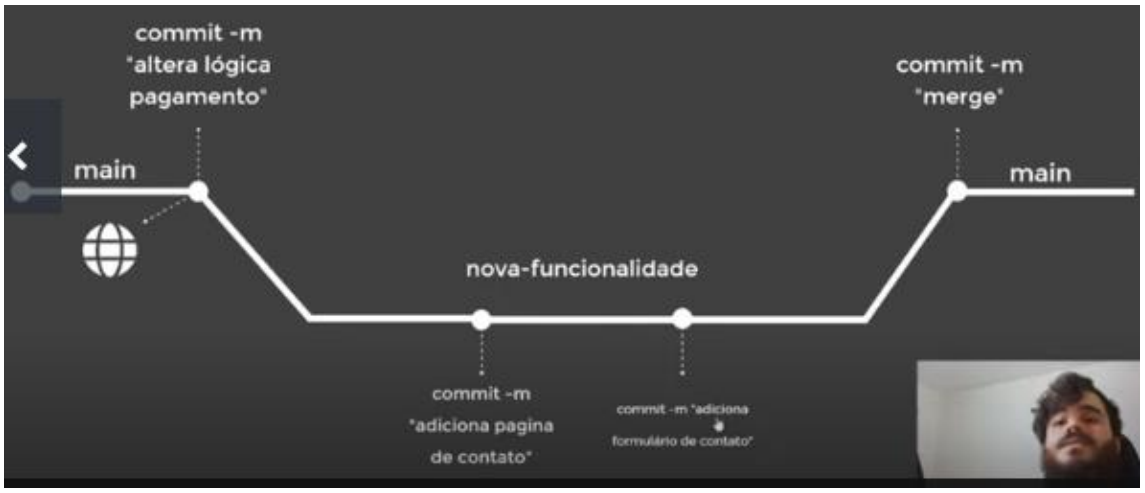
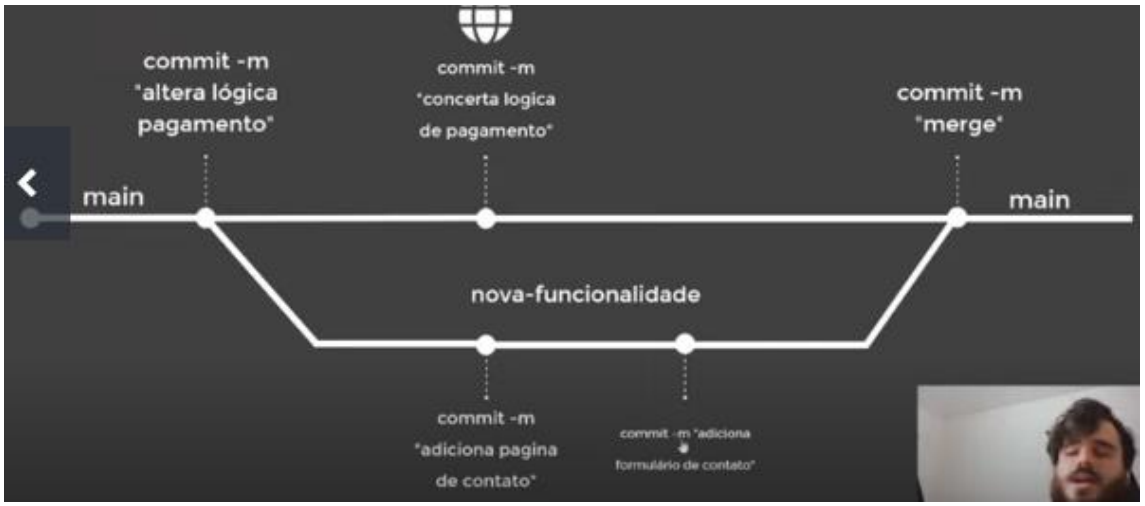
Trabalhando com Branches no GitHub

- **Aula 01: O que são branches (galhos)?**





- **Aula 02: Entendendo na prática sobre branches**
    - Main ou Master?

        - O git decide o nome da branch para evitar nomes pejorativos (padrão: main), mas o usuário pode escolher previamente o nome de sua branch.

- **Aula 03: Guiando-se nas branches com a tag HEAD**
    - Para resgatar o código já enviado no GitHub → **git pull origin main**
    - Para ver as branches → **git branch**
    - Tag HEAD:
        - Tag que aponta o último commit (nos acompanha)

- **Aula 04: Comando checkout e merge**
    - Para se movimentar nas branches → **git checkout <nome branch>**
    - Para se movimentar nas branches e criar uma nova → **git checkout -b <nome branch>**
    - Para juntar os códigos → **git merge <nome outra branch>**

- **Aula 05: Comando stash e seus subcomandos**
    - Obs.: Index == Stage Area
    - Para mudar o nome de uma branch (dentro da própria) → **git branch -m <nome novo>**
    - Para mudar o nome de uma branch (fora da própria) → **git branch -m <nome antigo> <nome novo>**
    - Para deletar uma branch → **git branch -d <nome branch>**
    - Para usar o stash (serve para deixar as branch limpas ao realizar as transições entre elas) → **git stash save "comentário"**
    - Para ver os stash → **git stash list**
    - Para estourar o stash → **git stash pop <índice (ex.: 1)>**

- Para limpar o stash → **git stash clear**

- **Aula 06: Comando git log**
  - Para visualizar o histórico de commits → **git log**
  - Para sair da visualização do git log → pressione "**:Q"**
  - Para visualizar o log de uma pasta/arquivo apenas → **git log <nome pasta ou nome arquivo>**

- **Aula 07: Subcomandos específicos com git log**
  - Para visualizar o resumo do log em uma linha → **git log –oneline**
  - Para visualizar os logs de forma "ilustrada" → **git log –graph**
  - Para visualizar os logs de forma realmente gráfica (Windows) → **gitk**
  - Outra forma de visualizar os logs de forma gráfica → baixe o **GitHub Desktop**

- **Aula 08: Conceitos iniciais sobre reverter commits**
  - Git revert x git reset:
  - Para reverter um commit → **git reset <hash do commit> ou git reset <HEAD~numero>**
  - Git reset (flags):
    - -- soft
    - -- mixed
    - -- hard

- **Aula 09: Comandos para reverter commits (parte 1)**
  - **git reset <hash do commit> ou git reset HEAD~1** (~1 significa que o HEAD deve voltar uma casa)
  - **git reset:**
    - **-- soft <HEAD~num>** → move os commits para o **staging/index** (como se tivesse rodado o *git add* *)
    - **-- mixed** <*HEAD~num*> → (default) move os commits para o **working dir.** (como se não tivesse rodado o *git add* *)
    - **-- hard** → (destrói código: CUIDADO!) remove os commits, não os move.
  - **git revert <hash do commit> ou git revert HEAD~1** (~1 significa que o HEAD deve voltar uma casa)
  - **git revert:**
    - não precisa passar as flags, ele trabalha apenas com os commits e não com os arquivos
    - inverte os commits (ex.: adicionando → removendo)
  - **git reset/revert no diretório remoto:**
    - **git reset:**
      - git reset <hash do commit>
      - git push origin main (falha)
      - git pull origin main
      - corrige para o merge...

        - git push origin main (sucesso)
    - **git revert:**
        - git revert <hash do commit>
        - git push origin main (“sucesso”, precisa corrigir para o merge)

- **Aula 10: Conceitos iniciais sobre estruturação de commits**
    - Por que me importar?
        - Melhor legibilidade do histórico
        - Amigável para novos desenvolvedores
        - Amigável ao versionamento semântico
    - De forma geral:
        - Commits atômicos (pequenas alterações em um commit só)
        - Português x Inglês (depende da empresa e da convenção)
    - Estrutura:
        - Assunto:
            - Curto e compreensível
            - Até 50 caracteres
            - Começar com letra maiúscula
            - Não terminar em ponto
            - Escrito de forma imperativa (ordem)
        - Corpo:
            - Adicione detalhes ao commit
            - Tente quebrar a linha em 75 caracteres
            - Identifique sua audiência
            - Explique tudo
            - Use markdown
        - Rodapé:
            - Referencie assuntos relacionados

- **Aula 11/12: Praticando estruturação de commits**
    - **git commit** → abre editor de texto para descrever o commit
    - para sair do vim: **:q!**
    - **git config --global core.editor "code --wait"** → configura o editor de texto (no caso, o VSCode)
    - **git config --global core.editor "vim"** → configura o editor de texto (no caso, o Vim)
    - **Utilizando o Vim:**
        - pressione V para entrar no modo VISUAL
        - pressione Delete para deletar tudo
        - pressione Insert para digitar algo
        - pressione Esc para sair do modo EDIÇÂO
        - digite :WQ para salvar e sair

    - closes #1 referencia a issues #1
- **Commits semânticos:**
    - Semantic Versioning:
        - (3.2.7) (major | minor | patch)
            - Major: alterações (grandes) que quebram a compatibilidade
            - Minor: alterações (grandes ou pequenas) que NÃO quebram a compatibilidade
            - Patch: alterações (pequenas como correções de bugs)

    - Estrurura:
        - <tipo> [escopo opcional]: <descrição>
        - [corpo opcional]

        - [rodapé(s) opcional(is)]
- Tipos:
    - fix → se relaciona ao patch
    - feat → se relaciona ao minor
    - !<fix/feat> → quebra compatibilidade (se relaciona ao major)
- Existem ferramentas para Conventional Commits