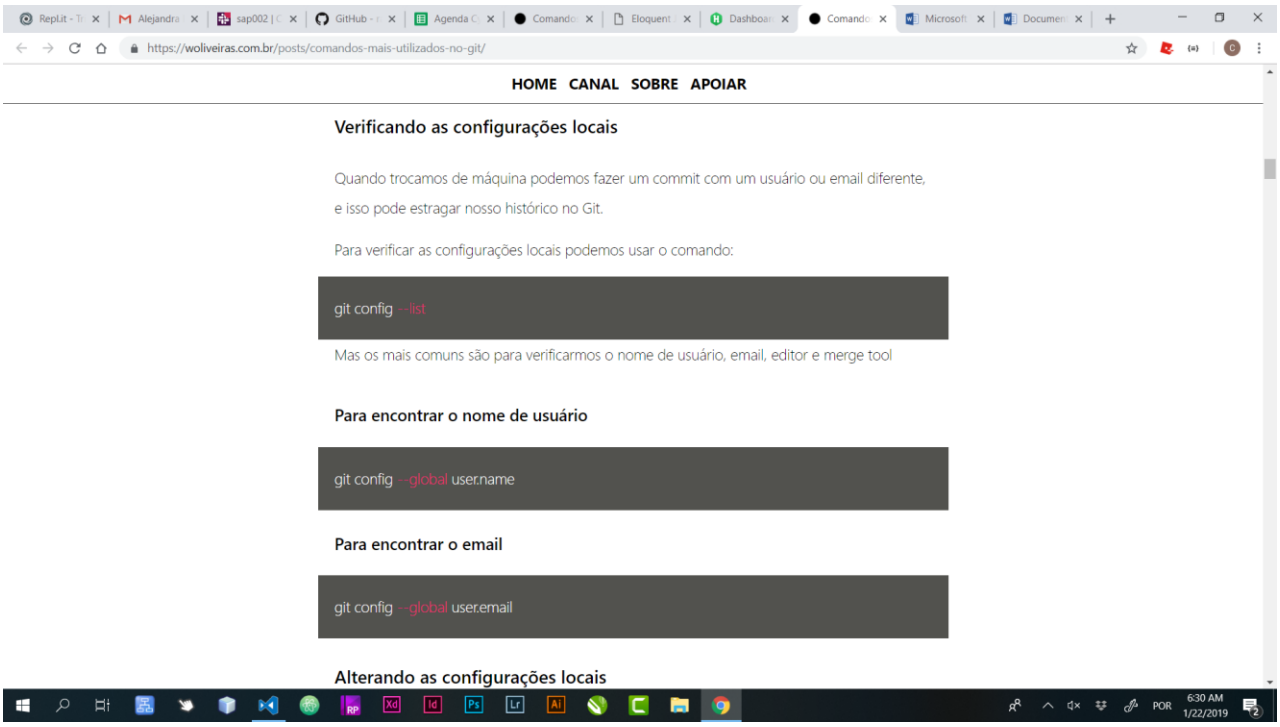
Comandos mais utilizados no git



https://woliveiras.com.br/posts/comandos-mais-utilizados-no-git/

HOME CANAL SOBRE APOIAR

Verificando as configurações locais

Quando trocamos de máquina podemos fazer um commit com um usuário ou email diferente, e isso pode estragar nosso histórico no Git.

Para verificar as configurações locais podemos usar o comando:

```
git config --list
```

Mas os mais comuns são para verificarmos o nome de usuário, email, editor e merge tool

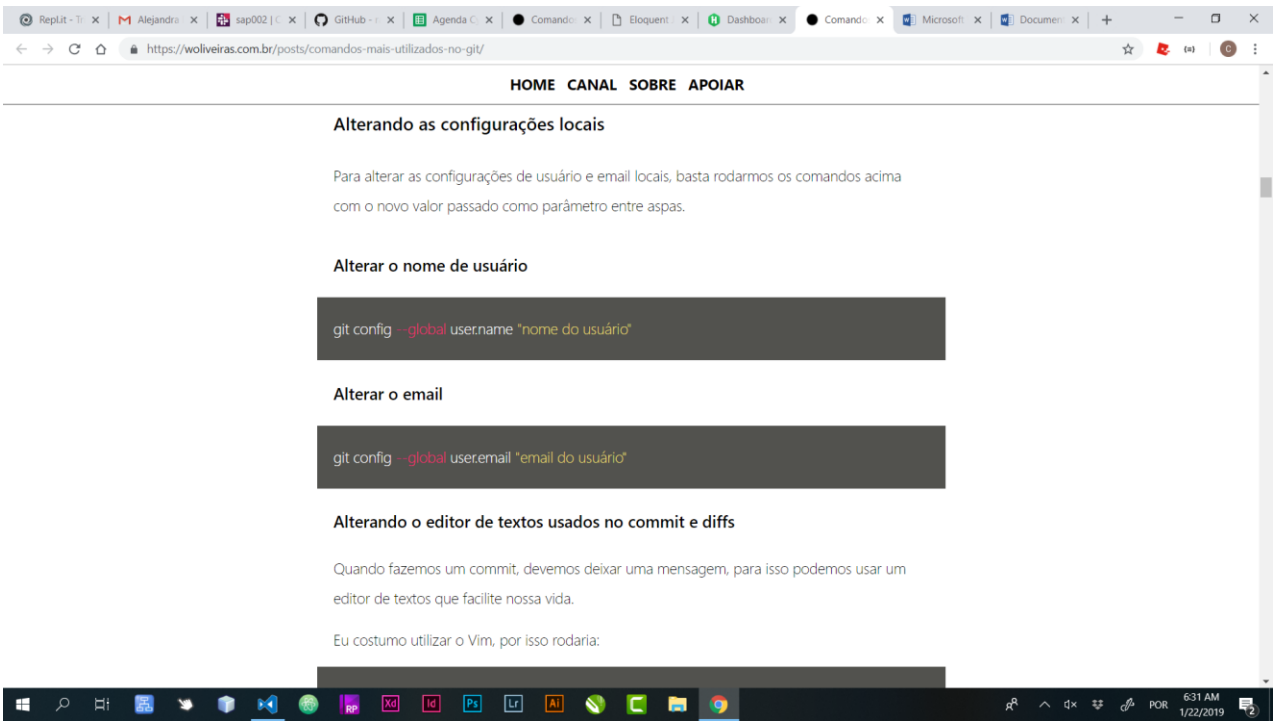
Para encontrar o nome de usuário

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

Para encontrar o email

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

Alterando as configurações locais



https://woliveiras.com.br/posts/comandos-mais-utilizados-no-git/

HOME CANAL SOBRE APOIAR

Alterando as configurações locais

Para alterar as configurações de usuário e email locais, basta rodarmos os comandos acima com o novo valor passado como parâmetro entre aspas.

Alterar o nome de usuário

```
git config --global user.name "nome do usuário"
```

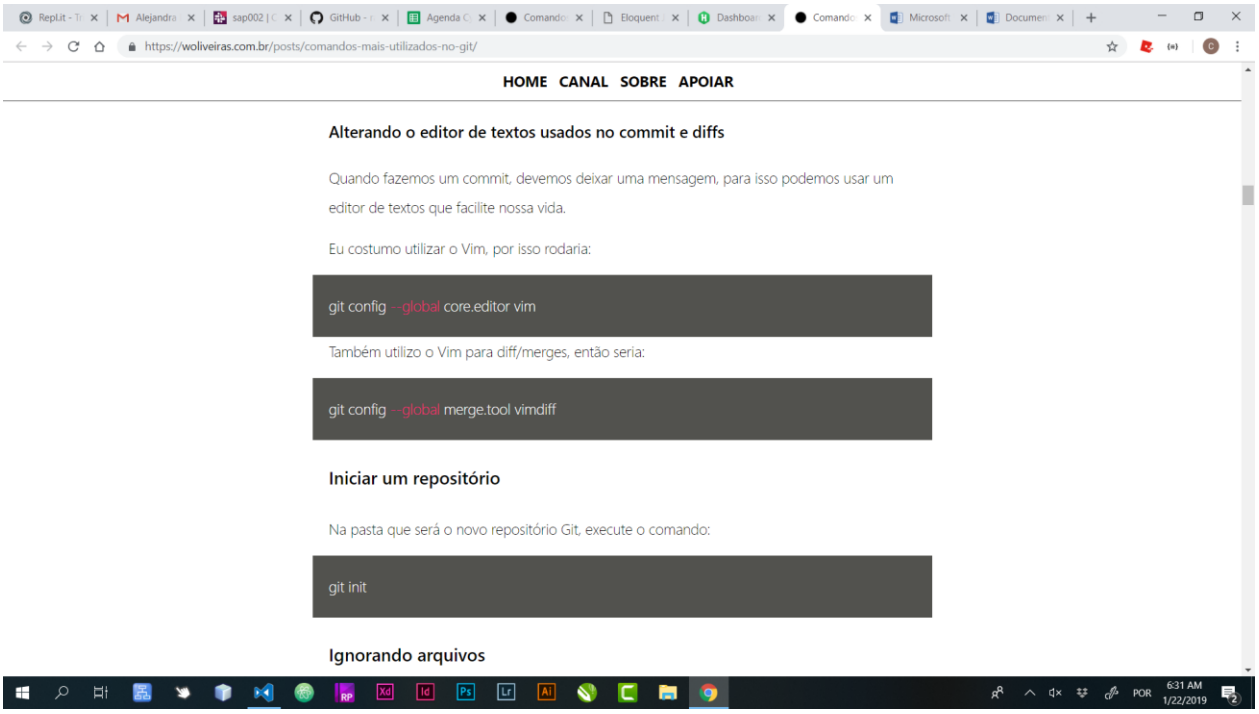
Alterar o email

```
git config --global user.email "email do usuário"
```

Alterando o editor de textos usados no commit e diffs

Quando fazemos um commit, devemos deixar uma mensagem, para isso podemos usar um editor de textos que facilite nossa vida.

Eu costumo utilizar o Vim, por isso rodaria:



https://woliveiras.com.br/posts/comandos-mais-utilizados-no-git/

HOME CANAL SOBRE APOIAR

Alterando o editor de textos usados no commit e diffs

Quando fazemos um commit, devemos deixar uma mensagem, para isso podemos usar um editor de textos que facilite nossa vida.

Eu costumo utilizar o Vim, por isso rodaria:

```
git config --global core.editor vim
```

Também utilizo o Vim para diff/merges, então seria:

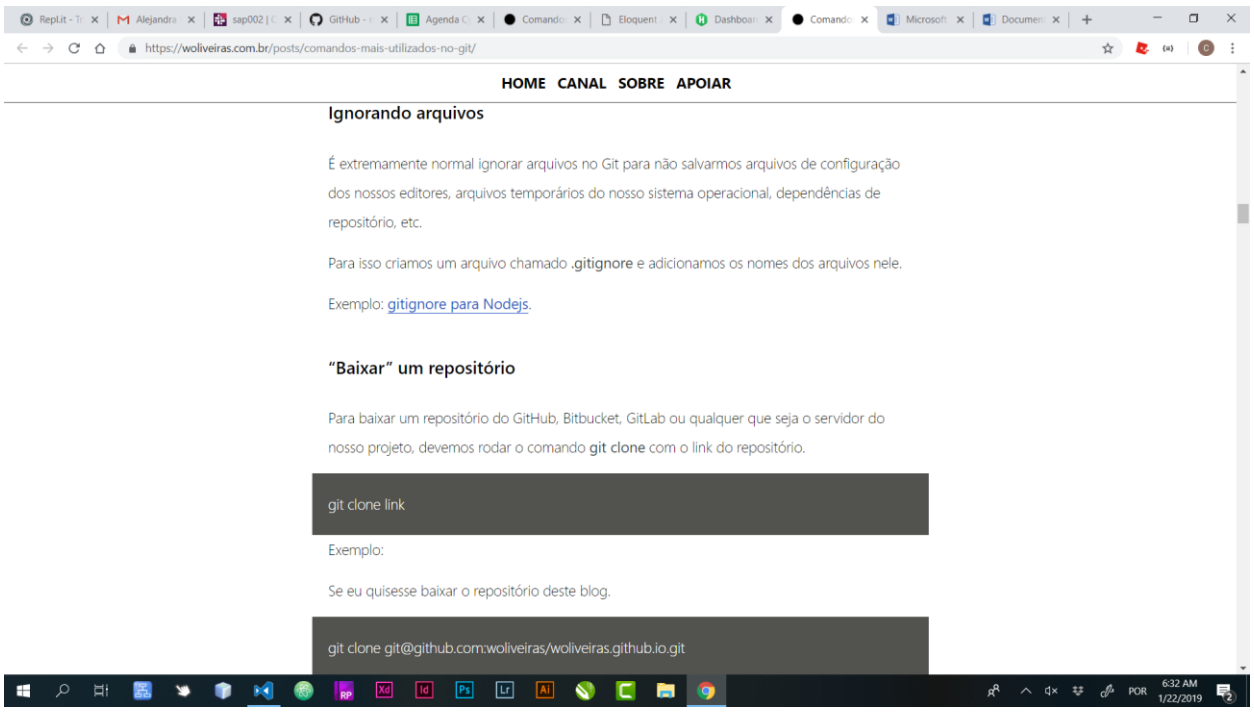
```
git config --global merge.tool vimdiff
```

Iniciar um repositório

Na pasta que será o novo repositório Git, execute o comando:

```
git init
```

Ignorando arquivos



https://woliveiras.com.br/posts/comandos-mais-utilizados-no-git/

HOME CANAL SOBRE APOIAR

Ignorando arquivos

É extremamente normal ignorar arquivos no Git para não salvarmos arquivos de configuração dos nossos editores, arquivos temporários do nosso sistema operacional, dependências de repositório, etc.

Para isso criamos um arquivo chamado **.gitignore** e adicionamos os nomes dos arquivos nele.

Exemplo: gitignore para Nodejs.

"Baixar" um repositório

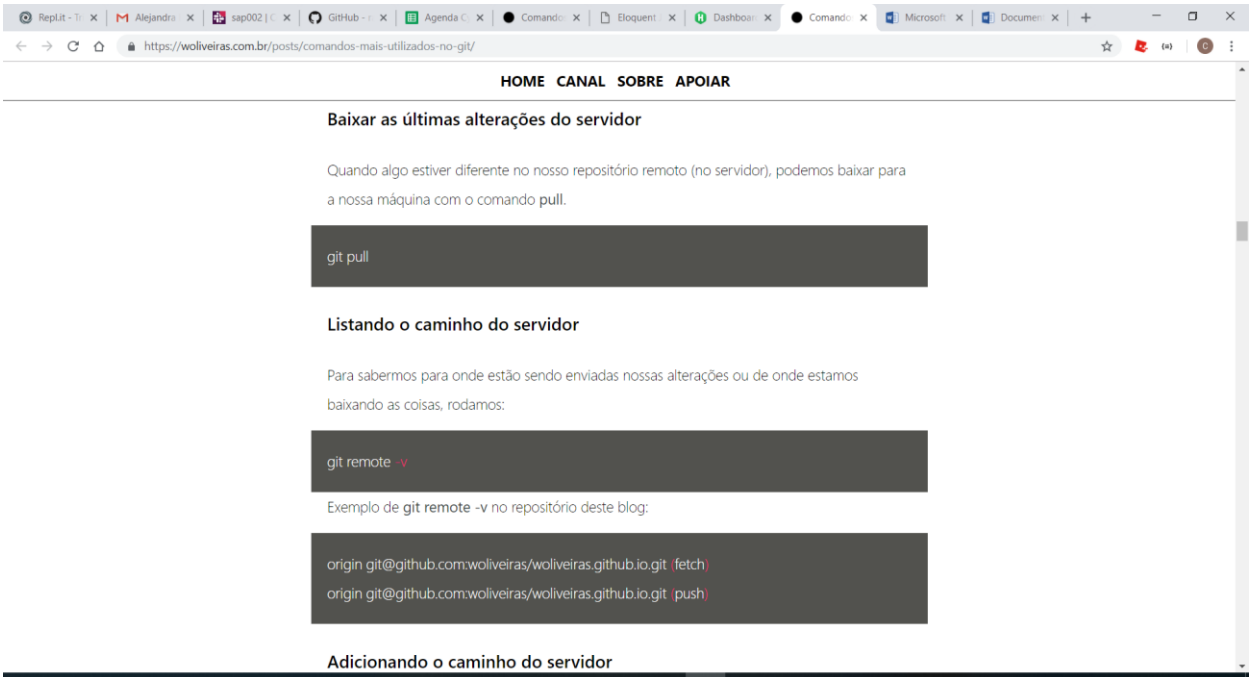
Para baixar um repositório do GitHub, Bitbucket, GitLab ou qualquer que seja o servidor do nosso projeto, devemos rodar o comando **git clone** com o link do repositório.

```
git clone link
```

Exemplo:

Se eu quisesse baixar o repositório deste blog.

```
git clone git@github.com:woliveiras/woliveiras.github.io.git
```



https://woliveiras.com.br/posts/comandos-mais-utilizados-no-git/

HOME CANAL SOBRE APOIAR

Baixar as últimas alterações do servidor

Quando algo estiver diferente no nosso repositório remoto (no servidor), podemos baixar para a nossa máquina com o comando **pull**.

```
git pull
```

Listando o caminho do servidor

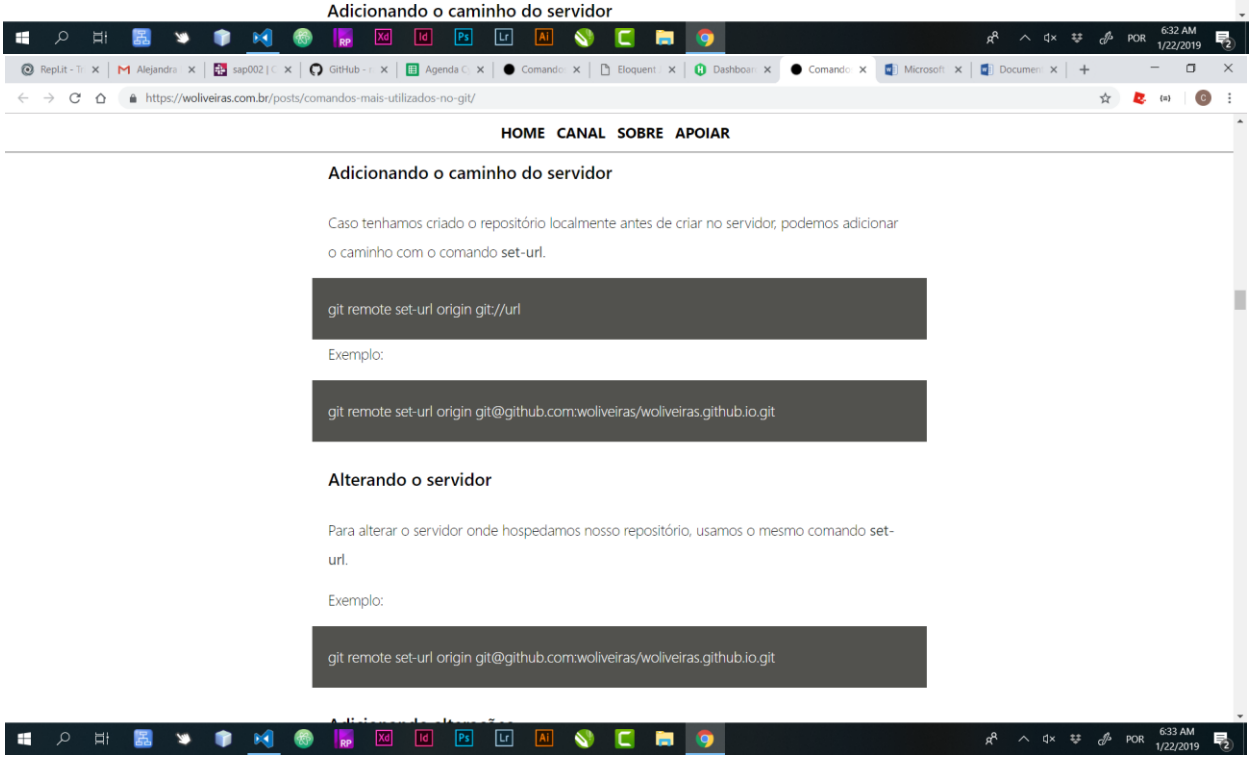
Para sabermos para onde estão sendo enviadas nossas alterações ou de onde estamos baixando as coisas, rodamos:

```
git remote -v
```

Exemplo de **git remote -v** no repositório deste blog:

```
origin git@github.com:woliveiras/woliveiras.github.io.git (fetch)
origin git@github.com:woliveiras/woliveiras.github.io.git (push)
```

Adicionando o caminho do servidor



https://woliveiras.com.br/posts/comandos-mais-utilizados-no-git/

HOME CANAL SOBRE APOIAR

Adicionando o caminho do servidor

Caso tenhamos criado o repositório localmente antes de criar no servidor, podemos adicionar o caminho com o comando **set-url**.

```
git remote set-url origin git://url
```

Exemplo:

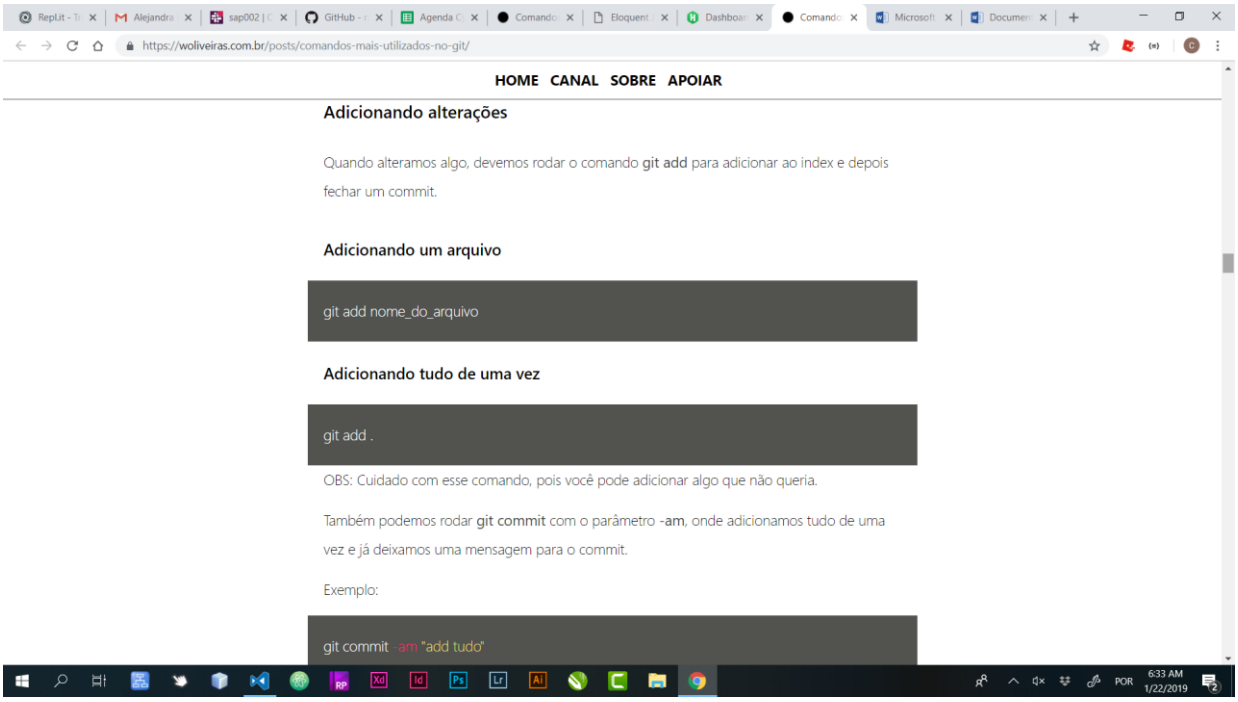
```
git remote set-url origin git@github.com:woliveiras/woliveiras.github.io.git
```

Alterando o servidor

Para alterar o servidor onde hospedamos nosso repositório, usamos o mesmo comando **set-url**.

Exemplo:

```
git remote set-url origin git@github.com:woliveiras/woliveiras.github.io.git
```



https://woliveiras.com.br/posts/comandos-mais-utilizados-no-git/

HOME CANAL SOBRE APOIAR

Adicionando alterações

Quando alteramos algo, devemos rodar o comando **git add** para adicionar ao index e depois fechar um commit.

Adicionando um arquivo

```
git add nome_do_arquivo
```

Adicionando tudo de uma vez

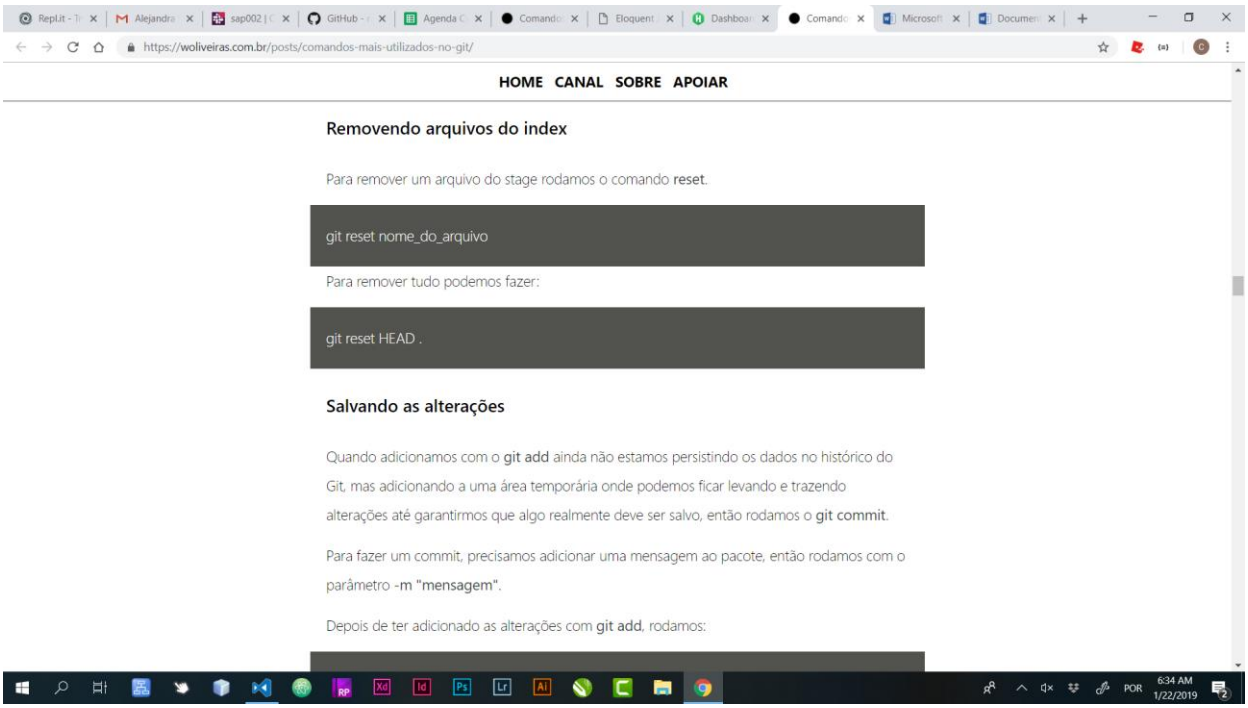
```
git add .
```

OBS: Cuidado com esse comando, pois você pode adicionar algo que não queria.

Também podemos rodar **git commit** com o parâmetro **-am**, onde adicionamos tudo de uma vez e já deixamos uma mensagem para o commit.

Exemplo:

```
git commit -am "add tudo"
```



https://woliveiras.com.br/posts/comandos-mais-utilizados-no-git/

HOME CANAL SOBRE APOIAR

Removendo arquivos do index

Para remover um arquivo do stage rodamos o comando **reset**.

```
git reset nome_do_arquivo
```

Para remover tudo podemos fazer:

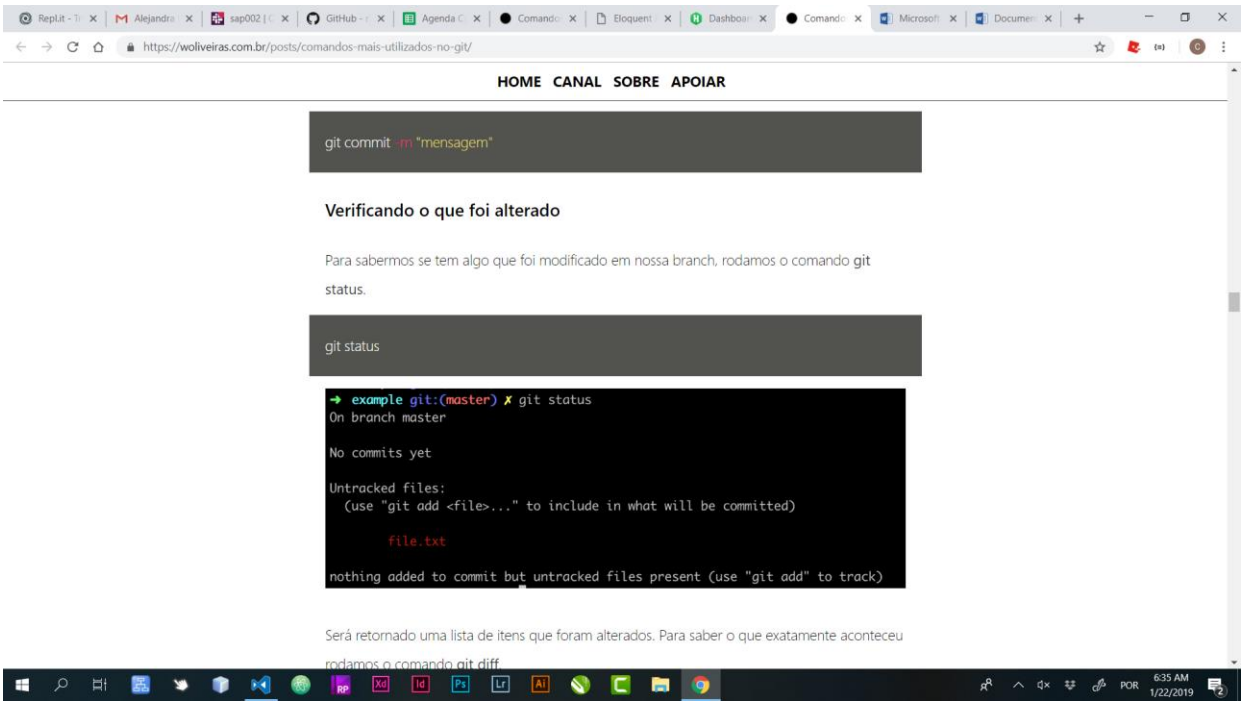
```
git reset HEAD .
```

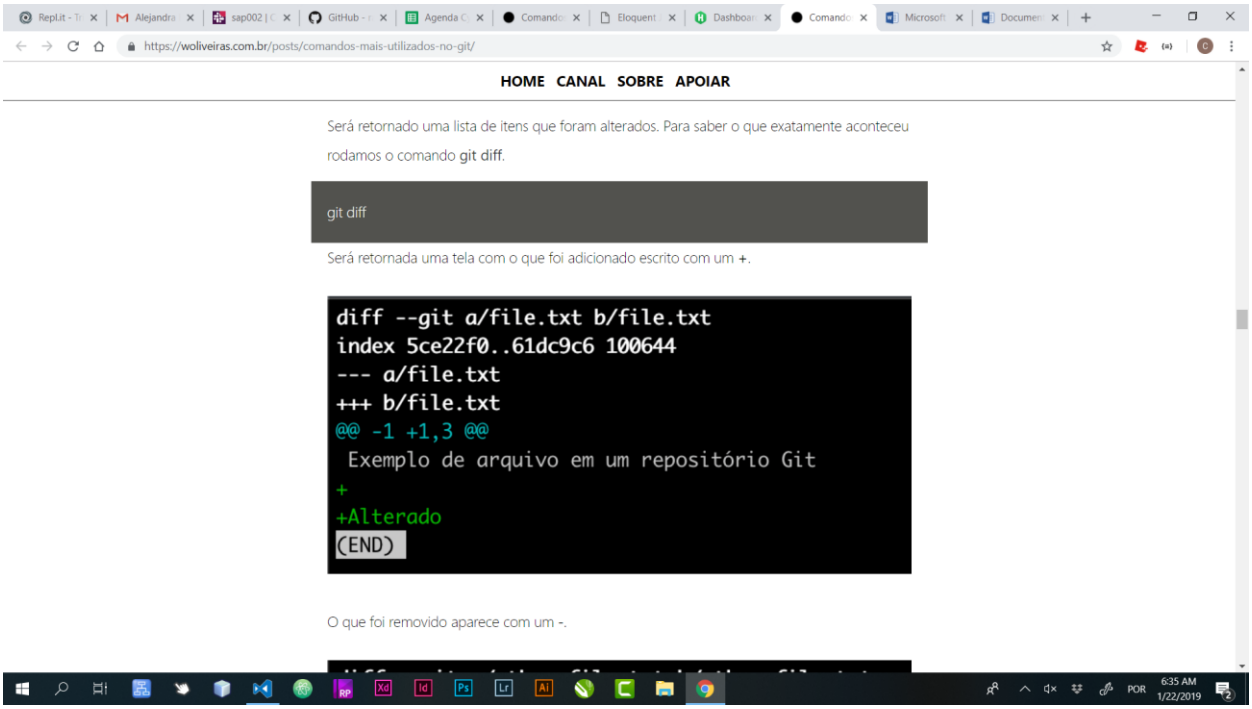
Salvando as alterações

Quando adicionamos com o **git add** ainda não estamos persistindo os dados no histórico do Git, mas adicionando a uma área temporária onde podemos ficar levando e trazendo alterações até garantirmos que algo realmente deve ser salvo, então rodamos o **git commit**.

Para fazer um commit, precisamos adicionar uma mensagem ao pacote, então rodamos com o parâmetro **-m "mensagem"**.

Depois de ter adicionado as alterações com **git add**, rodamos:


https://woliveiras.com.br/posts/comandos-mais-utilizados-no-git/
HOME CANAL SOBRE APOIAR
git commit -m "mensagem"
Verificando o que foi alterado
Para sabermos se tem algo que foi modificado em nossa branch, rodamos o comando git status.
git status
➜ example git:(master) ✗ git status
On branch master
No commits yet
Untracked files:
(use "git add <file>..." to include in what will be committed)
file.txt
nothing added to commit but untracked files present (use "git add" to track)
Será retornado uma lista de itens que foram alterados. Para saber o que exatamente aconteceu
rodamos o comando git diff.


https://woliveiras.com.br/posts/comandos-mais-utilizados-no-git/
HOME CANAL SOBRE APOIAR
Será retornado uma lista de itens que foram alterados. Para saber o que exatamente aconteceu rodamos o comando git diff.
git diff
Será retornada uma tela com o que foi adicionado escrito com um +.
diff --git a/file.txt b/file.txt
index 5ce22f0..61dc9c6 100644
--- a/file.txt
+++ b/file.txt
@@ -1 +1,3 @@
Exemplo de arquivo em um repositório Git
+
+Alterado
(END)
O que foi removido aparece com um -.

O que foi removido aparece com um -.

```
diff --git a/other_file.txt b/other_file.txt
index af643f7..f307a34 100644
--- a/other_file.txt
+++ b/other_file.txt
@@ -1,3 +1,2 @@
 Este é outro arquivo.

-Loko, né?
(END)
```

Caso tenhamos mais de um arquivo alterados por vez, podemos analisar todo o histórico com git diff ou observar somente um arquivo com git diff nome_do_arquivo.

Trabalhando com branches

Listando as branches existentes

https://woliveiras.com.br/posts/comandos-mais-utilizados-no-git/

HOME CANAL SOBRE APOIAR

Trabalhando com branches

Listando as branches existentes

```
git branch
```

Criando uma nova branch

Podemos rodar o comando git branch ou git checkout, conforme os exemplos:

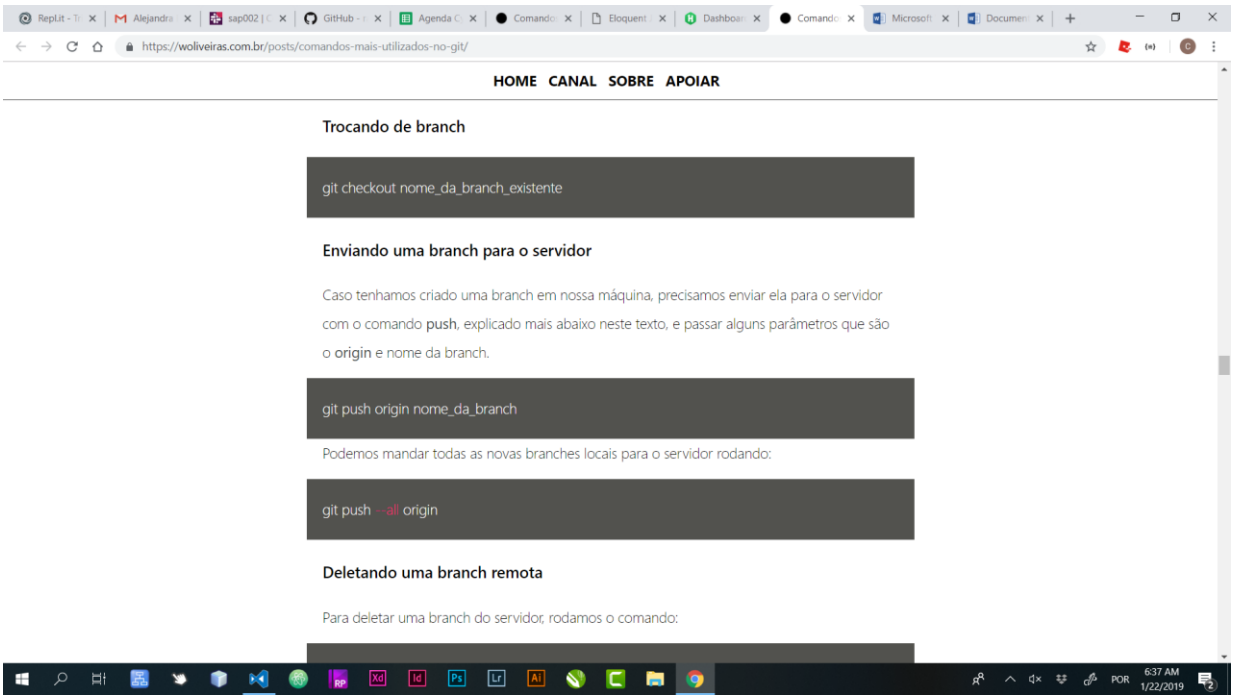
```
git branch nome
```

Criando uma nova branch e já trocando para ela

```
git checkout -b nome_da_nova_branch
```

Deletando uma branch

```
git branch -d nome
```



https://woliveiras.com.br/posts/comandos-mais-utilizados-no-git/

HOME CANAL SOBRE APOIAR

Trocando de branch

```
git checkout nome_da_branch_existente
```

Enviando uma branch para o servidor

Caso tenhamos criado uma branch em nossa máquina, precisamos enviar ela para o servidor com o comando **push**, explicado mais abaixo neste texto, e passar alguns parâmetros que são o **origin** e nome da branch.

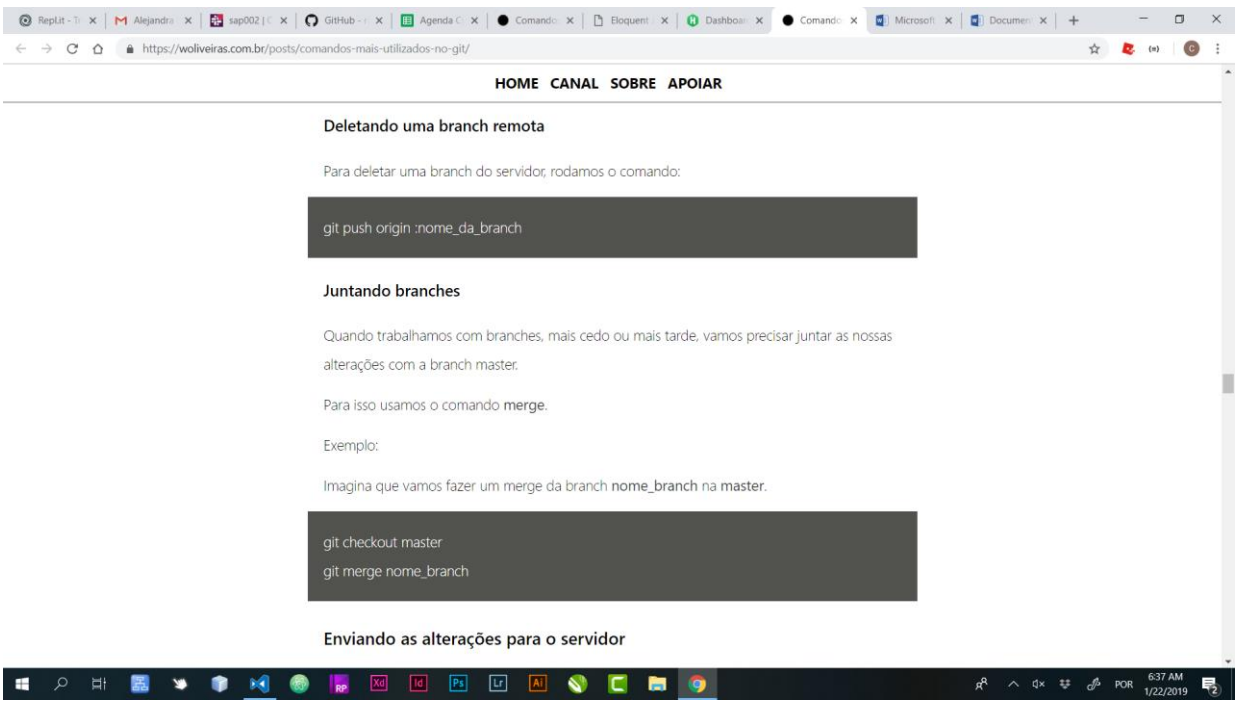
```
git push origin nome_da_branch
```

Podemos mandar todas as novas branches locais para o servidor rodando:

```
git push --all origin
```

Deletando uma branch remota

Para deletar uma branch do servidor, rodamos o comando:



https://woliveiras.com.br/posts/comandos-mais-utilizados-no-git/

HOME CANAL SOBRE APOIAR

Deletando uma branch remota

Para deletar uma branch do servidor, rodamos o comando:

```
git push origin :nome_da_branch
```

Juntando branches

Quando trabalhamos com branches, mais cedo ou mais tarde, vamos precisar juntar as nossas alterações com a branch master.

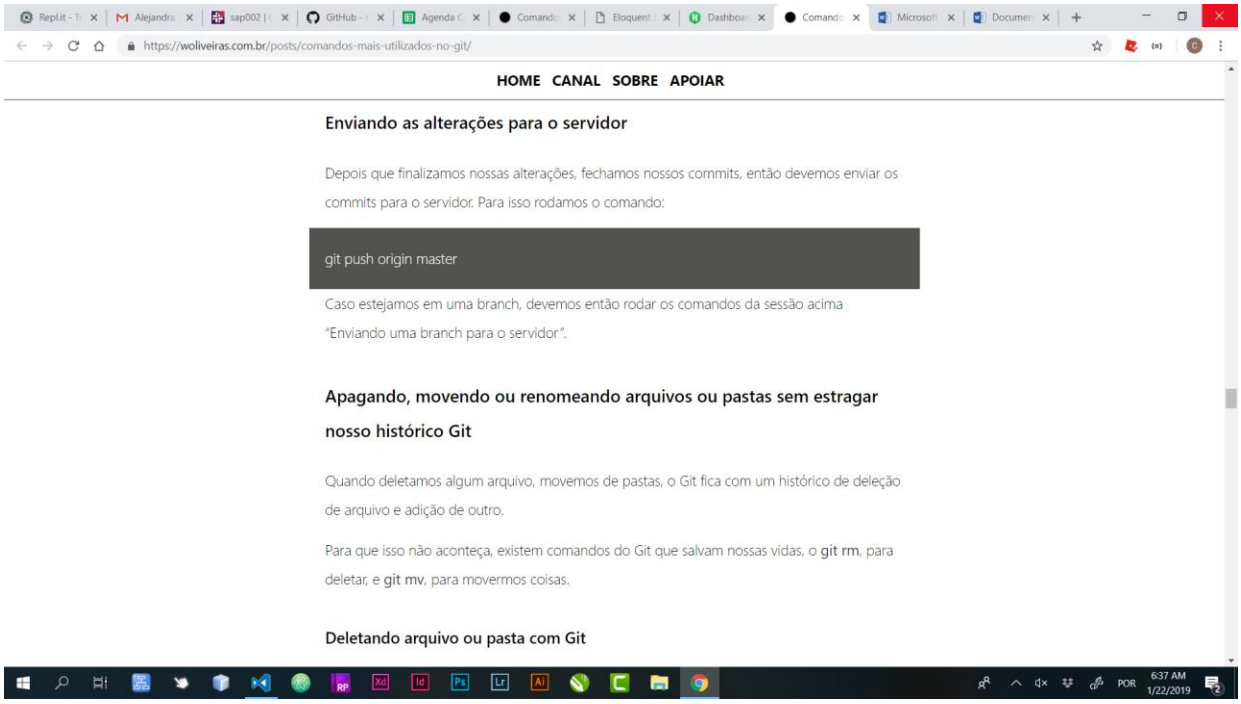
Para isso usamos o comando **merge**.

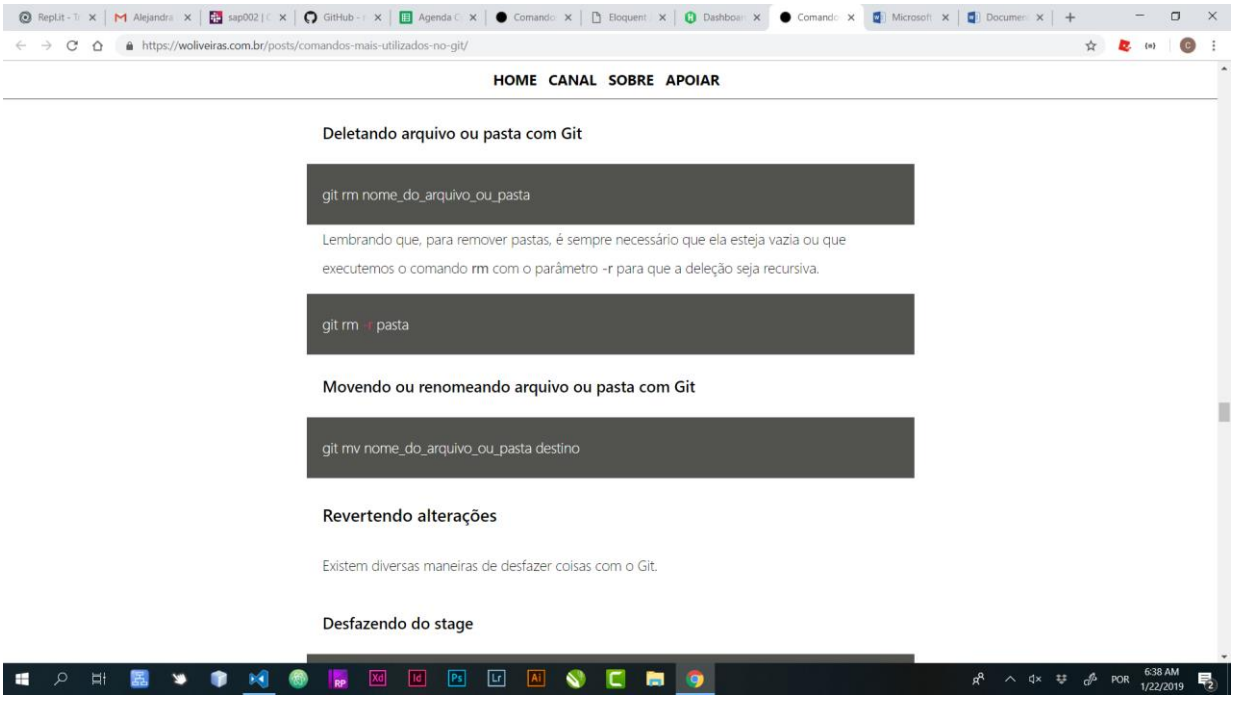
Exemplo:

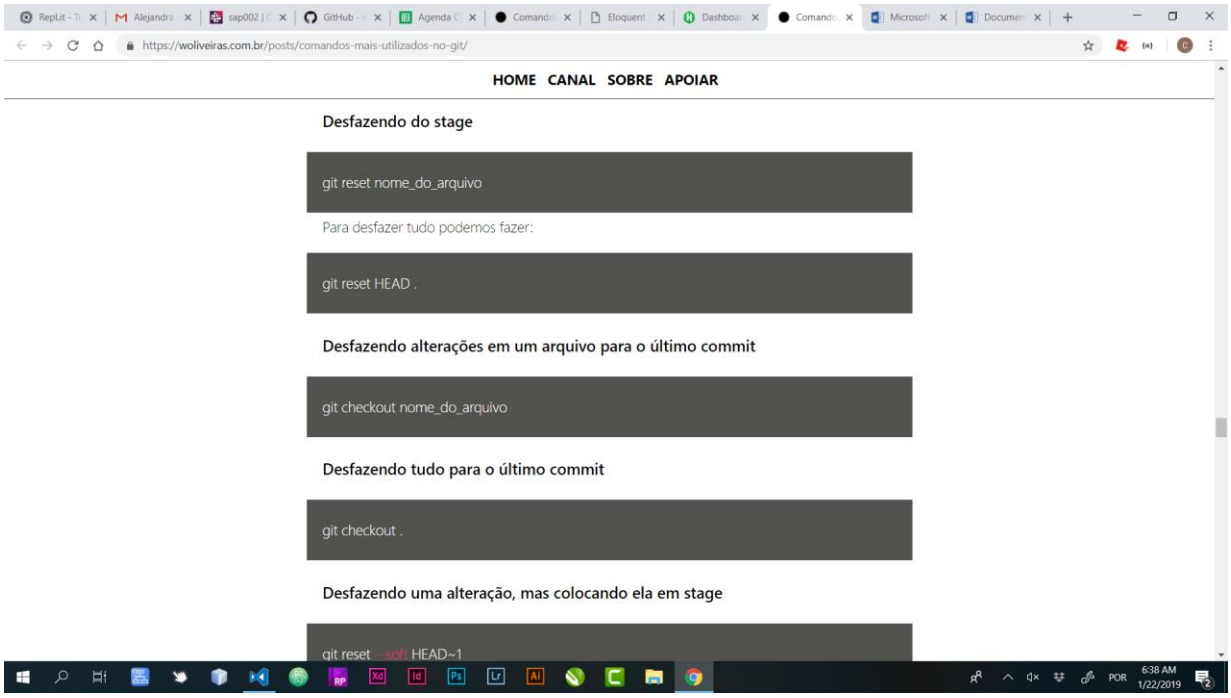
Imagina que vamos fazer um merge da branch **nome_branch** na **master**.

```
git checkout master
git merge nome_branch
```

Enviando as alterações para o servidor


https://woliveiras.com.br/posts/comandos-mais-utilizados-no-git/
HOME CANAL SOBRE APOIAR
Enviando as alterações para o servidor
Depois que finalizamos nossas alterações, fechamos nossos commits, então devemos enviar os commits para o servidor. Para isso rodamos o comando:
git push origin master
Caso estejamos em uma branch, devemos então rodar os comandos da sessão acima "Enviando uma branch para o servidor".
Apagando, movendo ou renomeando arquivos ou pastas sem estragar nosso histórico Git
Quando deletamos algum arquivo, movemos de pastas, o Git fica com um histórico de deleção de arquivo e adição de outro.
Para que isso não aconteça, existem comandos do Git que salvam nossas vidas, o git rm, para deletar, e git mv, para movermos coisas.
Deletando arquivo ou pasta com Git
POR 6:37 AM 1/22/2019


https://woliveiras.com.br/posts/comandos-mais-utilizados-no-git/
HOME CANAL SOBRE APOIAR
Deletando arquivo ou pasta com Git
git rm nome_do_arquivo_ou_pasta
Lembrando que, para remover pastas, é sempre necessário que ela esteja vazia ou que executemos o comando rm com o parâmetro -r para que a deleção seja recursiva.
git rm -r pasta
Movendo ou renomeando arquivo ou pasta com Git
git mv nome_do_arquivo_ou_pasta destino
Revertendo alterações
Existem diversas maneiras de desfazer coisas com o Git.
Desfazendo do stage
POR 6:38 AM 1/22/2019



https://woliveiras.com.br/posts/comandos-mais-utilizados-no-git/

HOME CANAL SOBRE APOIAR

Desfazendo do stage

```
git reset nome_do_arquivo
```

Para desfazer tudo podemos fazer:

```
git reset HEAD .
```

Desfazendo alterações em um arquivo para o último commit

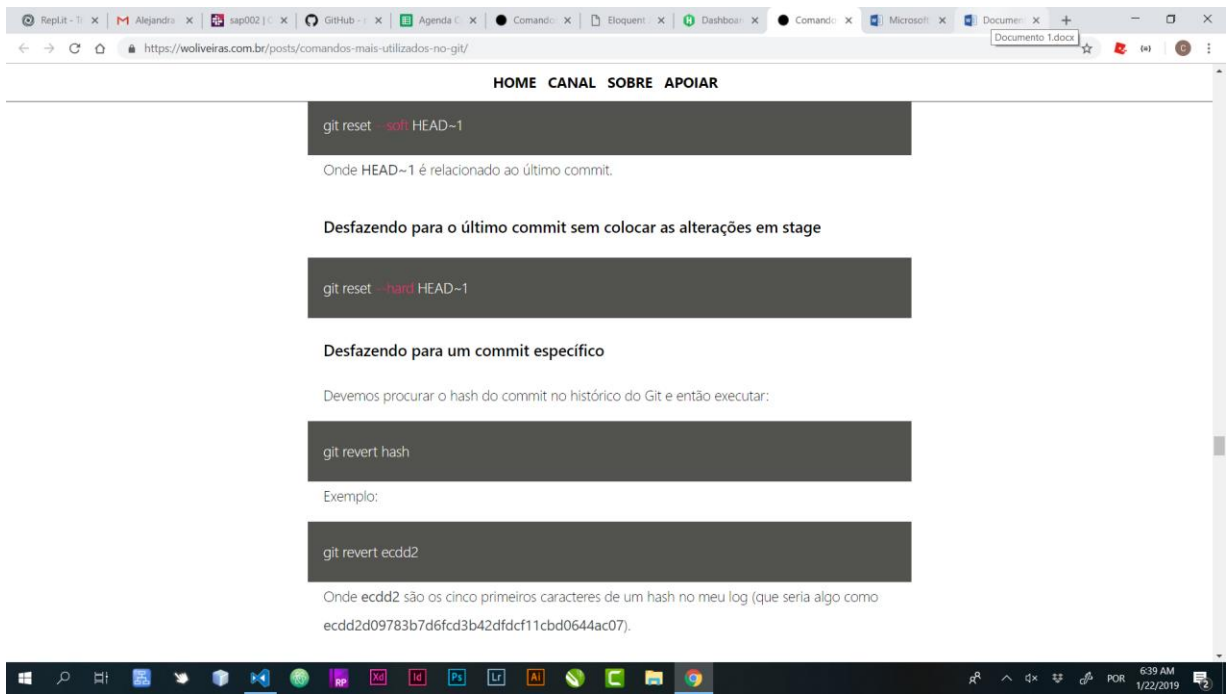
```
git checkout nome_do_arquivo
```

Desfazendo tudo para o último commit

```
git checkout .
```

Desfazendo uma alteração, mas colocando ela em stage

```
git reset --soft HEAD~1
```



https://woliveiras.com.br/posts/comandos-mais-utilizados-no-git/

HOME CANAL SOBRE APOIAR

```
git reset --soft HEAD~1
```

Onde **HEAD~1** é relacionado ao último commit.

Desfazendo para o último commit sem colocar as alterações em stage

```
git reset --hard HEAD~1
```

Desfazendo para um commit específico

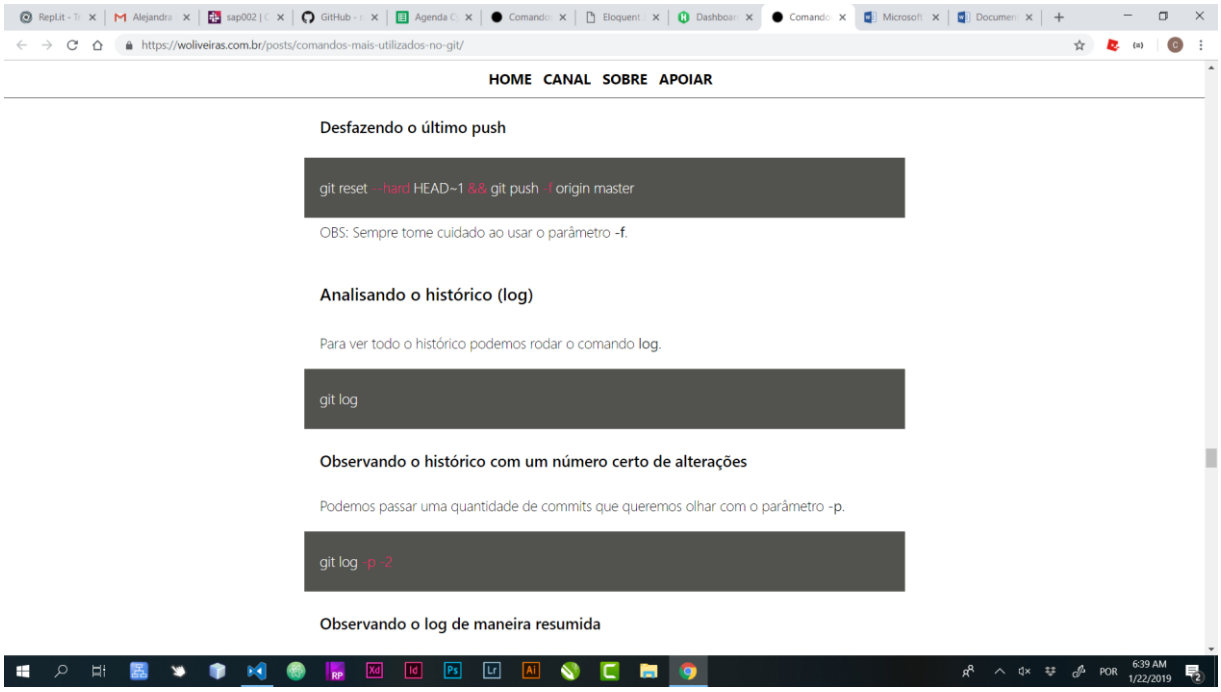
Devemos procurar o hash do commit no histórico do Git e então executar:

```
git revert hash
```

Exemplo:

```
git revert ecdd2
```

Onde **ecdd2** são os cinco primeiros caracteres de um hash no meu log (que seria algo como **ecdd2d09783b7d6fcd3b42dfdcf11cbd0644ac07**).



https://woliveiras.com.br/posts/comandos-mais-utilizados-no-git/

HOME CANAL SOBRE APOIAR

Desfazendo o último push

```
git reset --hard HEAD~1 && git push -f origin master
```

OBS: Sempre tome cuidado ao usar o parâmetro **-f**.

Analisando o histórico (log)

Para ver todo o histórico podemos rodar o comando **log**.

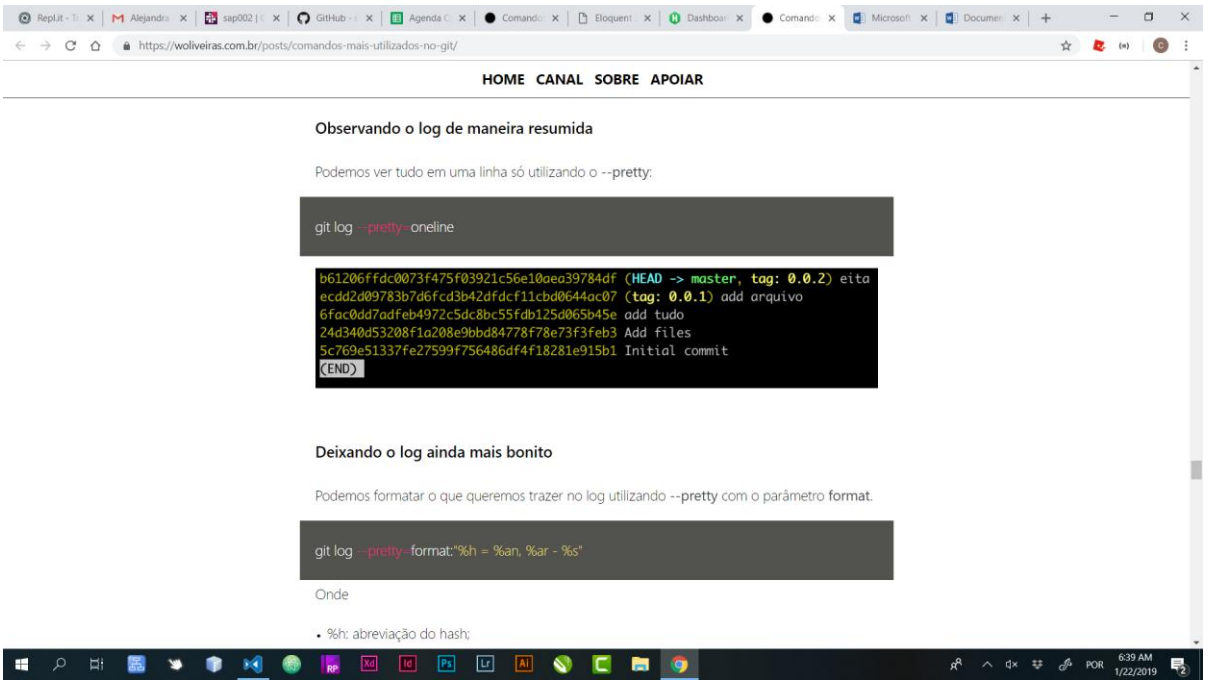
```
git log
```

Observando o histórico com um número certo de alterações

Podemos passar uma quantidade de commits que queremos olhar com o parâmetro **-p**.

```
git log -p -2
```

Observando o log de maneira resumida



https://woliveiras.com.br/posts/comandos-mais-utilizados-no-git/

HOME CANAL SOBRE APOIAR

Observando o log de maneira resumida

Podemos ver tudo em uma linha só utilizando o **--pretty**:

```
git log --pretty=oneline
```

```
b61206ffdc0073f475f03921c56e10aea39784df (HEAD -> master, tag: 0.0.2) eita
ecdd2d09783b7d6fcd3b42dfdcf11cbd0644ac07 (tag: 0.0.1) add arquivo
6fac0dd7adfeb4972c5dc8bc55fdb125d065b45e add tudo
24d340d53208f1a208e9bbd84778f78e73f3feb3 Add files
5c769e51337fe27599f756486df4f18281e915b1 Initial commit
(END)
```

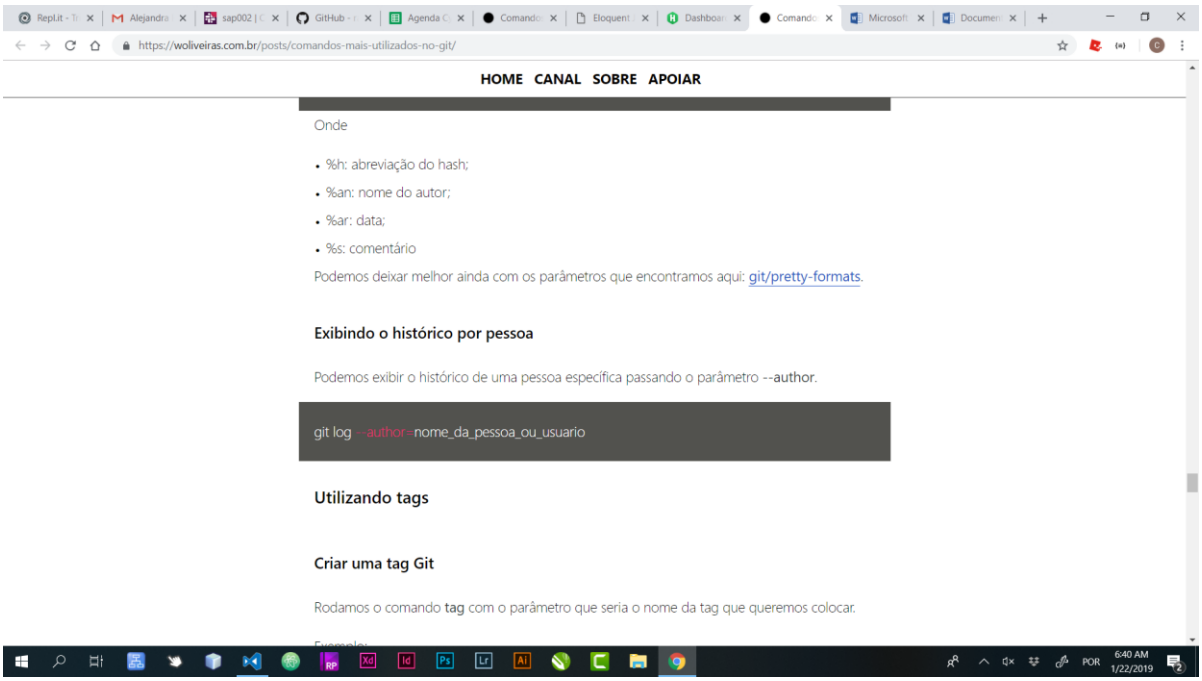
Deixando o log ainda mais bonito

Podemos formatar o que queremos trazer no log utilizando **--pretty** com o parâmetro **format**.

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

Onde

- %h: abreviação do hash;



HOME CANAL SOBRE APOIAR

Onde

- %h: abreviação do hash;
- %an: nome do autor;
- %ar: data;
- %s: comentário

Podemos deixar melhor ainda com os parâmetros que encontramos aqui: git/pretty-formats.

Exibindo o histórico por pessoa

Podemos exibir o histórico de uma pessoa específica passando o parâmetro --author.

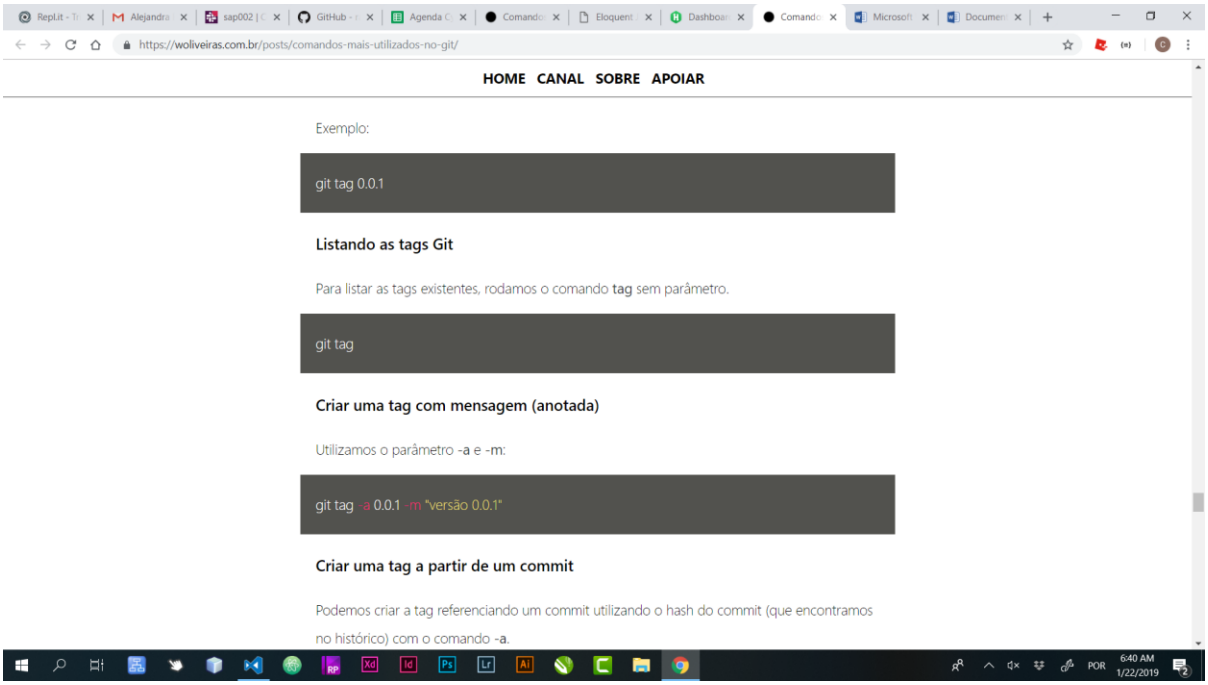
```
git log --author=nome_da_pessoa_ou_usuario
```

Utilizando tags

Criar uma tag Git

Rodamos o comando **tag** com o parâmetro que seria o nome da tag que queremos colocar.

Exemplo:



HOME CANAL SOBRE APOIAR

Exemplo:

```
git tag 0.0.1
```

Listando as tags Git

Para listar as tags existentes, rodamos o comando **tag** sem parâmetro.

```
git tag
```

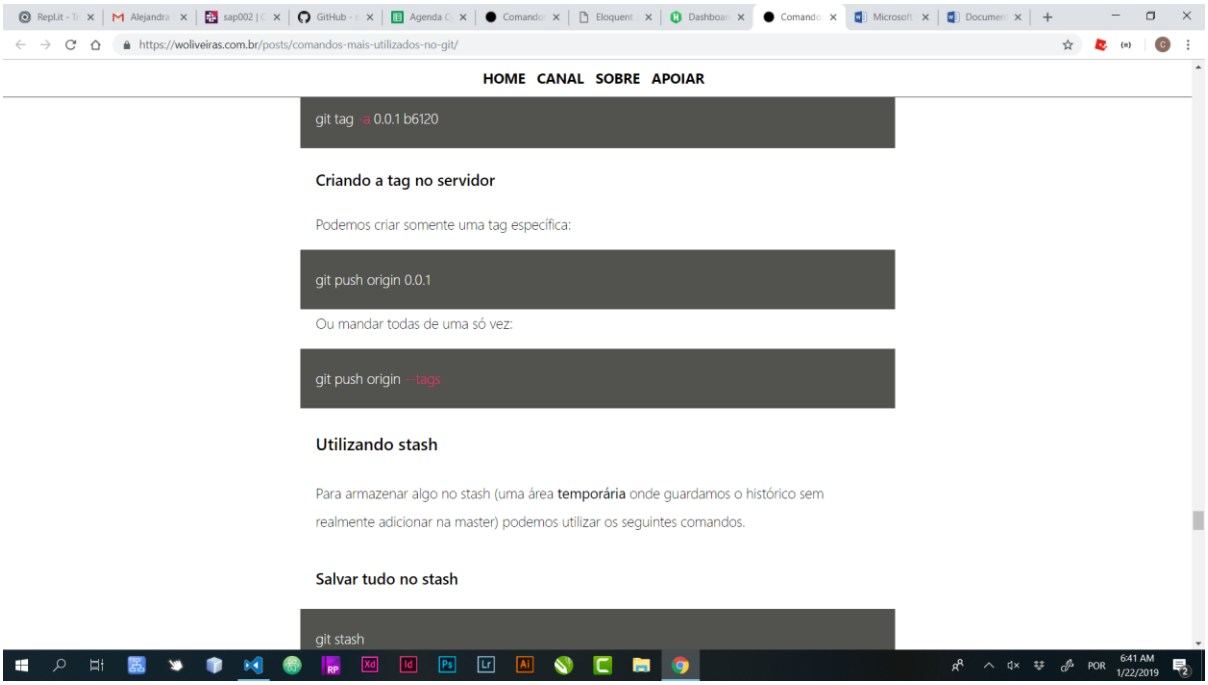
Criar uma tag com mensagem (anotada)

Utilizamos o parâmetro -a e -m:

```
git tag -a 0.0.1 -m "versão 0.0.1"
```

Criar uma tag a partir de um commit

Podemos criar a tag referenciando um commit utilizando o hash do commit (que encontramos no histórico) com o comando -a.



https://woliveiras.com.br/posts/comandos-mais-utilizados-no-git/

HOME CANAL SOBRE APOIAR

```
git tag -a 0.0.1 b6120
```

Criando a tag no servidor

Podemos criar somente uma tag específica:

```
git push origin 0.0.1
```

Ou mandar todas de uma só vez:

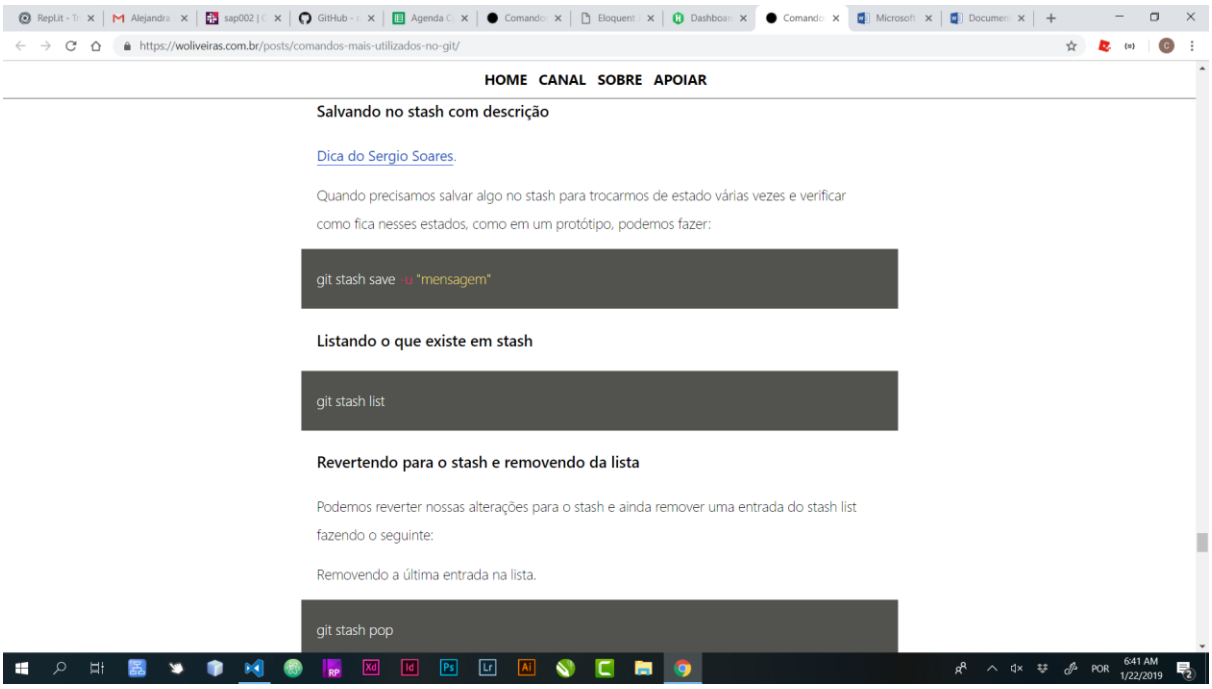
```
git push origin --tags
```

Utilizando stash

Para armazenar algo no stash (uma área **temporária** onde guardamos o histórico sem realmente adicionar na master) podemos utilizar os seguintes comandos.

Salvar tudo no stash

```
git stash
```



https://woliveiras.com.br/posts/comandos-mais-utilizados-no-git/

HOME CANAL SOBRE APOIAR

Salvando no stash com descrição

Dica do Sergio Soares.

Quando precisamos salvar algo no stash para trocarmos de estado várias vezes e verificar como fica nesses estados, como em um protótipo, podemos fazer:

```
git stash save -u "mensagem"
```

Listando o que existe em stash

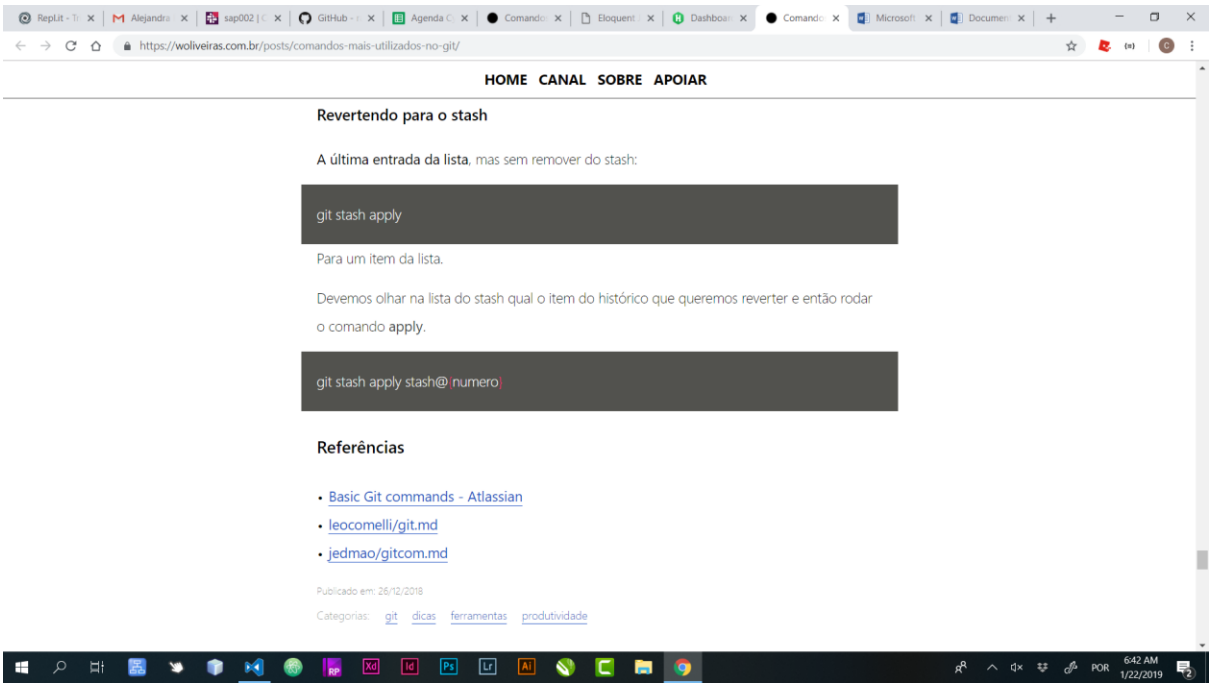
```
git stash list
```

Revertendo para o stash e removendo da lista

Podemos reverter nossas alterações para o stash e ainda remover uma entrada do stash list fazendo o seguinte:

Removendo a última entrada na lista.

```
git stash pop
```


https://woliveiras.com.br/posts/comandos-mais-utilizados-no-git/
HOME CANAL SOBRE APOIAR
Revertendo para o stash
A última entrada da lista, mas sem remover do stash:
git stash apply
Para um item da lista.
Devemos olhar na lista do stash qual o item do histórico que queremos reverter e então rodar o comando apply.
git stash apply stash@{numero}
Referências
Basic Git commands - Atlassian
leocomelli/git.md
jedmao/gitcom.md
Publicado em: 26/12/2018
Categorias: git dicas ferramentas produtividade
POR
6:42 AM
1/22/2019