Relatório 26 - Git e Github para Iniciantes (III)

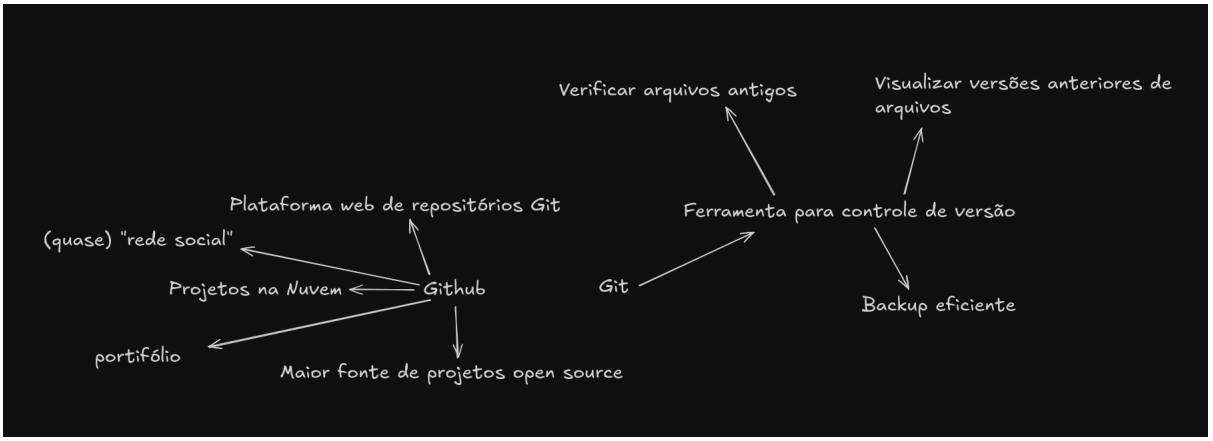
Felipe Fonseca

Descrição da atividade

Git e Github:

Git é uma ferramenta inventada por Linus Torvalds para manter um controle de versão do Kernel Linux, ele age com velocidade, possui design simples, um suporte muito bom e é capaz de lidar com projetos muito complexos. A necessidade desse controlador de versão passava por várias partes, como manter controle de tudo que já passou pelo projeto, podendo visualizar como o código estava versões atrás, podendo recuperar arquivos perdidos, e tendo basicamente backups de todas as versões de todos os arquivos que já passou, isso de uma forma muito otimizada, ou seja, sem a necessidade de ficar criando e copiando arquivos várias e várias vezes.

Já o github é basicamente uma plataforma ou um serviço de web que permite que repositórios que utilizam o git para fazer o controle de versões sejam armazenados na nuvem. O github é muito utilizado para guardar os arquivos do projeto, e ele permite o trabalho em conjunto dos programadores.



Configurando o Git

Nessa parte é ensinado algumas configurações básicas do git através de comandos. Como a configuração do nome que aparecerá quando você fizer um commit, que é feito através da variável user.name, ou então o email que está linkado na sua conta do github para quando fizer os commits, e para isso é usado a variável user.email. A variável core.editor também é citada, ela define o editor de texto que será aberto para algumas tarefas de texto, como escrever a mensagem de um commit.

Essencial do Git

Para um projeto, primeiramente criamos uma pasta e inicializamos ela como um repositório. Para isso eu usei apenas pelo terminal, os comandos mkdir para criar a pasta, o comando cd para entrar na pasta, e então o comando git init para iniciar o repositório.

Quando o git é inicializado, é criado um diretório chamado git. esse diretório é invisível, por isso possui um ponto antes dele, ou seja, para acessar, é necessário digitar "cd .git/".

Dentro do diretório tem vários arquivos relacionados ao repositório, como um arquivo de configuração, um arquivo com a descrição etc.

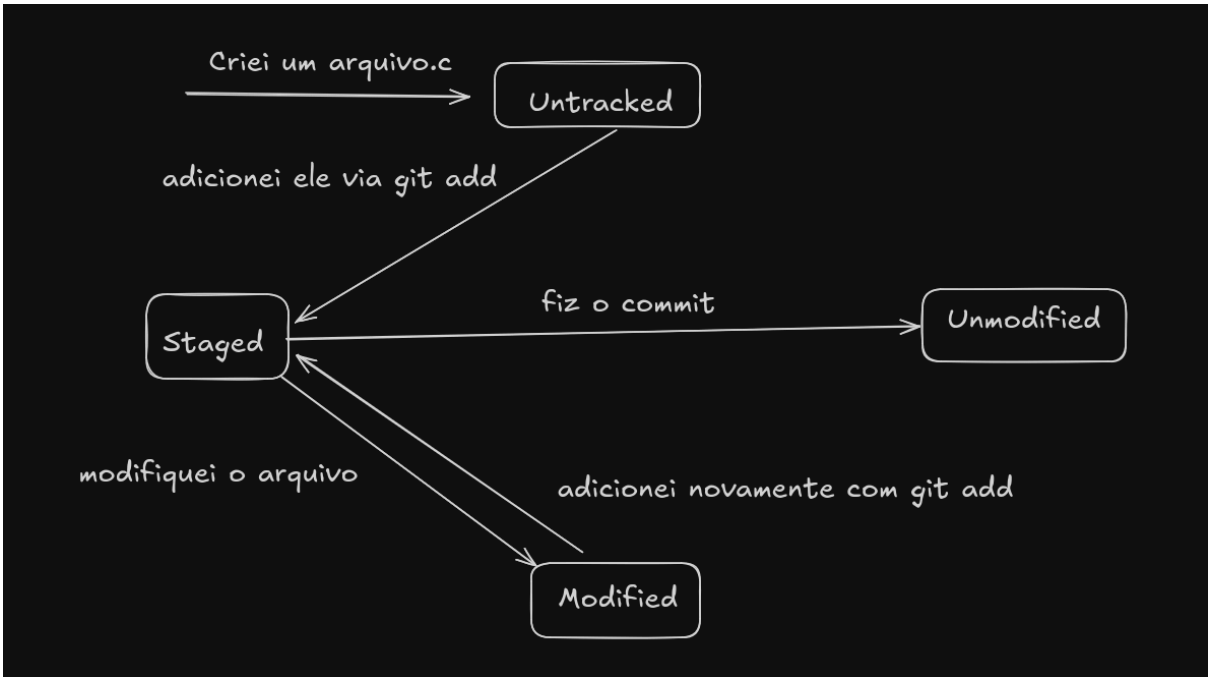
Com o comando vi <arquivo> você acessa o arquivo (se existir) ou cria o arquivo, e então abre o editor no terminal, porém eu não gosto muito do vi, então eu utilizo um outro editor chamado

nano, eu acho ele bem mais fácil de utilizar e as instruções dele são bem mais claras, pois as opções ficam na parte de baixo da tela.

O arquivo chamado README.md é um arquivo comum de se encontrar em repositórios no github, que servem basicamente para colocar as informações do repositório ou algo do tipo.

Os arquivos tem basicamente um ciclo de vida:

- Untracked: arquivo que pode até estar na pasta, mas não é conhecido pelo git, está sendo ignorado ou não foi visto.
- Unmodified: após a adição do arquivo, ou após o último commit, o arquivo não foi modificado.
- Modified: após adição ou commit, ele já foi modificado.
- Staged: preparado para o commit.



Utilizando o comando git status, é mostrado algumas informações do repositório, como a branch em que eu estou, qual commit eu estou, e os arquivos que estão para ser comitados.

Utilizando o comando git add, é possível adicionar coisas no git para ser comitado. Como por exemplo: git add <nome_arquivo>. Se eu modificar esse arquivo, eu terei que utilizar o git add novamente para adicionar o que foi modificado no arquivo para o staged, e assim ele ficar pronto para ser comitado, caso contrário, se eu apenas utilizar o commit, o que eu modifiquei depois não será levado para o repositório.

Ao utilizar o comando git commit, você basicamente cria o snapshot da versão atual, com os arquivos que foram adicionados. O comando completo é: git commit -m "<mensagem>".

O comando git log é um comando para ver algumas informações do commit, como a hash, que é basicamente a identificação do commit, a branch do commit, quem fez o commit, ou seja, o autor, e também a data exata do commit.

Instruções do log:

- git log –decorate
- git log –author="<nome_do_autor>"
- git shortlog
- git log – graph

O comando git diff serve para ver o que foi mudado no arquivo, muito útil para verificar se houve erros e revisar o código antes de enviar para o commit.

O comando git checkout é um comando para voltar o arquivo a sua versão anterior, caso após o commit você tenha feito alguma alteração. Caso eu tenha adicionado o arquivo ao Stage com git add, eu posso voltar com o comando git reset HEAD <nome_arquivo>, então se eu quiser tirar a mudança, é só de novo utilizar o comando git checkout <nome_arquivo>. Ainda com o comando git reset, eu posso utilizá-lo para voltar caso eu já tenha até mesmo feito um commit, através dos seguintes parâmetros:

- –soft: volta o arquivo para o estado que estava em staged, ou seja, pronto para ir ao commit;
- –mixed: volta o arquivo para o estado antes do staged;
- –hard: volta tudo para o estado do commit anterior.

E então, o comando completo fica: git reset <tipo> <hash_do_commit>

Repositórios Remotos

Aqui é abordado como criar repositórios remotos no github e também como linkar com o repositório git que temos na pasta local.

Para criar o repositório no github é bem simples, e tem um botão de + logo no topo da página, basta preencher as informações de nome e descrição e pronto. Para linkar com o repositório local, o próprio github já ajuda com esses comandos, mas é basicamente:

git remote add origin git@github.com:<Usuario/Repositorio,git>.

Para mandar o que eu tenho para o repositório remoto, é só usar o comando:

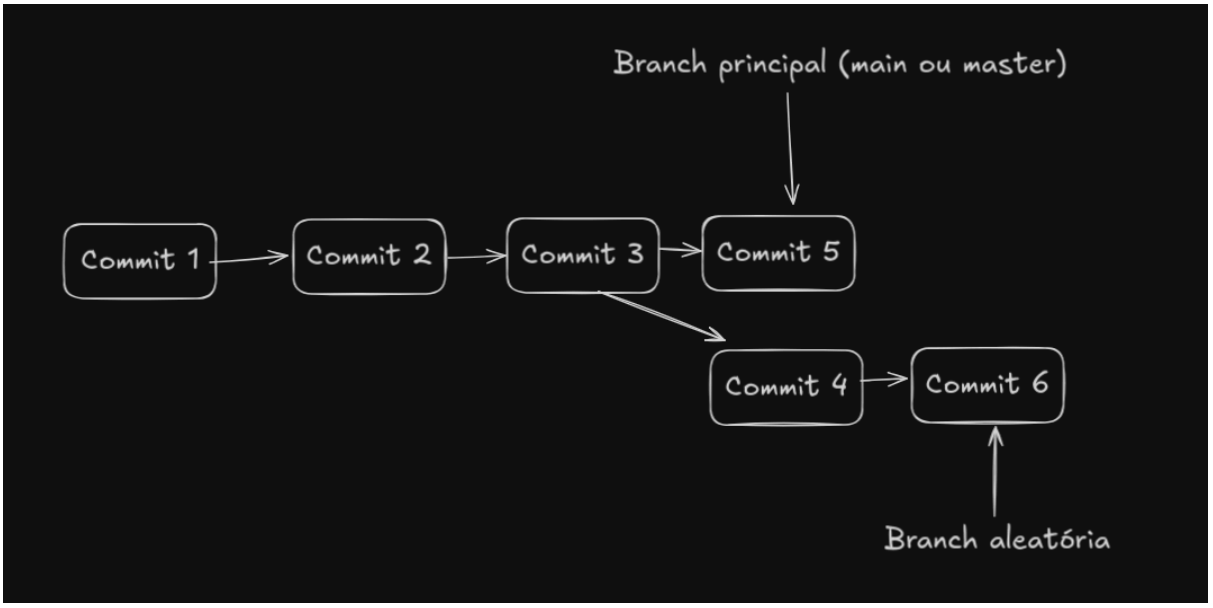
git push -u origin master;

Agora se eu for no repositório no site do github eu encontrarei todo o repositório lá. Caso eu faça alguma alteração no arquivo, basta fazer novamente o commit, e para enviar para o repositório remoto, é só utilizar o comando git push novamente.

Para baixar um repositório remoto, podemos utilizar o comando git clone, seguido do endereço desse repositório, que pode ser https, ssh ou github cli. No meu caso eu gosto de usar o ssh porque é mais rápido e prático pra mim.

Fork é basicamente uma forma de fazer uma cópia de um repositório que não é meu, é diferente do git clone pois o clone é mais como se eu estivesse baixando o repositório, enquanto o fork eu basicamente faço um clone do repositório por completo, podendo subir alterações que eu fizer etc.

Branch



De forma resumida, uma branch é basicamente um ponteiro que aponta para um commit. Ele serve para que o usuário possa, por exemplo, corrigir um bug em um branch, enquanto as pessoas testam coisas em outro branch, dessa forma, não há o risco de um “atrapalhar” o trabalho do outro ou algo do tipo.

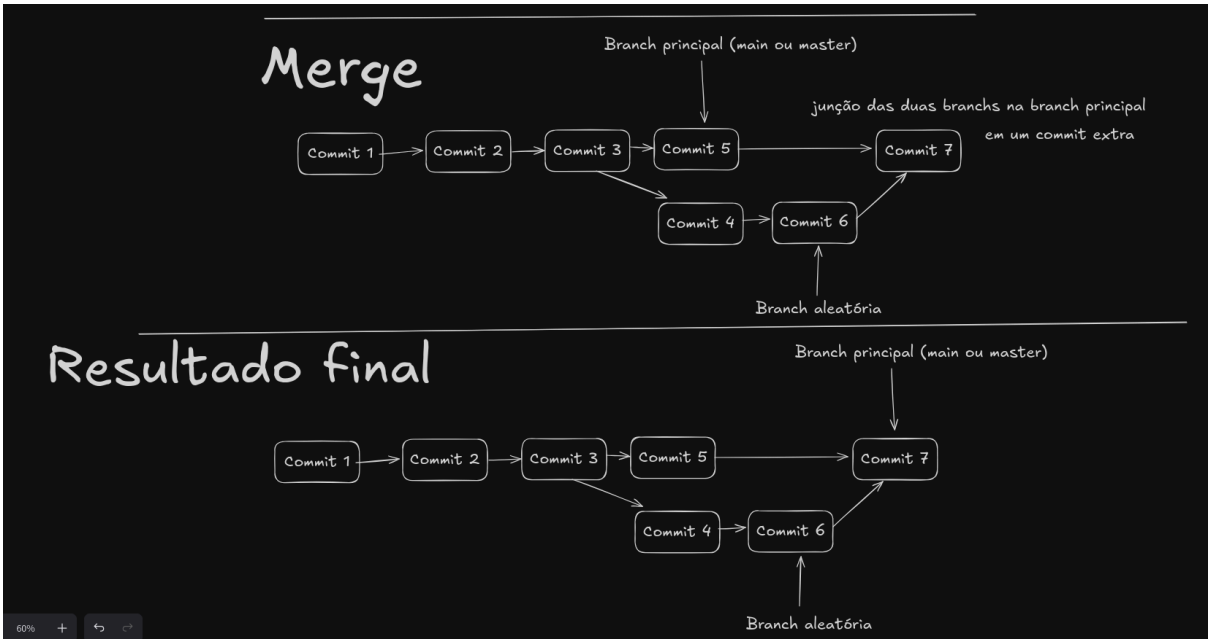
Para criar um novo branch, é só utilizar o comando: “git checkout -b testing”. Com o comando “git branch”, é possível ver todos os branchs existentes e em qual você está no momento.

Para trocar para outro branch, é só utilizar o comando: “git checkout <nome_do_branch>”.

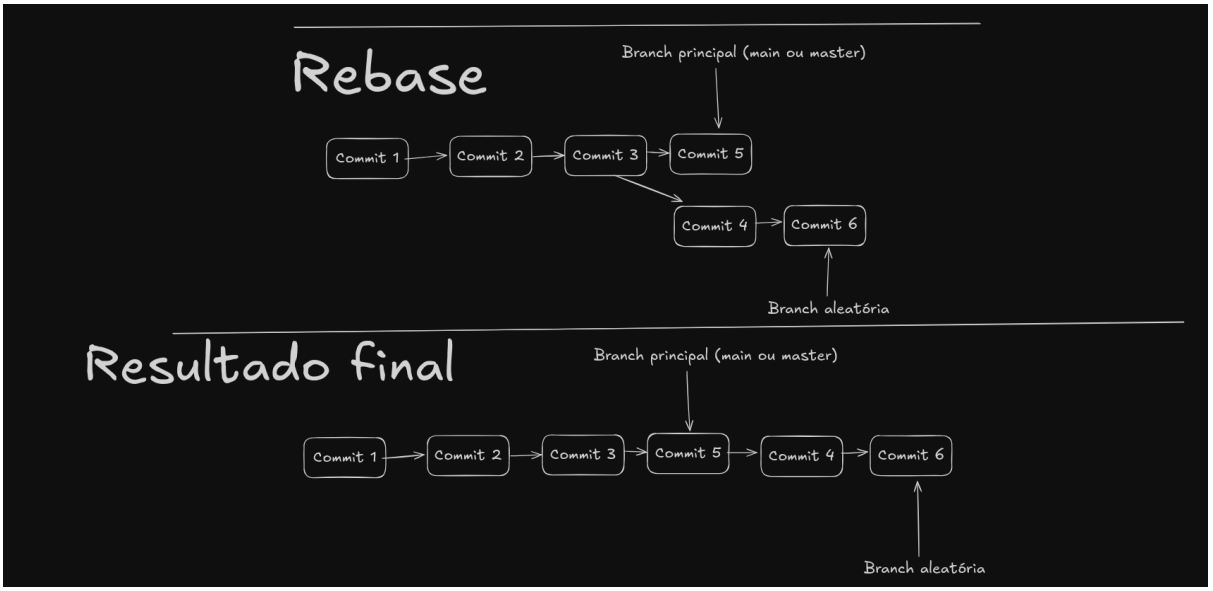
Para deletar um branch, é com o comando: “git branch -D <nome_do_branch>”.

Para unir branchs, temos dois métodos que podemos utilizar: o merge e o rebase.

Dados o cenário em que temos um um branch principal, que é o main, e então criamos uma branch para resolver um problema, porém ao mesmo tempo que fazemos alterações no branch, queremos fazer alterações no main. Utilizando o merge, ele faz basicamente um commit extra que faz a junção da branch criada para a branch principal que estamos para qual estamos jogando as alterações. O merge é bom por manter essas branchs, ou seja, ele não apaga nada que foi feito, porém isso também pode se tornar um problema dependendo da situação, pois cada vez que precisar juntar branchs vai precisar de um commit extra e o histórico do repositório também vai ficar muito poluído com muitas branchs.

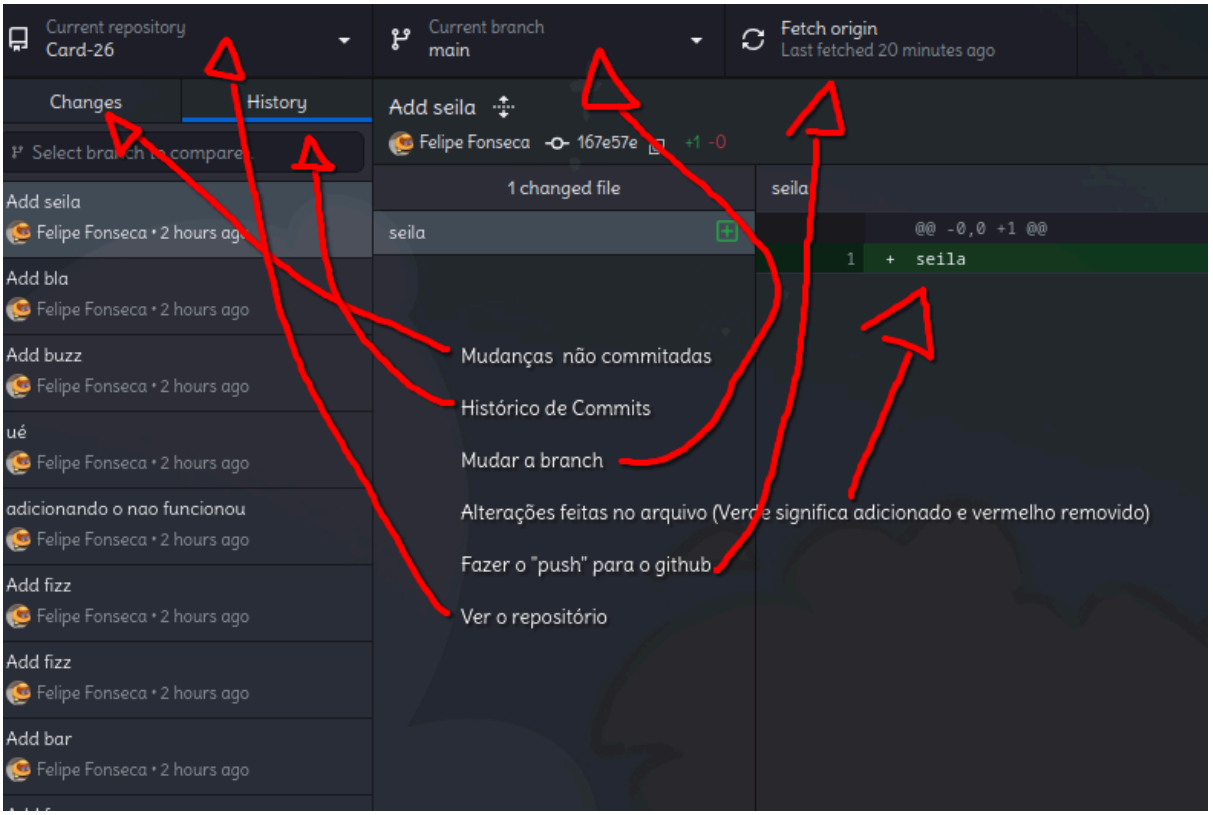


Já o rebase apenas move o commit para a frente do último commit da branch onde estamos jogando, ou seja, ele desfaz a “árvore” do branch, unindo as branchs de forma linear. Ele é basicamente o contrário do merge, pois se o merge tinha como ponto ruim os commits extras e um histórico poluído, o rebase não possui esses commits extras e possui um histórico reto, linear, sem bifurcações, porém isso gera um problema que é a perda da ordem cronológica dos commits. Como os commits são movidos de uma branch para a frente da outra, o usuário pode acabar se perdendo com facilidade na ordem em que os commits foram feitos e na ordem das alterações também.



Github Desktop

Um dos problemas da utilização do git, é que ele pode ser muito complexo para iniciantes, isso porque toda sua configuração e uso é via comandos no terminal. Para facilitar o processo, foi criado o github desktop, que é uma interface gráfica que permite visualizar o repositório de forma mais fácil, vendo as mudanças que foram feitas antes de commitar, também é possível ver todo o histórico de commits, ver as diferentes branchs e as alterações feitas em cada uma delas etc. Cada alteração feita em um arquivo também é possível de ser visualizada, então é realmente uma aplicação bem útil para quem ainda não tem experiência com comandos.



Nesse vídeo, o apresentador fala de um recurso que não havia sido citado no curso anterior, que é o gitignore. Basicamente é um arquivo na qual você pode colocar todos os arquivos que você não quer que sejam mandados para o repositório. Podem ser configurações privadas ou arquivos de instalação, qualquer coisa que não seja segura mostrar ao público.

Prática

Iniciei entanto brincando com o gitignore, criando o arquivo na pasta que eu clonei do repositório do card com o github desktop:

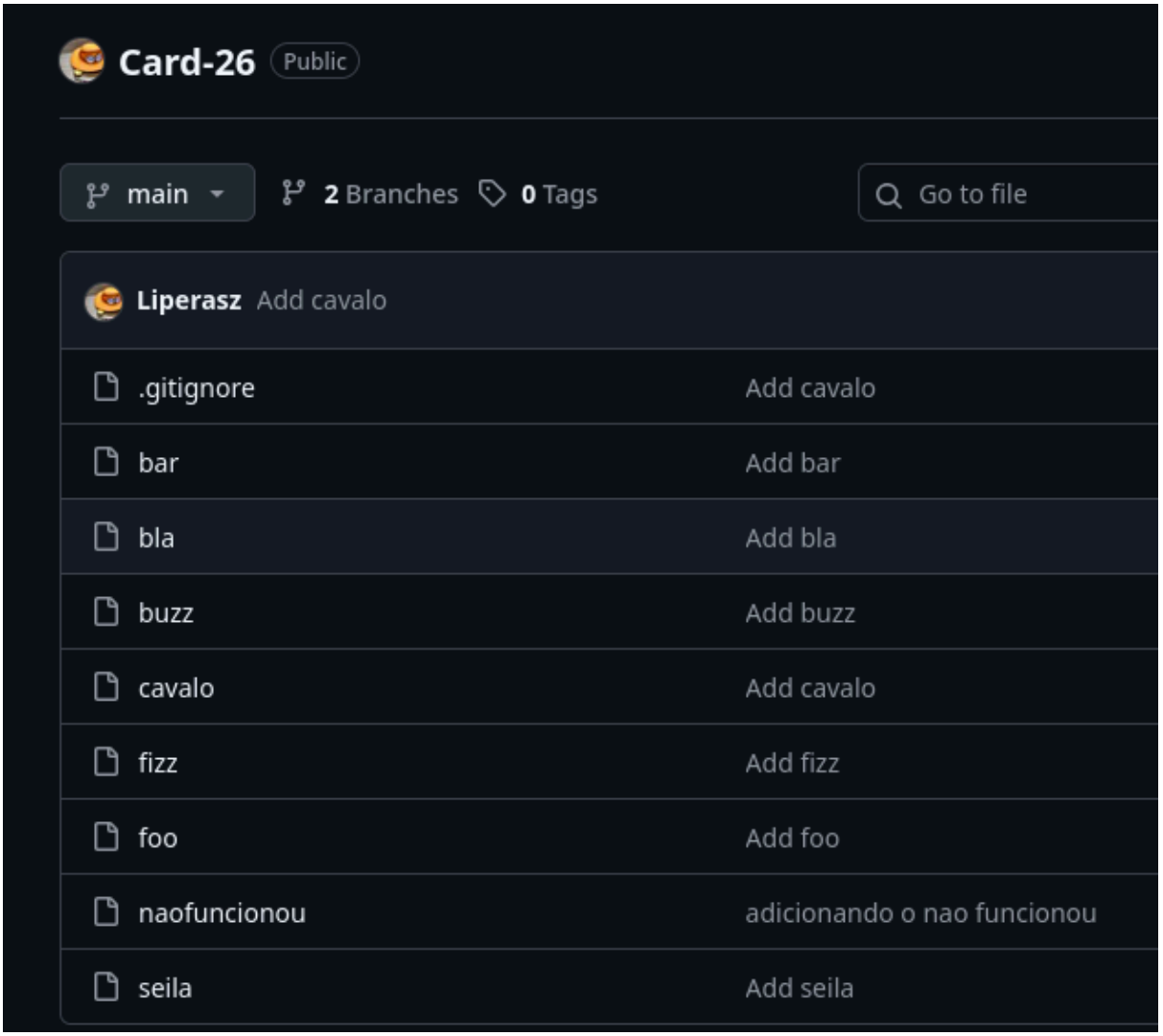
Criei dois arquivos, um chamado cavalo e um chamado de informacao-pessoal. O arquivo chamado informacao-pessoal eu coloquei no gitignore, a expectativa é que ele não seja adicionado no github. Para testar, utilizei os comandos no terminal mesmo.

- git add . (adiciona todas as alterações no repositório)
- git commit -m “”
- git push

Opa, apareceu um erro, que é o erro de autenticação com o github. Afinal ele pede o email e senha mas essa forma de autenticação já não é mais aceita assim. Para resolver isso, tive que fazer acesso remoto via uma chave ssh (eu já havia configurado ela a muito tempo, mas é uma chave que permite acesso remoto). Então eu apenas utilizei o seguinte comando para fazer o link:

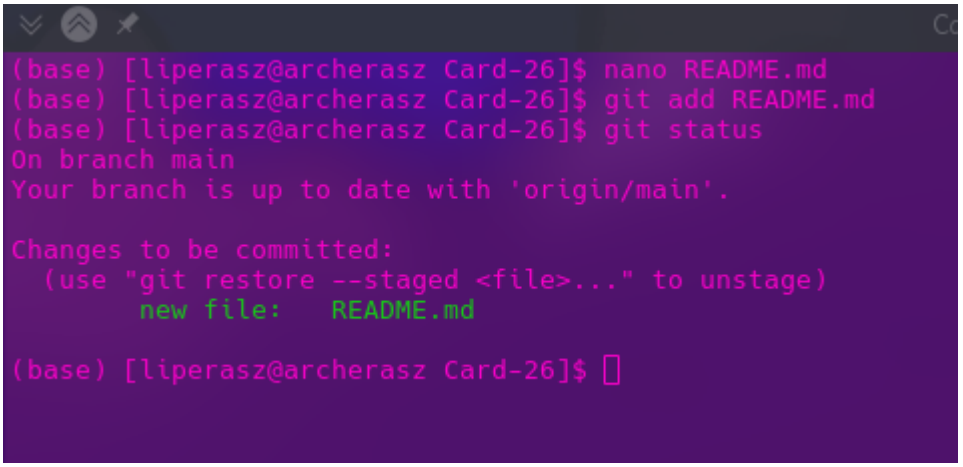
```
git remote set-url origin git@github.com:Liperasz/Card-26.git
```

E agora sim eu faço o git push.



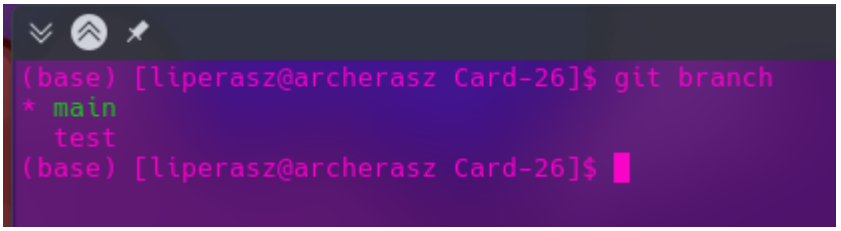
Como é possível ver, o arquivo informacao-pessoal não foi adicionado, isso porque ele está sendo marcado para ser ignorado pelo gitignore.

Como em determinado momento durante a aula eu exclui todos os arquivos que tinham, eu novamente vou criar um README.md para o repositório. Utilizando o comando nano README.md no terminal eu posso criar esse arquivo e modificá-lo. Eu adicionei o arquivo e utilizei o comando git status para visualizar:

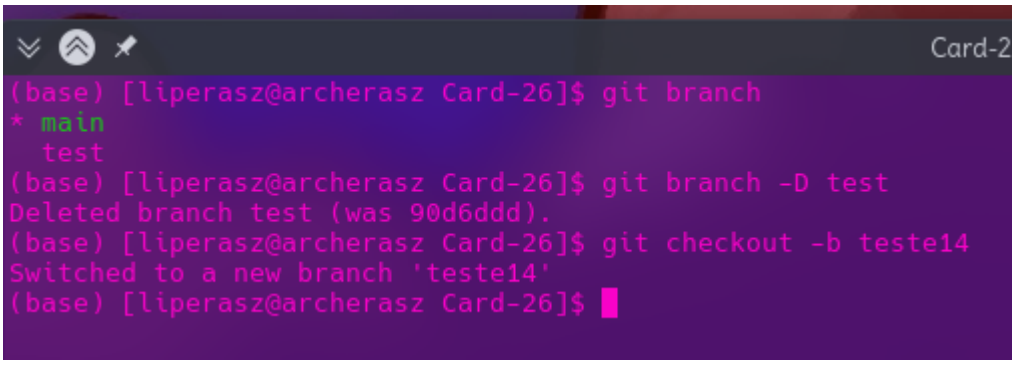


Eu fiz o commit e o push desse arquivo readme.

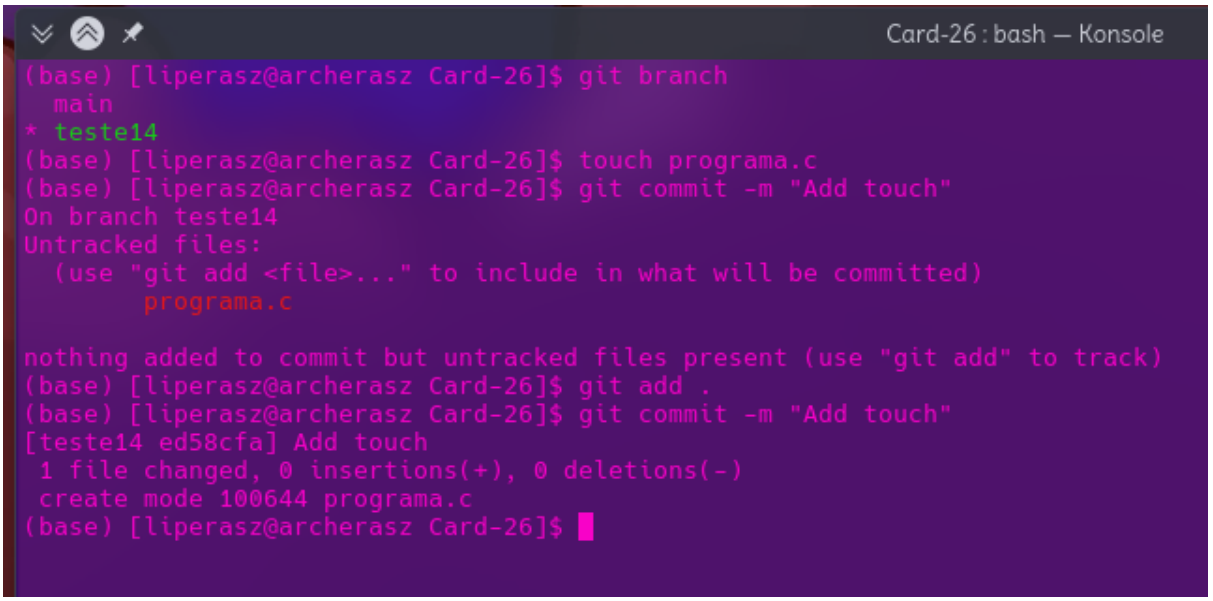
Eu posso olhar as branchs que eu tenho, utilizando o comando git branch:



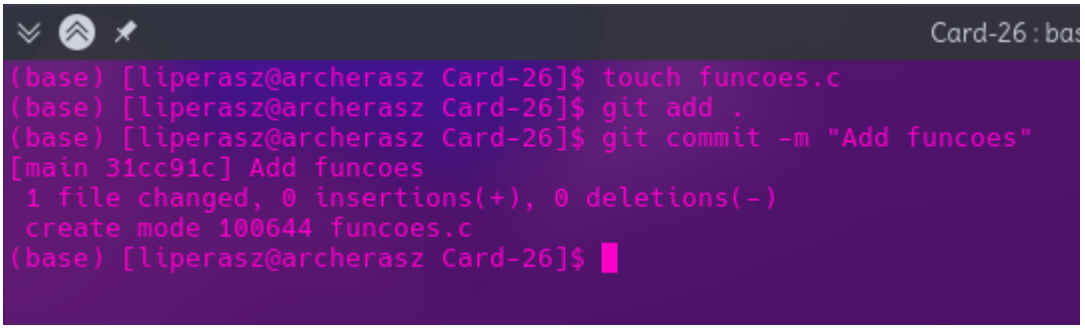
Eu vou deletar essa branch test e vou criar agora uma branch chamada teste14. Com o comando git branch -D test eu deleto a branch test, e com o comando git checkout -b teste14 eu crio uma branch nova chamada teste14.



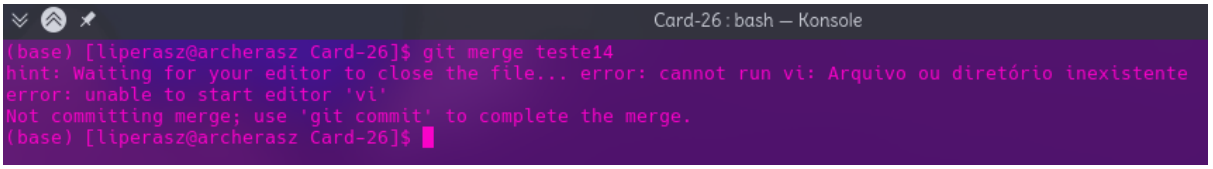
Eu fui criar um arquivo chamado programa.c, e eu quis fazer o commit dele. Porém como eu fui fazer um commit sem alterações, o próprio git já me avisou que tem um arquivo que talvez eu queira incluir, e foi o que eu fiz, então depois eu fiz o commit:



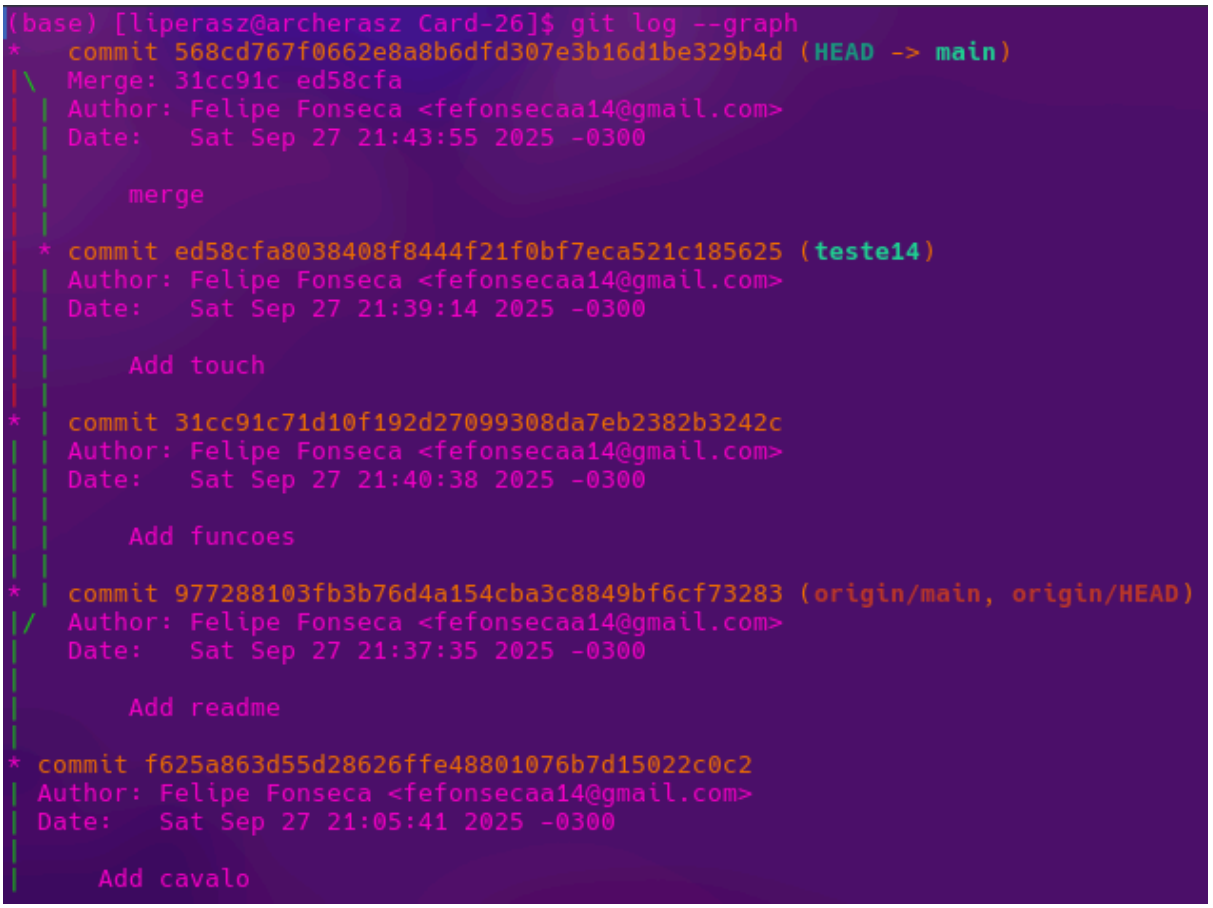
Voltando pro main, eu fiz agora o mesmo mas com um arquivo chamado funcoes.c:



Vou realizar a mesclagem via Merge, usando então o comando git merge teste14.



Por algum motivo essa mensagem aparece, porém eu não entendo o motivo e aparentemente o código funciona do mesmo jeito. Vou usar o git log –graph para ver se funcionou:



Como dá pra ver aqui, aparentemente funcionou sem problemas, realmente não sei o que significa aquela mensagem.

Dois comandos interessantes que não foram citados no curso são: git fetch e git pull. Git fetch basicamente baixa os commits do repositório remoto sem alterar os arquivos

locais. Enquanto isso, o git pull é basicamente uma mistura do git fetch com o merge, pois ele deixa o repositório local igual ao remoto.

Tem algumas instruções que também não foram citadas no curso, como a possibilidade de forçar um push ou um pull, ignorando qualquer restrição ou conflito (não é muito recomendado mas dá pra fazer).

Conclusões

Nesse card foram abordados os conceitos de git e github e a diferença entre eles, o que mostra que na verdade são duas coisas bem diferentes, afinal um é uma ferramenta e o outro é uma plataforma. Também foram mostrados muitos comandos git que são muito úteis para a manutenção de um projeto, pois é realmente importante que se possa ter esse controle de versão.

A utilização de branchs foi a parte mais confusa de entender, principalmente na hora de juntar com merge ou rebase, mas acho que deu pra entender legal.

Além disso, foi mostrado a interface do github desktop, eu utilizei um pouco e achei interessante a ideia, é realmente bem fácil de utilizar. Porém tendo em vista que muitas vezes o trabalho será feito através de servidores ou em computadores mais fracos (como meu notebook), eu irei continuar utilizando apenas os comandos via terminal mesmo, apesar de ser mais difícil de aprender no começo, tenho certeza que a longo prazo será muito mais eficiente.

A conclusão final mesmo é de que o git e o github são uma combinação muito poderosa e de que é quase que indispensável para qualquer projeto, tanto pessoal mas principalmente projeto coletivo. É uma forma extremamente útil para programar em conjunto e também para controlar o andamento do projeto.

Referências

Pesquisa dos comandos: https://gist.github.com/leocomelli/2545add34e4fec21ec16;

cursos da pasta;

github desktop: https://www.youtube.com/watch?v=Fj3gtbaF8WA.