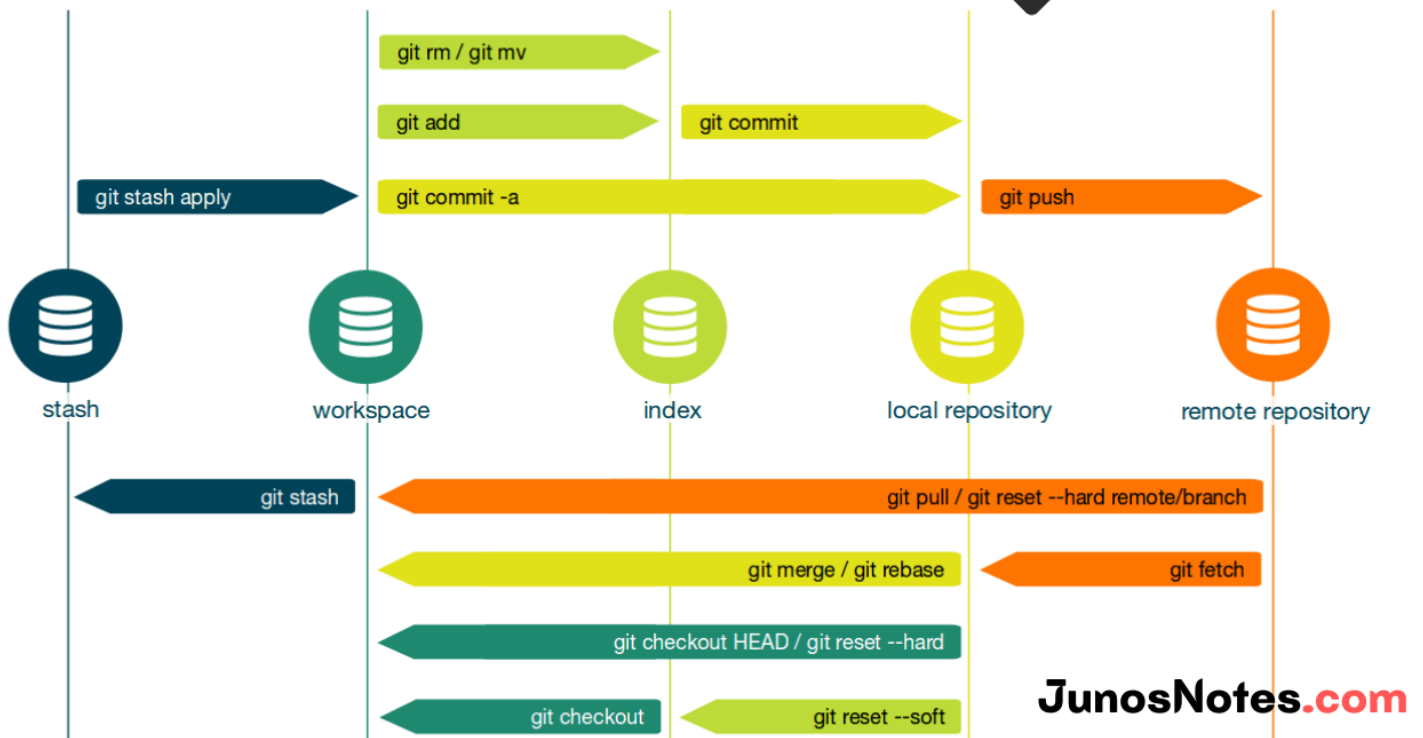


Basic Git Commands



git init

Esse comando é responsável por inicializar um novo repositório. Em um processo de versionamento de código, esse é o primeiro comando básico.

git add

Este é responsável por adicionar supostas mudanças no diretório do projeto à área de staging, dando uma oportunidade de preparar o código antes de fazer o commit, existem duas formas de trabalhar-lo com o `.` no final para enviar tudo, ou especificando a pasta ou arquivo a ser adicionado `'git add hello_world.php'`

git commit

Para submeter as mudanças incluídas no `git add`, usa-se o comando `git commit`, habitualmente é usado com a flag `-m "mensagem"`; que significa *mensagem*, ou seja iremos adicionar um título a este commit, relacionado a alteração atrelada a ele. `'git commit -m "Adicionei o arquivo hello_world.php"'`

git status

Após realizar um commit é de prática comum realizar um `git status`, para verificar se existem mudanças no repositório remoto as quais seu código local ainda não tem, existem duas possíveis respostas deste comando

- 1- `'On branch master No commits yet'` - o que quer dizer que você está em dia com o repositório remoto, e não há mudanças a serem feitas.
- 2- `'On branch master Your branch is ahead of "nome da branch"'` - o que quer dizer que algum dos arquivos foi atualizado por outro usuário e o seu código local está desatualizado

git push

Este comando é responsável por enviar seu commit mais recente até o repositório remoto, existem algumas formas de trabalhar com ele, e erros, os mais comuns são:

Git push -u origin 'nome da branch' - a primeira forma é enviar o código para uma branch específica

*Git push --set-upstream-branch origin 'nome da branch' - aqui estamos associando esta branch uma flag -upstream que significa que esta branch está atualizada e será a padrão, após realizar este comando, na próxima vez que for realizar o envio do código para o repositório remoto pode usar apenas o **git push** sem nenhuma flag ou especificação. * **cuidado ao usar este comando trabalhando em branches diferentes.***

O Erro mais comum é relacionado ao comando anterior:

```
! [rejected]      master -> master (fetch first)
error: failed to push some refs to 'https://github.com/CADOX-Technology/ARKEEVA.git'
hint: Updates were rejected because the remote contains work that you do not
hint: have locally. This is usually caused by another repository pushing to
hint: the same ref. If you want to integrate the remote changes, use
hint: 'git pull' before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

Isso acontece porque estamos tentando enviar um código para um repositório que está à frente do local, para resolver este problema existem dois comandos, e o mais usado é o **git pull**

Git pull

Git pull é a versão automatizada do **git fetch**. Ele baixa uma ramificação de um repositório remoto e a mescla imediatamente na ramificação atual, este comando também pode retornar erros, que veremos à frente com o **git merge**

Após a realização bem sucedida deste comando, deve possibilitar a realização do **git push** para que ambos os códigos local e remoto fiquem iguais.

-Versionamento, atualização e erros-

Saindo dos comandos básicos

Git branch

Suas principais funções são a de criar uma branch, listar todas as branches e deletar uma branch. Uma branch pode nos auxiliar imensamente no processo de manutenção do código. Por exemplo, atualmente nosso arquivo "hello_world.php" está printando uma informação com erro de digitação. Para fazer a alteração correndo o menor número de risco, podemos criar uma nova branch apenas para isso. Assim, criamos a branch usando o seguinte código: **git branch 'nome da branch'** (que em vias de documentação é recomendado que seja relacionado ao erro a ser consertado ou nova função a ser implementada)

git branch sem nenhuma especificação irá nos listar todas as branches do projeto,
git branch -d 'nome da branch' é utilizado para deletar um branch

git checkout

Esse comando do Git tem como função mudar de branch ou voltar para algum estado do seu projeto. Sendo de extrema importância para qualquer repositório que utilize branches, basta utilizar o comando **git checkout 'nome da branch que eu quero utilizar'**

Outra funcionalidade interessante do **git checkout** é sua utilização para voltar em algum commit anterior utilizando o comando **base mais o código de soma de verificação do commit:**
git checkout 166 ae0c4d3f420721acbb115cc33848dfcc2121a

Para localizar o código do commit iremos utilizar um novo comando

Git tag

Essa função do Git serve para criar etiquetas de estados que sejam relevantes, como por exemplo, versão final ou alguma versão já utilizável do projeto. Assim, o Git salva versões para organizar melhor o projeto e também para poder voltar se em uma versão posterior ocorrer algum problema. Existem três formas de criar uma tag utilizando o Git, sendo elas: com anotações, sem anotações e após o commit já feito.

Para criar uma tag com mensagem se utiliza o comando abaixo colocando o nome da tag após o -a e uma mensagem qualquer depois do -m:

```
git tag -a v1.0 -m 'Minha versão 1.0'
```

Já para criar uma tag sem anotações usa-se apenas o comando base com nome da versão:

```
git tag v1.1
```

E para marcar um commit depois de já tê-lo feito se usa o comando git tag com o nome da versão e a soma de verificação do commit.

```
git tag v1.2 166ae0c4d3f420721acbb115cc33848dfcc2121a
```

Agora como achar a verificação do commit? Cada commit feito tem uma soma de número e letras para identificá-lo

git log

Permite que você explore as revisões passadas de um projeto. Ele oferece muitas opções de formatação para indicar os snapshots commitados, ao rodar o comando no cmd temos um resultado parecido com este:

```
commit 2d952be9182e53bd1f7f0fd4e652e44f0fb3333c (HEAD -> master)
```

```
Author: lMainente <caetanomainente@outlook.com>
```

```
Date: Thu Feb 1 16:51:51 2024 -0300
```

```
manual git
```

```
commit 7c46c3b13e0879f523d8a47089f7297ea576e923 (origin/master, origin/HEAD)
```

```
Author: lMainente <caetanomainente@outlook.com>
```

```
Date: Thu Feb 1 13:03:43 2024 -0300
```

```
conexao com o banco
```

```
commit 5bfcbf8a116654434d9d1996a27b441fb87eda3e
```

```
Author: lMainente <caetanomainente@outlook.com>
```

Onde podemos identificar o código do commit para ações futuras, junto com seu autor e comentário.

Git merge

Combina as alterações de diferentes branches no repositório. O comando git merge é usado para incorporar as mudanças de uma branch específica em outra branch ativa.

Exemplo:

Suponha que você esteja atualmente na branch "main" e queira mesclar as alterações da branch "feature-nova" nela.

Certifique-se de estar na branch de destino:

```
git checkout main
```

Execute o comando git merge para mesclar as alterações da branch "feature-nova":

```
git merge feature-nova
```

Isso irá incorporar as alterações da branch "feature-nova" na branch "main". Se houver conflitos, você precisará resolvê-los manualmente.

Este é um comando que pode ser realizado automaticamente com o git pull, quando nós temos um arquivo com códigos conflitantes nas versões, usando a IDE vscode é possível realizar este processo via source control, escolhendo entre quatro opções

```
You, 1 second ago | 1 author (You) | Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
<<<<<< HEAD (Current Change)
hehe
=====
hH
>>>>>> 52e107aa74639f482e08ea7779af4eccf10154ed (Incoming Change)
```

Accept Current Change, Accept Incoming Change, Accept both changes e Compare Changes

Onde:

Accept Current Change mantém o código feito por você, que esta no seu repositório local

Accept Incoming Change substitui o seu bloco de código por um no qual esta vindo do repositório remoto

Accept both changes mantém os dois códigos

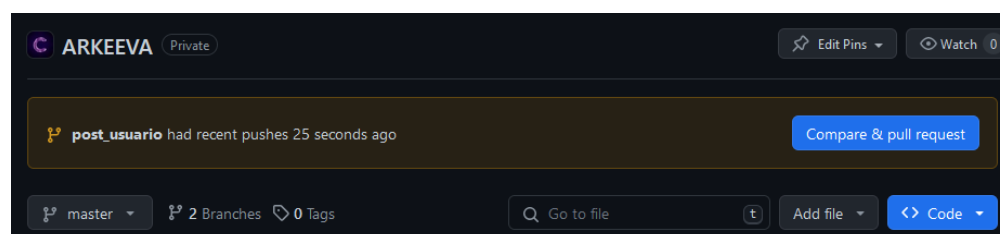
Compare Changes abre um editor para que você possa visualizar ambos os códigos conflitantes de forma mais clara e dependendo do tamanho do bloco, resolve-los a mão

Após Realizar este processo é necessário novamente commitar as mudanças realizadas e agora sim enviar via git push para o repositório remoto, porém também existe outra forma de trabalhar este envio, dependendo da alteração ou tamanho da sua equipe, o processo de git push ou merge direto na branch main pode afetar a ordem de trabalho, por isso existe:

Git pull request

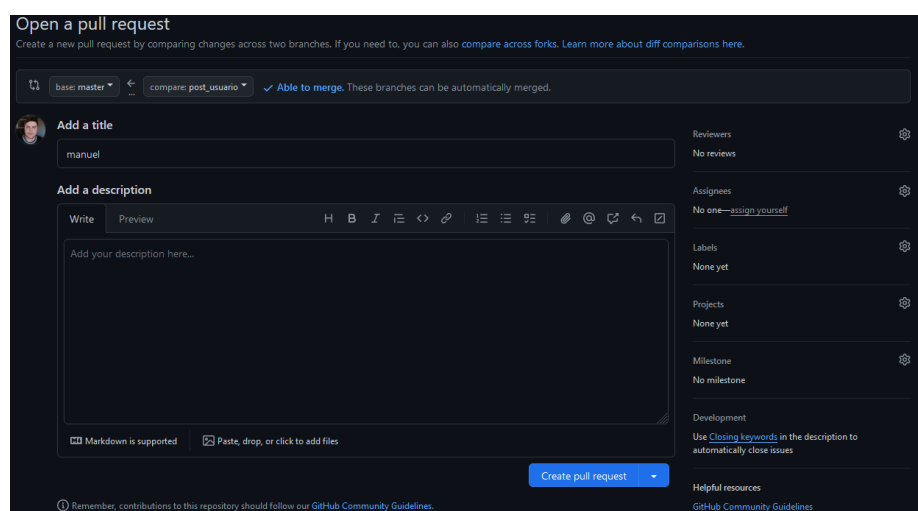
Este comando que não é bem um comando é realizado pelo github, ou seja pela interface gráfica do git.

Após realizar o push do código na sua nova branch, você irá notar no repositório uma notificação



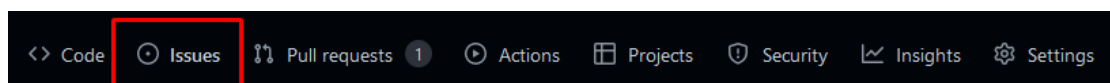
Essa notificação se refere a que você pode criar um pull request do código da sua branch, para que um responsável realize o comando git merge na branch principal.

Ao clicar você pode colocar um título assim como descrição e atribuir tags, e responsáveis pela revisão

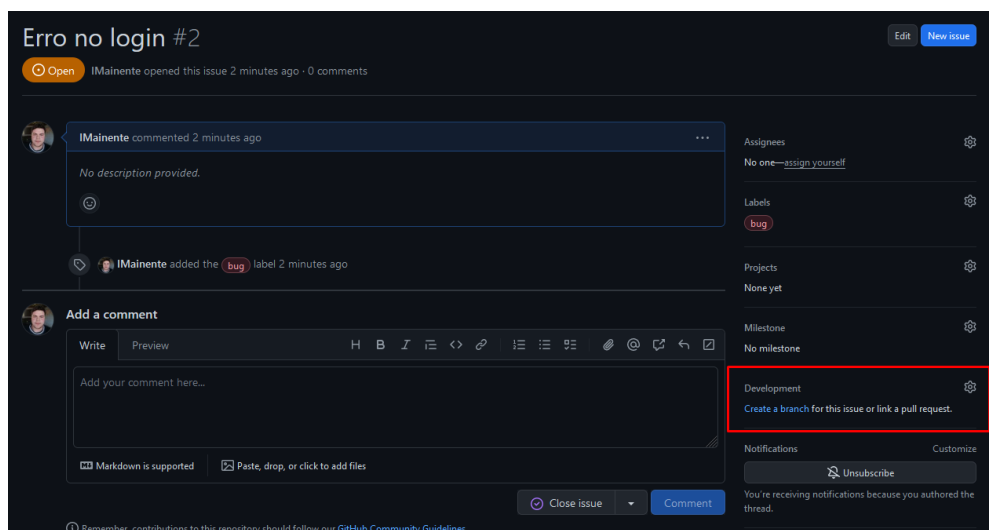


Após solicitar você verá seu pull request na fila para merge, podendo ser realizado pelo github, ou comando merge no cmd.

Agora iremos para a parte de issues do github



Uma forma de manter os bugs encontrados no projeto de forma organizada e sempre que identificado um criar um issue no github, desta forma é possível visualizar erros que já aconteceram e qual o código relacionado a resolução dele, uma vez a qual você pode criar um branch pelo github para este erro, assim caso aconteça algo parecido de novo existe um log.



Agora e se o que causou este erro foi um commit errado? Existem também comandos para voltar a versões antigas do seu código.

1. Desfazer o último commit (ainda não foi enviado):

`git reset HEAD^`

Este comando desfaz o último commit, mantendo as alterações no seu working directory. Você pode então fazer as correções necessárias e commitar novamente.

2. Desfazer o último commit (inclusivo dos arquivos):

git reset --hard HEAD^

Este comando desfaz o último commit e descarta as alterações feitas, revertendo completamente para o estado do commit anterior.

3. Alterar a mensagem do último commit:

git commit --amend -m "Nova mensagem do commit"

Este comando permite que você altere a mensagem do último commit. Ele também pode ser usado para adicionar mais alterações ao último commit se você esqueceu de incluir alguma coisa.

4. Reverter um commit existente:

git revert código-do-Commit (revisitar git log)

Isso cria um novo commit que desfaz as alterações introduzidas por um commit anterior, mantendo o histórico de commits intacto.

5. Rebase interativo para alterar commits:

git rebase -i HEAD~n

O rebase interativo permite reescrever o histórico de commits de uma branch. n é o número de commits que você deseja revisar. Uma vez no modo interativo, você pode escolher "edit" para cada commit que deseja alterar, fazer as alterações necessárias e, em seguida, continuar com o rebase.