com esse procedimento será realizada além da criação da versão do código ou arquivo disponibilizado, também, os arquivos para download de acordo com a versão que disponibilizei.

Para acompanhar essa alteração pelo terminal podemos digitar:

*git fetch*

*git pull*

Abaixo conseguimos acompanhar a atualização da tag criada no github

```
jonathan@jonathan-desktop:~/Manual-git$ git fetch
From https://github.com/qajonathanpoa/Manual-git
* [new tag]          1.0      -> 1.0
jonathan@jonathan-desktop:~/Manual-git$
```

## 6- Criação de tags/releases pelo terminal

Para definirmos a nossa release através do terminal o processo é mais simples ainda, bastando digitar através do nosso terminal os seguintes comandos seguidos de **ENTER**:

*git tag número da tag/versão*

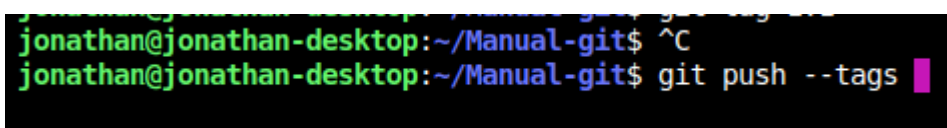
```
* [new tag]          1.0      -> 1.0
jonathan@jonathan-desktop:~/Manual-git$ ^C
jonathan@jonathan-desktop:~/Manual-git$ git tag 1.1
jonathan@jonathan-desktop:~/Manual-git$
```

*Definindo a criação da tag 1.1 em nosso repositório*



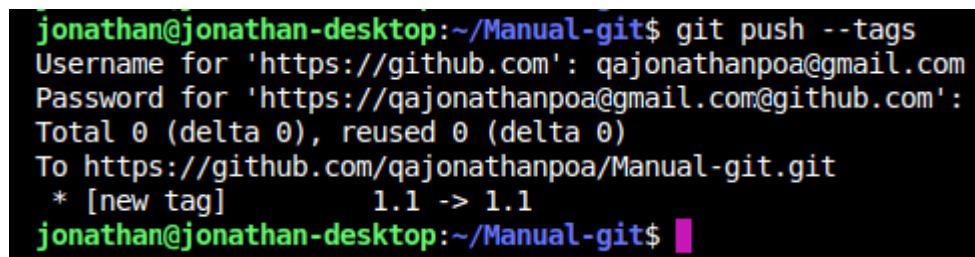
Em seguida podemos digitar o comando abaixo para enviarmos para nosso servidor GitHub a definição de nossa tag/release criada:

```
git push -tags
```

A terminal window with a black background and green text. The prompt is 'jonathan@jonathan-desktop:~/Manual-git\$'. The user has entered '^C' to clear the line, and then 'git push --tags'. A pink cursor is visible at the end of the command.

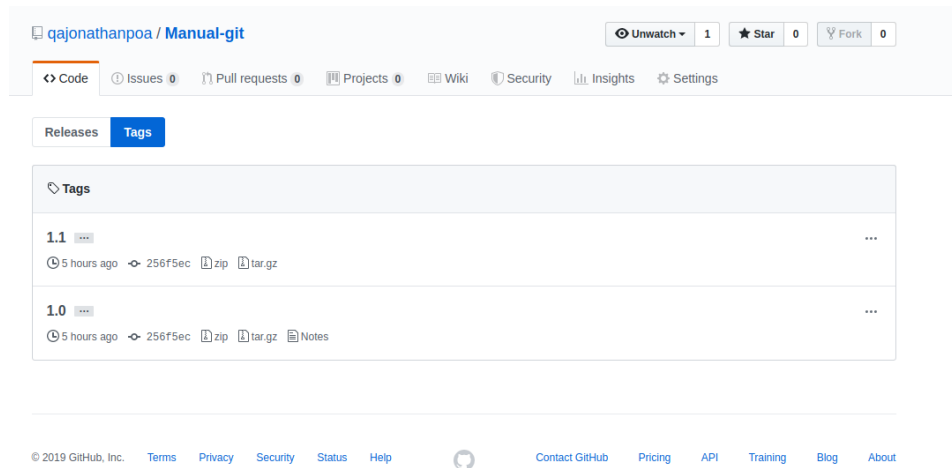
```
jonathan@jonathan-desktop:~/Manual-git$ ^C
jonathan@jonathan-desktop:~/Manual-git$ git push --tags
```

Serão solicitados usuário e senha e com isso criamos com sucesso a tag de versão em nosso repositório GitHub.

A terminal window with a black background and green text. The prompt is 'jonathan@jonathan-desktop:~/Manual-git\$'. The user enters 'git push --tags'. The terminal shows the following output: 'Username for 'https://github.com': qajonathanpoa@gmail.com', 'Password for 'https://qajonathanpoa@gmail.com@github.com':', 'Total 0 (delta 0), reused 0 (delta 0)', 'To https://github.com/qajonathanpoa/Manual-git.git', '\* [new tag] 1.1 -> 1.1'. The prompt returns to 'jonathan@jonathan-desktop:~/Manual-git\$'. A pink cursor is visible at the end of the prompt.

```
jonathan@jonathan-desktop:~/Manual-git$ git push --tags
Username for 'https://github.com': qajonathanpoa@gmail.com
Password for 'https://qajonathanpoa@gmail.com@github.com':
Total 0 (delta 0), reused 0 (delta 0)
To https://github.com/qajonathanpoa/Manual-git.git
 * [new tag] 1.1 -> 1.1
jonathan@jonathan-desktop:~/Manual-git$
```

Se acessarmos o nosso repositório no GitHub conseguimos observar com sucesso a criação de nossa Tag pelo terminal bem como seus respectivos arquivos:



## 7- GIT Branchs- Criando Branchs pelo GitHub

As branches são criadas para preservar a integridade da “master” de nossos projetos e também para facilitar a manutenção em nossos códigos que fazem parte do projeto.

Geralmente nós chamamos as alterações de “*features*” quando as branches se referem a novas funcionalidades, ou o termo que acharmos melhor de acordo com a necessidade.

E denominamos essa alteração ou correção no Git para identificar a branch:

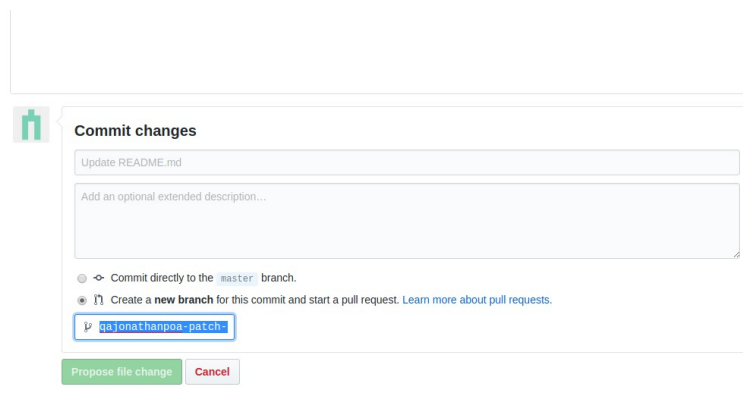
***As branches vêm da Master e por esse motivo as Branchs herdam os commits da Master também!***

**feature/”nome da alteração/branch”**

A criação da branch bem como as tags/releases também podem ser criadas pelo terminal ou pelo painel do GitHub.

Pelo painel do GitHub basta selecionarmos o nosso repositório, realizar alguma alteração e clicar em "Criar uma nova branch", opção disponível

próximo ao rodapé da página e em seguida definimos o nome desejado:



## 8- GIT Branchs- Criando Branchs pelo terminal

A criação de uma Branch pelo terminal é extremamente simples, basta abriremos o nosso terminal e digitarmos o comando a seguir:

*git branch nome da branch*

```
jonathan@jonathan-desktop:~$ cd Manual-git/  
jonathan@jonathan-desktop:~/Manual-git$ git branch nome da branch
```

### Dica:

Para deletar uma Branch pelo terminal basta digitarmos o seguinte comando:

*git branch -D nome da branch que quero deletar*

```
nenhuma modificação adicionada à submissão (utilize "git add" e/ou "git commit -a")
jonathan@jonathan-desktop:~/Manual-git$ git branch -D nome da branch
```

## 9- GIT checkout

O comando git checkout serve para trocarmos de branch em nosso projeto através do terminal.

Se por acaso viermos a criar uma nova branch pelo terminal e desejamos acessar a branch criada ou até mesmo realizar push para essa nova branch devemos digitar os seguintes comandos abaixo:

*gitbranch- (para exibir na coloração verde qual branch está ativa no momento)*

```
jonathan@jonathan-desktop:~/Manual-git$ git branch
feature/atualizacao-do-readme
* feature/atualizacao_do_github
master
jonathan@jonathan-desktop:~/Manual-git$
```

O comando abaixo irá definir a branch iremos trabalhar. Caso eu tenha feito alguma alteração em uma branch anterior e tente

conectar em uma nova branch, o GIT irá recriminar informando que devo commitar as minhas alterações antes de realizar essa troca.

*git checkout (nome da branch)*

Caso eu tenha criado a branch apenas no GitHub, basta digitar o mesmo comando acima seguido do nome da branch criada

Com isso o GIT irá atualizar as branches e permitir pushes e pulls futuros.

```
nenhuma modificação adicionada à submissão (utilize "git add" e/ou "git commit -a")
jonathan@jonathan-desktop:~/Manual-git$ git checkout feature/atualizacao_do_github
```

Em seguida podemos digitar o comando "git branch" que o terminal irá exibir na cor **verde** a branch na qual estamos trabalhando:

```
* feature/mudancas
master
jonathan@jonathan-desktop:~/Manual-git$
```

## 10- GIT Merge

O GIT Merge serve justamente para juntarmos o código da branch (com as novas alterações), código o qual fizemos cópia da **master** anterior as alterações, com o código original da Master.

Antes de realizarmos o Merge de uma branch para a **Master**, devemos dar um "git checkout master" para acessarmos na **Master** ou na **Branch** a qual queremos que herde as alterações feitas, fazendo um papel de **master**.

## EX:

Se eu quero definir que a branch "**feature/atualizacao\_git\_hub**" seja a master, basta eu rodar o comando para realizar o **switch** entre branches:

```
git checkout feature/atualizacao_git_hub
```

```
branch 'master' set up to track remote branch 'master' from 'origin'.  
jonathan@jonathan-desktop:~/Manual-git$ git checkout feature/atualizacao_do_github
```

Em seguida posso rodar o comando *git merge* normalmente

No exemplo abaixo como estamos na **Master**, basta digitarmos o seguinte comando:

Ou seja, gitmerge e o nome da branch na qual desejamos realizar o merge.

```
git merge feature/atualizacao_do_github
```

Em seguida podemos dar o comando

```
git push
```

Com isso a **Master** e a nossa **Branch** ficarão no mesmo nível no GitHub.

Após o término do Merge eu posso ir até o painel do **GitHub** e excluir a **branch** criada pois já está feito o Merge com a Master e não precisarei mais dela.

Esse processo deve ser feito tanto no **GitHub** quanto no terminal utilizando o comando:

### **No terminal:**

*git branch -D nome da branch a ser deletada*

Após esse procedimento eu posso também já criar uma nova Branch com o comando

*git checkout -b nome da nova branch*

Essa branch será criada já com os commits da Master e com isso eu já posso iniciar o meu trabalho de versionamento.

Ou seja ela será um clone da master pós Merge.

## **11- GIT Rebase**

O comando GIT "rebase" deve ser utilizado quando existe a situação de termos um commit feito por outra pessoa ou branch, na Master.

Com isso nós encontramos uma situação de conflito pois a Master estará diferente da nossa branch. Nesse caso devemos seguir o seguinte procedimento:

Podemos confirmar que estamos na branch desejada com o comando gitbranch:

*git branch*

```
jonathan@jonathan-desktop:~/Manual-git$ git checkout feature/branchA
M      Comandos_versionamentoGIT.docx
Switched to branch 'feature/branchA'
Your branch is up to date with 'origin/feature/branchA'.
jonathan@jonathan-desktop:~/Manual-git$ git branch
* feature/branchA
  master
jonathan@jonathan-desktop:~/Manual-git$
```

Após o procedimento de alteração de branch, posso realizar o commit da branch na qual desejo basear com a master digitando o seguinte comando:

### **Na branch a qual me encontro:**

*git commit -m "commit da branch a"*

Para atualizar o meu branch local, posso digitar o comando:

*git fetch*

Com isso iremos perceber que a nossa branch está diferente da Master, pois a Master está com um commit a mais do que nossa branch.

Nesse momento podemos utilizar o comando gitrebase:

*1- git rebase master*

*2- git checkout master*

*3- git pull*

*4- git checkout (branch a qual estou trabalhando)*



5- *git rebase master*

6- Com a mensagem do git informando que os conflitos estão corrigidos posso rodar o comando:

*git rebase --continue*

## 12- GIT Cherry-pick

O comando **cherry-pick** serve para passarmos commits específicos de uma branch para outra.

Caso comum quando realizamos várias alterações em uma branch e desejamos escolher apenas uma alteração em especial.

Para obtermos sucesso no comando cherry-pick podemos seguir os seguintes passos:

1- Devemos realizar o comando *"git fetch"* para se certificar que a branch local está sincronizada com a remota do GitHub.

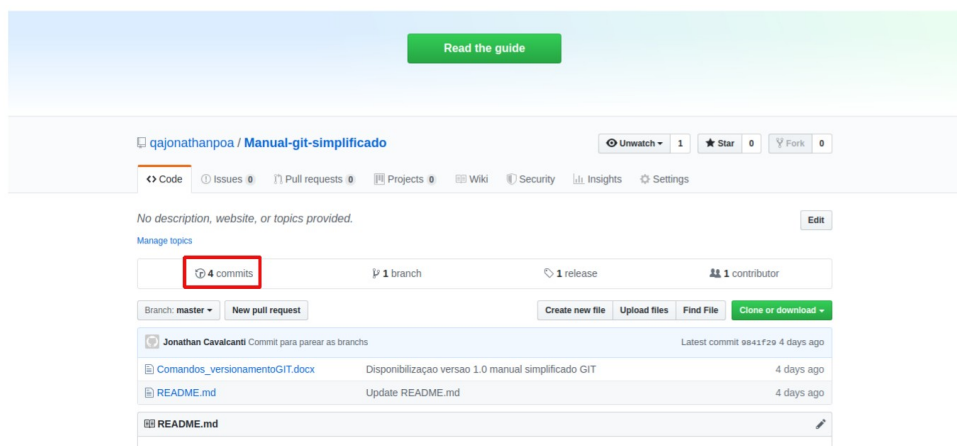
2- Acessar a branch master com o comando *"git checkout master"*

3- Digitar o comando *"git cherry-pick nome da branch a qual desejamos pegar o commit, seguido do hash"*. Como no exemplo abaixo digitado no nosso terminal:

```
jonathan@jonathan-desktop:~/Manual-git$ git cherry-pick feature/mudancas 9841f29f3fce6daa5d12de99c85a4eadd68605b6
```

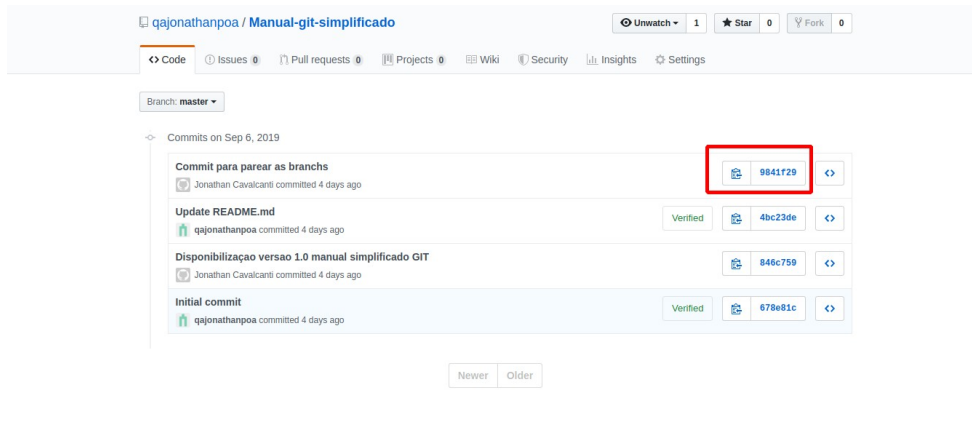
No comando acima eu estou querendo pegar um commit da branch "feature/mudancas" com o seu respectivo hash.

O hash no **GitHub**, fica localizado no painel de nossos repositórios, clicando sobre a opção **commits**, dentro da branch desejada:

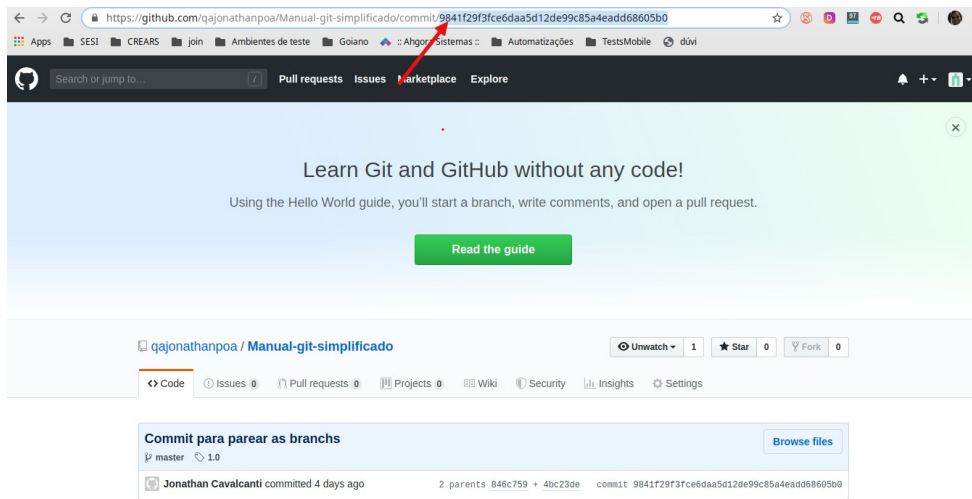


4- O hash é a numeração que se encontra ao lado dos commits de acordo com a imagem abaixo.

Nesse caso devemos clicar sobre o hash desejado;



5- Em seguida devemos selecionar a numeração após a barra commit, de acordo com a imagem abaixo e colar dentro de nosso terminal após o comando do cherry-pick;



Se tudo ocorrer corretamente ao digitarmos o comando *"git log"* veremos como último commit a branch que nós realizamos o cherry-pick.

## 13- GIT Reset

O comando GIT reset serve para voltarmos o commit de nossa branch para uma versão específica, ou seja, para excluirmos commits que não desejamos em nossa branch.

1- Para definirmos qual será branch que queremos definir como HEAD de nossos commits, basta digitarmos o seguinte comando:

```
git reset a269e60cc384768d8e0646554f7d30215a8c84ec
```

A terminal window with a dark purple background. The first line shows a git commit hash and the HEAD pointing to master. The second line shows the author and date. The third line is a comment in Portuguese. The fourth line shows the new git commit hash after the reset. The fifth line shows the author and date. The sixth line is another comment in Portuguese.

```
commit 8c54bcecf2b17f360ec926744b9e29a5ba701abc (HEAD -> master, origin/master)
Author: Jonathan Cavalcanti <jonathan.cavalcanti@jointecnologia.com.br>
Date:   Fri Sep 13 15:33:50 2019 -0300

    commit para ajustar a master sem stash

commit a269e60cc384768d8e0646554f7d30215a8c84ec
Author: Jonathan Cavalcanti <jonathan.cavalcanti@jointecnologia.com.br>
Date:   Fri Sep 13 14:48:51 2019 -0300

    o stash vai apagar o título
```

*Na imagem acima eu quero que o commit HEAD seja o que possui o texto – “o stash vai apagar o título”.*

Ou seja, basta digitarmos o comando git reset + o número hash do commit desejado.

**Importante:** O reset não faz o trabalho de apagar ou excluir o nosso trabalho, ele apenas retorna versões de nossos commits dentro da branch.

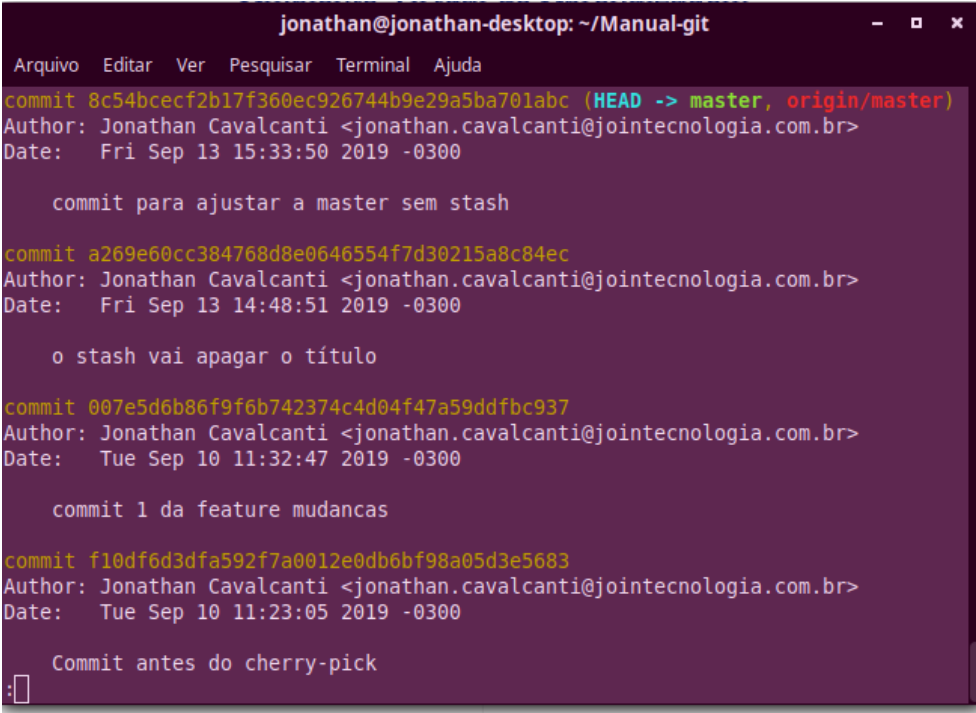
2- Após o processo de *reset* de nossa branch é normal o **GitHub** nos informar que estamos com uma versão atrás da versão remota, bastando então digitarmos o comando:

```
git push -f
```

## Reset através de parâmetros

Também conseguimos executar um reset utilizando parâmetros dentro de nossa branch.

1- Digitamos *git log* para analisarmos nossos commits:

A screenshot of a terminal window titled 'jonathan@jonathan-desktop: ~/Manual-git'. The terminal shows the output of the 'git log' command. The output lists four commits in reverse chronological order. Each commit entry includes a commit hash, a message in parentheses, the author's name and email, and the date and time. The commit messages are: 'commit para ajustar a master sem stash', 'o stash vai apagar o título', 'commit 1 da feature mudancas', and 'Commit antes do cherry-pick'. The terminal has a menu bar with 'Arquivo', 'Editar', 'Ver', 'Pesquisar', 'Terminal', and 'Ajuda'.

```
jonathan@jonathan-desktop: ~/Manual-git
Arquivo  Editar  Ver    Pesquisar  Terminal  Ajuda

commit 8c54bcecf2b17f360ec926744b9e29a5ba701abc (HEAD -> master, origin/master)
Author: Jonathan Cavalcanti <jonathan.cavalcanti@jointecnologia.com.br>
Date:   Fri Sep 13 15:33:50 2019 -0300

    commit para ajustar a master sem stash

commit a269e60cc384768d8e0646554f7d30215a8c84ec
Author: Jonathan Cavalcanti <jonathan.cavalcanti@jointecnologia.com.br>
Date:   Fri Sep 13 14:48:51 2019 -0300

    o stash vai apagar o título

commit 007e5d6b86f9f6b742374c4d04f47a59dddfbc937
Author: Jonathan Cavalcanti <jonathan.cavalcanti@jointecnologia.com.br>
Date:   Tue Sep 10 11:32:47 2019 -0300

    commit 1 da feature mudancas

commit f10df6d3dfa592f7a0012e0db6bf98a05d3e5683
Author: Jonathan Cavalcanti <jonathan.cavalcanti@jointecnologia.com.br>
Date:   Tue Sep 10 11:23:05 2019 -0300

    Commit antes do cherry-pick
:
```

2- Se eu quiser definir que a minha HEAD será "commit 1 da feature mudancas", basta digitar o comando a seguir:

```
git reset HEAD~2
```

Com isso o git irá definir o meu commit HEAD da mesma forma se digitássemos git reset + hash.

## **14- GIT Stash**

O comando GIT stash serve para criarmos um ponto de salvamento temporário em nossa branch a fim de guardar o estado de nossos arquivos e permitir que seja possível continuar nosso trabalho de onde paramos.

O **GIT** cria um ponto paralelo ao ponto de commit atual da branch, onde serão guardadas as últimas informações que não estão presentes no último commit do branch.

1- Para criarmos um ponto de salvamento utilizando o GIT stash devemos digitar os seguintes comandos:

```
git stash save "nome do ponto de salvamento desejado"
```

2- Para realizarmos o uso do ponto de partida criado pelo stash podemos digitar os seguintes comandos:

```
git stash list – para vermos os "stashes" criados
```

```
git stash apply stash – número do stash desejado
```

**Exclusão de stashes criados:**

*git stash drop stash – número do stash desejado*

## **Criando branches utilizando stashes**

*git stash branch new\_branch*

Ainda falando do git stash save também é possível adicionar a pilha, os arquivos não versionados pelo git(untracked files), bastando apenas passar a flag *--include-untracked*.

Com essa flag o git irá adicionar a pilha, e limpar a árvore de trabalho.

Ao usar o stash, pode-se perceber que o diretório principal ficará limpo.

Ou seja, os arquivos acompanhados não irão mais aparecer na lista. Vale ressaltar que se o usuário deseja adicionar tantos os arquivos acompanhados como não acompanhados ao stash, deve usar o seguinte comando:

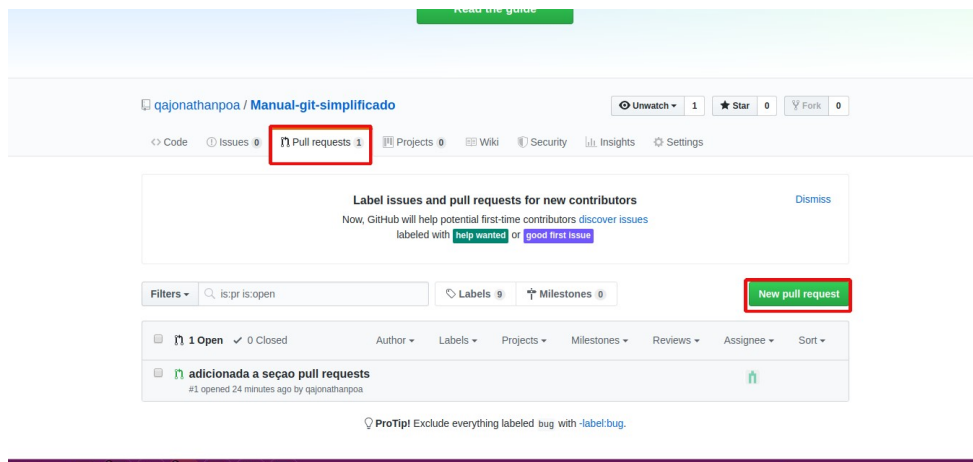
*git stash --all*

## **15- Como criar um pull request**

O processo de pull request consiste em fazer o pedido de uma branch criada localmente, para a inserção a branch master.

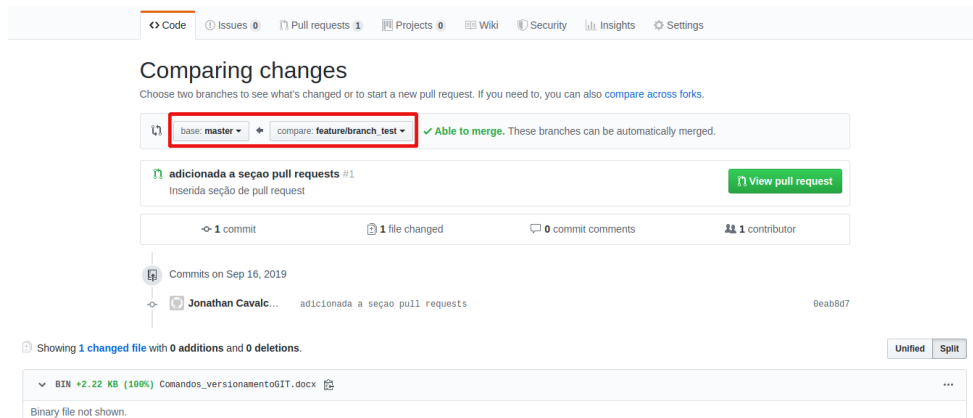
Que é atualizar os commits criados pela branch criada localmente com o mesmo nível da **MASTER**.

1- Devemos acessar o nosso painel do **GitHub** e clicarmos sobre a branch desejada e em seguida sobre o botão pull requests/new pull requests



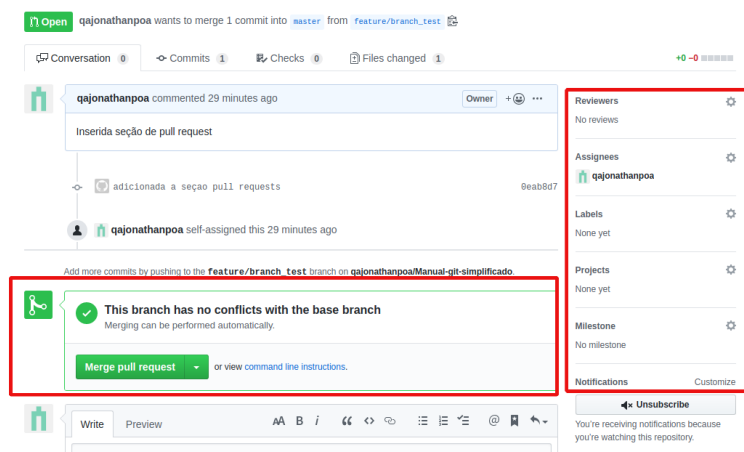
2- Em seguida devemos selecionar qual branch será a branch que desejamos comparar:





Com isso podemos clicar sobre a opção “Send pull request” para criarmos de fato nosso pull request para revisão e demais etapas.

3- Se tudo ocorrer normalmente veremos as opções “Merge pull request” habilitada, bem como as opções de adicionarmos revisores e demais etapas ao nosso merge.



E assim conclui-se a etapa de trabalho e manipulação do **GIT** em nosso repositório.

**INTRODUÇÃO AO**

**GIT-FLOW**

O intuito do Git-flow é facilitar e diminuir a quantidade de comandos digitados nas operações do GIT, como, push, criação de branches e etc.

1- Para iniciar o Git-flow no projeto basta digitar o seguinte comando:

*git flow init*

O Git-flow procurará se a branch "develop" está criada em nosso projeto e caso não esteja irá nos orientar a realizar a criação da branch mencionada com o comando:

*git checkout -b develop*

## **Comandos e atribuições**

*git flow feature start nomedabbranch*

- Cria uma branch (git checkout -b) baseada na branch DEVELOP;
- Automaticamente nomeia com o padrão feature/NOMEDABRANCH;
- Passa a usar a nova branch (git checkout feature/NOMEBRANCH);
- 
- *git flow feature finish nomedabbranch*
- 
- Realiza merge com a feature branch NOMEDABRANCH dentro da Develop (git checkout develop);
- Git checkout develop;
- Git merge NOMEBRANCH;
- Deleta a feature branch (git branch -D NOMEDABRANCH);
- E muda para a branch Develop;

*git flow feature publish nomedabbranch*

- Cria uma branch remota (git push –set-upstream origin feature/NOMEDABRANCH);

*git flow release start nomedabbranch*

- Cria uma branch (git checkout -b) baseada na DEVELOP;
- Automaticamente nomeia com release/NOMEDABRANCH;
- Passa a usar a nova branch (git checkout release/NOMEDABRANCH);

*git flow release finish nomedabbranch*

- Merge da release branch;
- NOMEDABRANCH dentro da master – (git checkout master; git merge NOMEDABRANCH);
- Cria uma tag com o nome release/NOMEDABRANCH (git tag release/NOMEDABRANCH);
- Merge a release/NOMEDABRANCH na develop – git checkout develop; git merge release/NOMEDABRANCH;
- Deleta a release branch – git branch -D release/NOME DA BRANCH;
- Muda para a develop;

*git flow release publish nome da branch*

- Cria uma branch remota (git push –set-upstream origin release/NOMEDABRANCH);

*git flow hotfix start nomedabbranch*

- Cria uma branch – git checkout -b (baseada na master);
- Automaticamente nomeia com hotfix/NOMEDABRANCH;

- Passa a usar a nova branch – git checkout hotfix/NOMEDABRANCH;

*git flow hotfix finish nomedabbranch*

- Merge a hotfix branch;
- NOMEDABRANCH dentro da master – (git checkout master; git merge NOMEDABRANCH);
- Cria uma tag com o nome hotfix/NOMEDABRANCH (git tag hotfix/NOMEDABRANCH) na master;
- Merge a hotfix/NOMEDABRANCH na develop – git checkout develop; git merge hotfix/NOMEDABRANCH;
- Merge a hotfix/NOMEDABRANCH na master – git checkout master; git merge hotfix/NOMEDABRANCH;
- Deleta a hotfix branch – git branch -D hotfix/NOMEDABRANCH;
- Muda para a develop – git checkout develop;

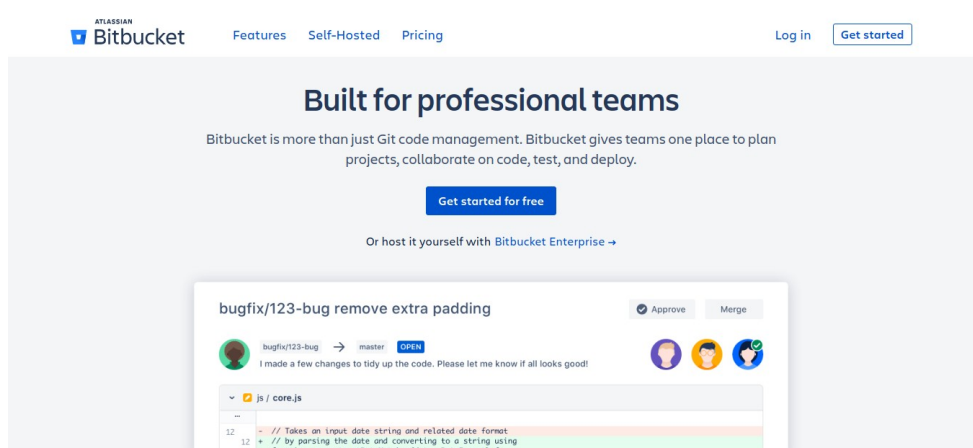
## **Utilizando o BitBucket**

O BitBucket assim como o GitHub é um gerenciador de repositórios que podemos utilizar com o Git.

Para acessarmos e utilizarmos esse maravilhoso gerenciador devemos acessar o endereço a seguir:

<https://bitbucket.org/>

Acessando o portal do BitBucket devemos clicar na opção Get Started e criar uma conta, a fim de podermos definir nossa chave pública SSH.



## Observação:

Uma grande vantagem do **BitBucket** é que podemos realizar a criação de nossos repositórios de maneira privada, sem a necessidade de fazer assinaturas, algo que no GitHub se torna necessário.

## Ferramentas para uso do Git via GUI

Podemos utilizar o GIT sem o uso de linha de comando também e aqui estão algumas ferramentas disponíveis na internet que podemos utilizar e agregando ao nosso dia a dia.

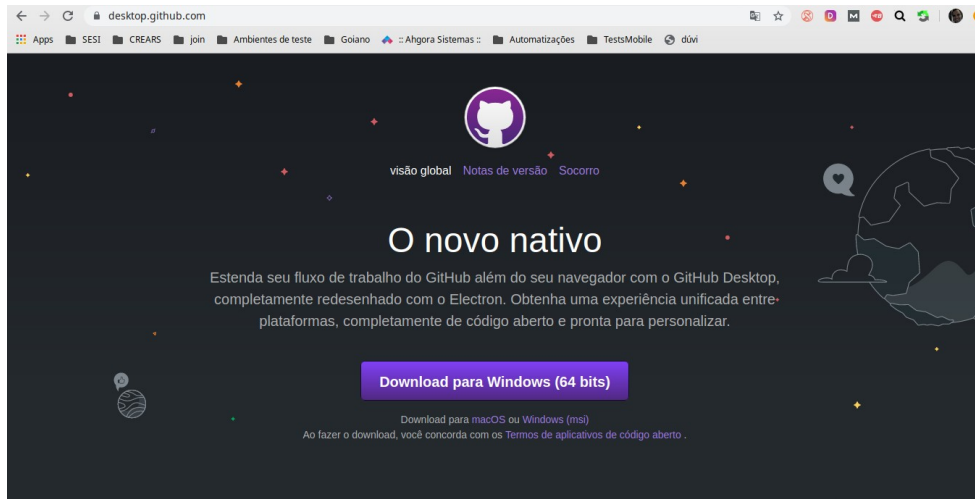
### 1- GitHub Desktop

Estenda seu fluxo de trabalho do GitHub além do seu navegador com o GitHub Desktop, redesenhado com o Electron.

Obtenha uma experiência unificada entre plataformas, completamente de código aberto e pronta para personalizar.

O **GitHub** desktop é uma ferramenta gratuita e super intuitiva que está disponível para Windows e para MAC.

Link para download: <https://desktop.github.com/>



## 2- GitKraken

O GitKraken é outra ferramenta GUI que promete a diminuição do uso de linha de comando no GIT.

Ele possui uma versão gratuita, mas a versão mais completa e robusta do software é paga. Porém para conhecer podemos utilizar a versão gratuita.

É um programa um pouco mais difícil de entender, devido a grande quantidade de ferramentas que o software possui, mas com uso no dia a dia se torna simples e fácil o domínio da ferramenta.

Disponível nas plataformas Windows, Linux e MAC.

Link para download: <https://www.gitkraken.com/download>

### **3- Git Extensions**

O **Git Extensions** é outra opção muito boa para utilizarmos via GUI.

A sua instalação é um pouco mais complicada e requer algumas configurações, mas no próprio hub do GitHub temos acesso ao seu manual, onde conseguimos acompanhar dicas de instalação, bem como de uso da ferramenta.

A sua interface é um pouco mais técnica, mas com o uso no dia a dia também se torna algo fácil e rápido de se dominar.

Disponível nas plataformas Windows, Linux e MAC.

Link para download:

<https://github.com/gitextensions/gitextensions/releases/tag/v3.2.1..>