**Git e GitHub**

* Resumo dos códigos

git diff = verifica se tem modificação, informa tb se foi adicionado para área de staged(git add .)

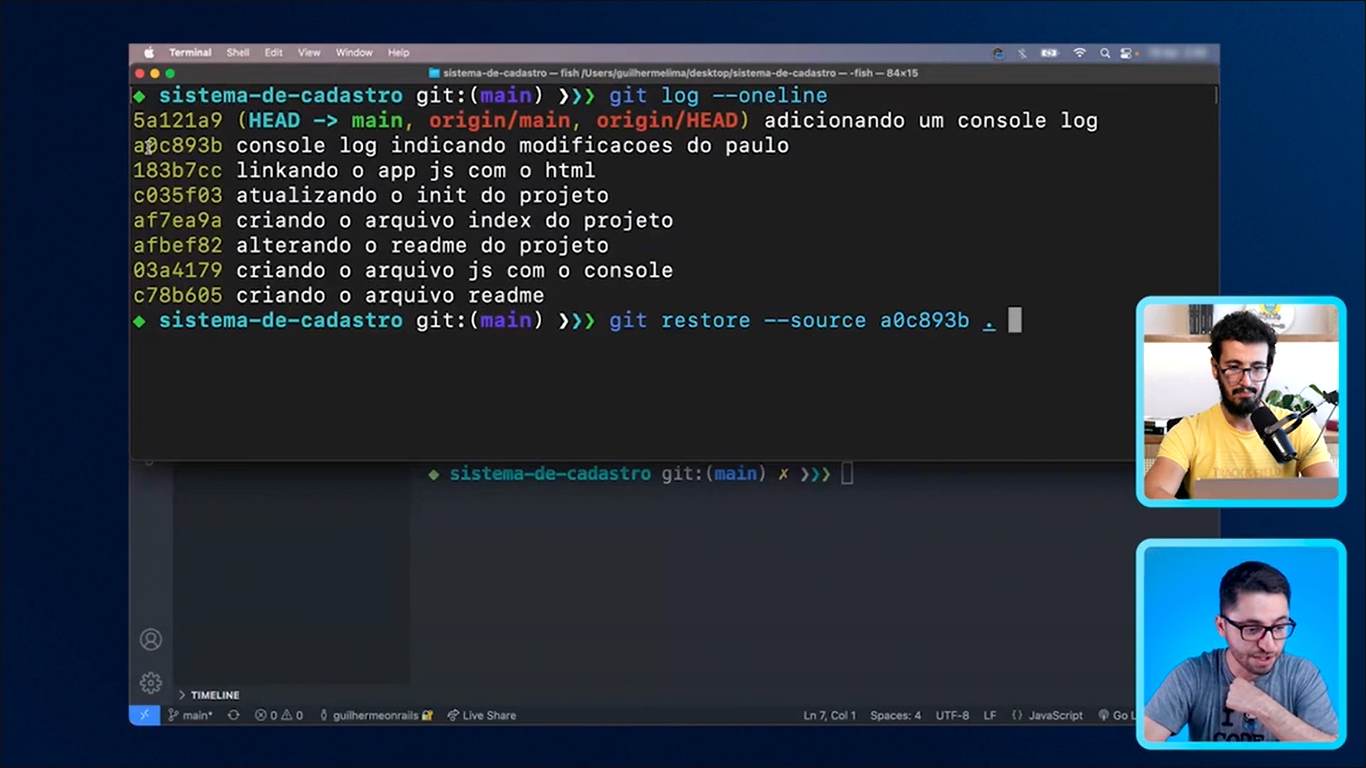
git restore –staged .\nomeDoArquivo = retorna o arquivo para área de não salvo (antes de git add .)

git log = mostra os últimos commits

git fetch = verifica o que está alterado no github sem baixar as modificações, sem fazer pull. Depois precisa fazer um git diff origin/main  
git log --oneline = mostra os todos os commits

git commit . -m “teste de commit” = o ponto indica que será feito commit em todo o sistema, se for usado o ponto no “git add .”, entã não precisa usar o ponto no git commit -m “frase”.

git restore --source a0c893b . = volta no o codigo no push desejado, usando o codigo hash. Esse código pode ser obtido usando o git log --oneline. Obs: o ponto indica retornar todo arquivo. Depois fazer commit e push.



=======fazendo modificações apenas em um arquivo, muito comum no trabalho em grupo. ==========

git add app.js

git commit app.js -m “modifiquei o app.js” = o commit será feito apenas no arquivo app.js

git push

====================================================================================

========== Branch: criando, alterando e excluindo (git checkout -b nome-da-branch) =============

1. Caminho 1: criar uma branch para cada aula do curso com o comando:

Com esse comando, você cria uma nova branch e muda automaticamente para ela para dar início ao desenvolvimento.

git checkout -b nome-da-branch

1. Caminho 2: criar uma branch para cada aula do curso, mas não entra na branch nova criada:

git branch nome-da-branch

1. Consultar as Branch

git branch

Essa é outra forma de criar uma branch. Nesse caso, ela é criada, mas não há a mudança automática para esta nova ramificação. Para isso, você pode usar o comando

git switch nome-da-branch.

Criando uma Branch de exemplo ‘desenvolvimento’.

git checkout -b desenvolvimento

Alterando entre Branch

git switch desenvolvimento. Ou git switch main Ou git checkout nomeDaBranch

verifica em qual branch estamos e quantas branch existem, obs: identifica onde vc está pelo nome HEAD.

git log --oneline --decorate

Depois Add, commitar e push

git add contato.html, ou git add .

git commit -m "adicionando email de contato"

git push origin desenvolvimento

==========Unir as Branch, o nome desse processo é ‘Marge’ =======================

1. Primeiro tem de voltar para a Branch principal (main)

git switch main.

1. Depois unir com a Branch que quiser, no caso desenvolvimento.

git merge desenvolvimento

====================================================================================

* Códigos detalhados

=====================Obter informações dos arquivos=======================

Se você deseja verificar o histórico de alterações, as mensagens de commits, o nome do autor daquele commit e outras informações sobre o projeto, existe um comando do git que pode te ajudar. Este comando é o git log.

Como já sabemos, os commits possuem hashs que os identificam de uma forma única, isto é, não existem dois commits com o mesmo hash. Com o git log podemos ver o hash e várias outras informações do commit.

Podemos visualizar todos os commits, um em cada linha com o comando:

git log --oneline

Se, em vez de menos informações, quisermos ver mais como as alterações do commit, podemos usar:

git log -p

Também podemos pesquisar as informações do autor daquele commit com o comando:

git log --author="user\_name"

E pesquisar informações por data:

git log --since=1.month.ago --until=1.day.ago

No comando acima, estamos buscando as informações do commit desde um mês atrás até um dia atrás.

Você também pode formatar a visualização das informações de commit com o comando:

git log --pretty="format:%h %s"

exibir as informações de commit, [aqui tem uma lista](https://devhints.io/git-log-format) de maneiras que você pode fazer isso.

Bem legal, não é? E se eu te contar que existem vários outros parâmetros que podemos passar no git log? Se você deseja saber mais sobre como exibir as informações de seus commits, você pode conferir [neste link](https://devhints.io/git-log).

===============================Git clone================================

Imagine que você esteja trabalhando em um projeto que já está configurado em um repositório de origem, e deseja colaborar com esse projeto. Com o git clone, é possível criar uma cópia de desenvolvimento em um repositório local, e todas as alterações que você fizer serão gerenciadas a partir desse repositório. O comando git clone é usado para selecionar um **repositório existente** e criar um clone ou uma cópia dele em um **repositório local**.

O comando git clone cria uma cópia de um repositório git existente, e esse repositório pode ser **local** ou **remoto**. Além disso, essa cópia é um **repositório git completo**, com seu próprio histórico, gerenciamento de seus próprios arquivos e é um ambiente isolado como um todo do repositório original.

Por conveniência, a clonagem cria uma **conexão remota** apontando para o repositório original. E é essa conexão que facilita muito a interação com o repositório central. Você pode consultar um exemplo demonstrando o git clone [aqui](https://www.atlassian.com/br/git/tutorials/setting-up-a-repository)!

Com o git clone você também pode clonar o repositório para uma pasta específica:

git clone <repositorio> <meu-projeto-clone>

O repositório localizado em repositorio é clonado para uma pasta chamada meu-projeto-clone.

Você também pode configurar o git clone e clonar o repositório a partir de uma **branch** específica, diferente da original dessa forma:

git clone -branch new\_feature <repositorio>

O exemplo acima clonaria apenas a **branch** new\_feature de repositorio. Outras configurações de opções do git clone você pode consultar [neste link](https://git-scm.com/docs/git-clone).