**Gui: Graphical user interface**

**Cli: Command line interface**

**Comandos no terminal**

Para praticamente todo o que fazemos no computador tem coimo ser feito diretamente por comandos no terminal.

- Navegar entre diretórios.

- Criar pastas e arquivos.

- Ler arquivos

- Executar comandos.

- E muitos mais...

**Versionamento do código**

Também conhecido como Sistema de Controle de Versão. ‘é, um software que tem a finalidade de gerenciar diferentes versões no desenvolvimento de um documento qualquer.’

Concorrentes do GIT

- CVS – Concurrent version system.

- Subversion

- Mercurial.

O Git foi desenvolvido inicialmente por Linus Tovalds para versionar o código-fonte do GNU-Linux, que ele também foi o criador. Ele considerava os outros versionadores gratuitos ruins.

Características destacadas:

-Suporte consistente para desenvolvimento não lineares.

- Desenvolvimento distribuído

- Compatibilidade com protocolos de sistemas existentes (ex.: HTTP, FTP, SSH)   
- Manipulação eficiente de projetos extensos.

**GITHUB**

É uma plataforma de hospedagem de código com controle de versão usando o GIT.

GitHub também é uma das maiores vitrines para desenvolvedores, onde podemos exibir nossos trabalhos e recrutadores e outros desenvolvedores podem nos encontrar.

Outras opções:

- GITLAB

- BitBucket.

(Aprender comandos no terminal0

- Master: Onde eu estou

- Develop: O que está sendo desenvolvido.

- feature: O que está fora do desenvolvimento.

-= Commit: Um registro de historial do arquivo, Pode ser como uma foto, nomeado pelo programa como uma série de dígitos e pode ser renomeada pelo usuário depois. Em sistemas de controle de versão, como o **Git**, um **commit** adiciona as alterações mais recentes do código-fonte para o repositório, tornando essas alterações parte da revisão principal do repositório. Um **commit** é o ato de enviar e guardar, ou seja, enviar dados ou **códigos** para armazenamento em um banco de dados ou em um sistema de controle de versão. Um novo **commit é** um herdeiro direto do HEAD que em geral **é** o topo do ramo atual e o ramo **é** atualizado para apontar para ele (a menos que nenhum ramo seja associado com a árvore de trabalho onde no caso o HEAD esteja desanexado como descrito em git-checkout[1]).

Para descarregar o GIT: <https://git-scm.com/>

Comandos no terminal:

- **ls:** Lista o conteúdo de um diretório

- **ls -a:** Lista os diretórios, arquivos ocultos e executáveis.

- **pwd:** Mostra o diretório corrente. Exemplo; **/c/Users/usuário**

- **cd [caminho]:** Muda de diretório. **cd caminho/**

**- cd .. :** Volta um diretório acima.

- **cd ~:** Volta para seu diretório /home

**-mkdir:** Cria uma pasta com o nome desejado.

**-mkdir [pasta1] [pasta2]:** Cria duas pastas ao mesmo tempo.

**- mkdir ‘Pasta Separada’;** Cria uma única pasta com o nome Pasta Separada

**- mkdir -p [pasta]/[sub-pasta]:** Cria um diretório e um sub diretório.

**(nota: Digita cd mais uma letra, depois pressiona tab duas vezes para mostrar opções disponíveis por essa letra.)**

**- rm:** Remove Arquivos.

**- rm -r:** Deleta pasta ou arquivo.

**- mv [arquivo] [caminho]:** Move o arquivo para um determinado caminho. Também serve para renomear pastas.

**- cp [arquivo] [caminho]:** Copia o arquivo para o determinado caminho.

**- echo [mensagem]:** Exibe uma mensagem em seu shell.

**- cat:** Escreve no terminal (Ctrl + D em uma linha vazia para sair)

**- cat > [Arquivo]:** Escrever dentro de um arquivo txt.

**cat [arquivo]:** Mostrar conteúdo do arquivo mencionado.

**cat >> [Arquivo]:** Edita o final do conteúdo txt

**- touch [arquivo]:** Cria um arquivo.

**- echo ‘mensagem’ > comando.txt** **=** Cria um arquivo de texto com a mensagem: ‘mensagem’.

**- help [comando]:** Mostra todos os parâmetros que o comando permite ser usado. Exemplo: **help cd.**

**- man [comando]:** Manual de comando

**- clear:** Limpar o terminal

**- history:** Ver o histórico de comandos

**- history [n]:** Ver os [n] últimos comandos.

**- history -c**: Apagar o histórico.

- **tree**

**- tree-a**

<http://linuxcommand.org/tlcl.php>

**(Aula extra do git)**

**- git init :** O GIT inicia. Mais os arquivos ainda estão em status de ‘untracked’

**- git add [arquivo]:** adiciona um arquivo ao GIT

**- git add . :** Adiciona todos os arquivos

**- git status:** Mostra o estado do arquivo.

|  |  |
| --- | --- |
| **git global –global user.name “Your name”** | **Configurações globais** |
| **git global – global user.email “Your email”** |

**- git rm –catched [file]...:** to unstage.

**- git commit -m:** Salva no repositório do GIT

- git branch -M “main”: Mudar master a main.

**-git remote add origin <copiar link> não funciona ctrl + v:** Este faz uma conexão remota

- **git push -u origin main;**

**O que vamos aprender nesta aula?**

- Configurar globalmente o GIT.

- Iniciar um repositório global GIT.

- Entender os 4 estados do ciclo de vida dos arquivos

- Nossos primeiros comandos.

**Status do arquivo** (Ciclo de vida do arquivo)

|  |  |  |  |
| --- | --- | --- | --- |
| **Untracked** | **Unmodified** | **Modified** | **Staged** |

**Mudando o status do arquivo**

**Adicionando o arquivo para o stage**

**- git add [arquivo] :** Adiciona arquivo específico

**- git add .:** Adiciona todos os arquivos

**Retira arquivo do Stage**

**- git rm --cached [arquivo]:** Remove arquivo no rasteado do Staged

**- git restore --staged [arquivo]:** Remove arquivo modificado do staged

**- git restore –staged .**

**Commita/Envia o Stage**

**- git commit -m ‘[Mensagem]’:** Commita com a mensagem escolhida

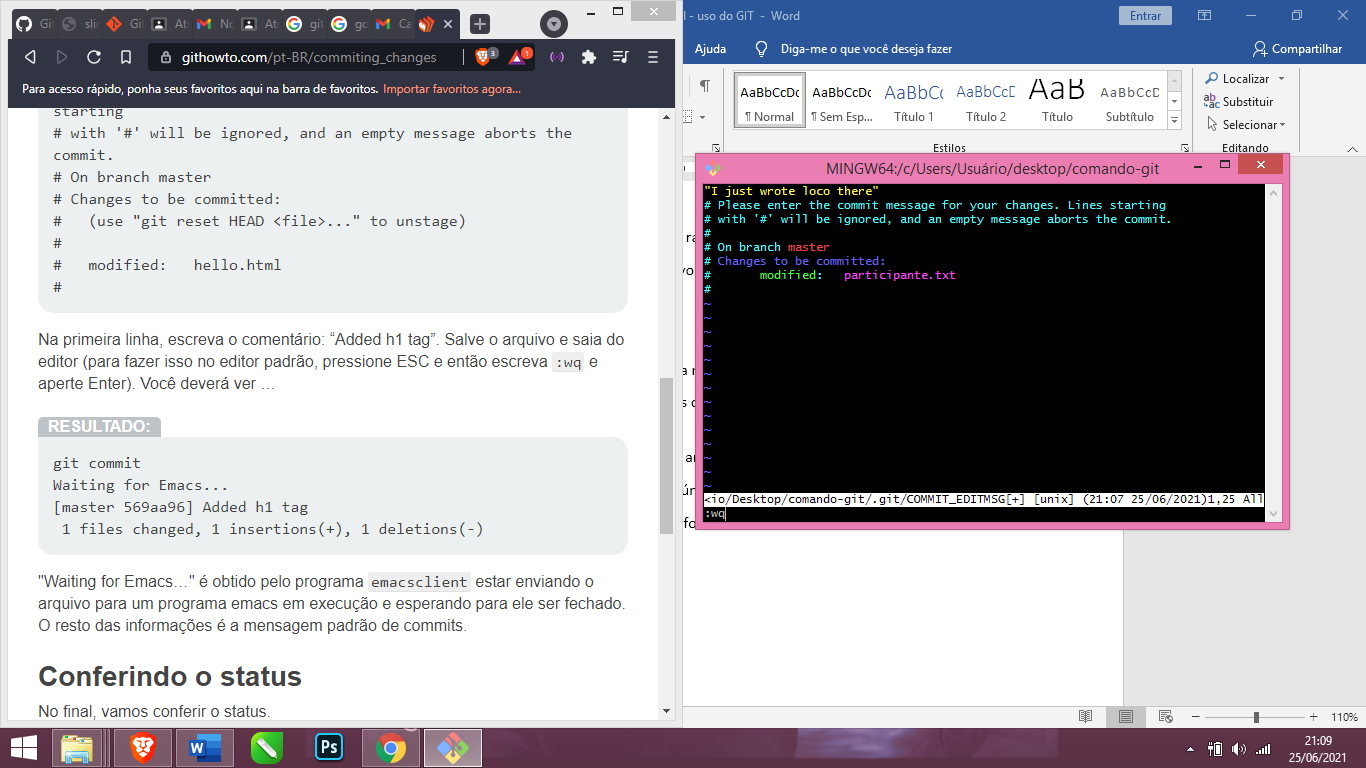
**- git commit -am ‘[Mensagem}’:** Adiciona todos os arquivos modificados e commita com a Mensagem escolhida.

- **history > aula\_3.txt:** Cria um histórico em um arquivo de texto (o Histórico feito no GIT BASH)

**- git log :** mostra o historial dos commit, seus números

**- git log –pretty=oneline:** Mostra o historial de forma mais resumida.

**- git diff>** diferencia entre versões



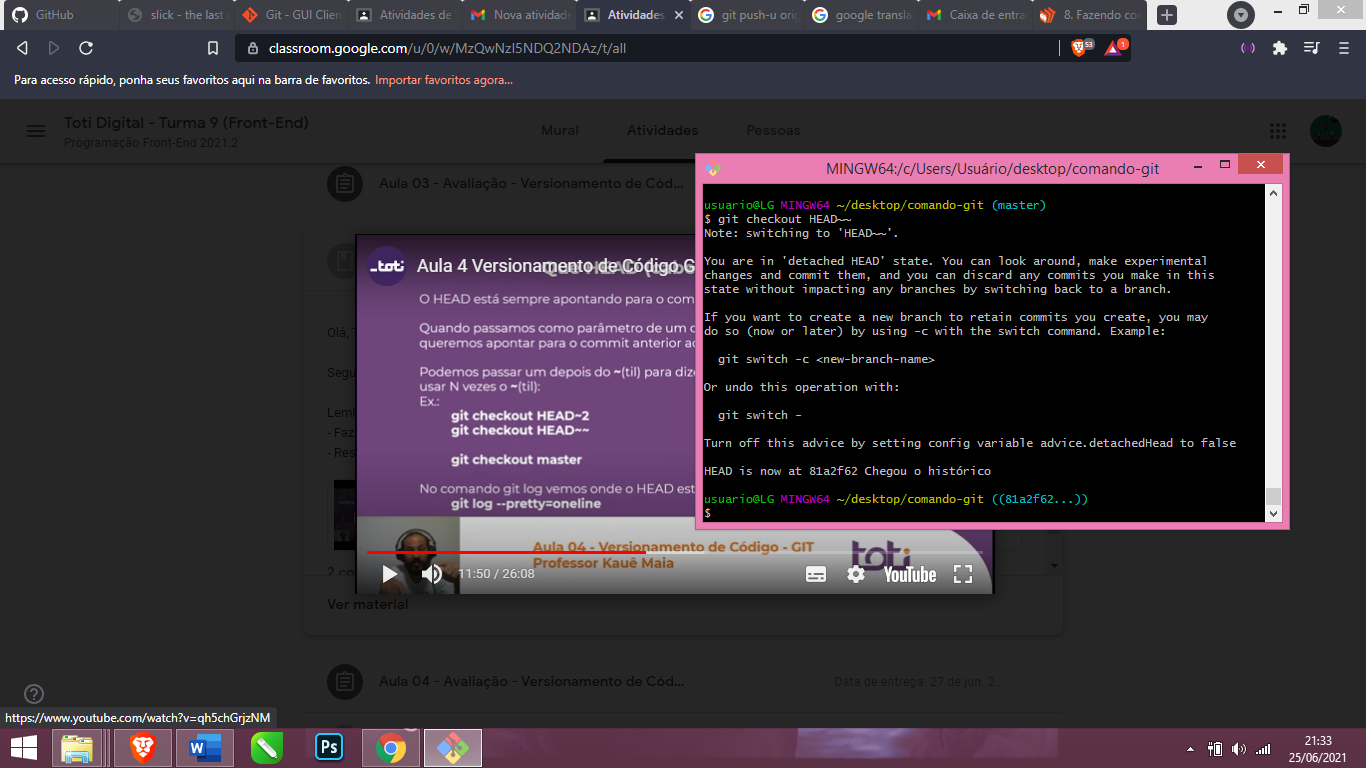
**Quando o commit está pedindo demais...**

**- git log -p:** Mostrar ainda mais detalhado as mudanças do commit.

**-git log –pretty=oneline -2:** Mostra neste exemplo as últimas duas líneas de commit.

**- git log –since=20.minutes:** Neste exemplo o git mostra os commit feitos nesse tempo, no caso, 20 minutos. (week, day, month, etc.)

**- git log –author=’MadJaze’:** Coloca os commits feitos pelo autor mencionado.



**- git checkout HEAD~2:** Volta dois commits.

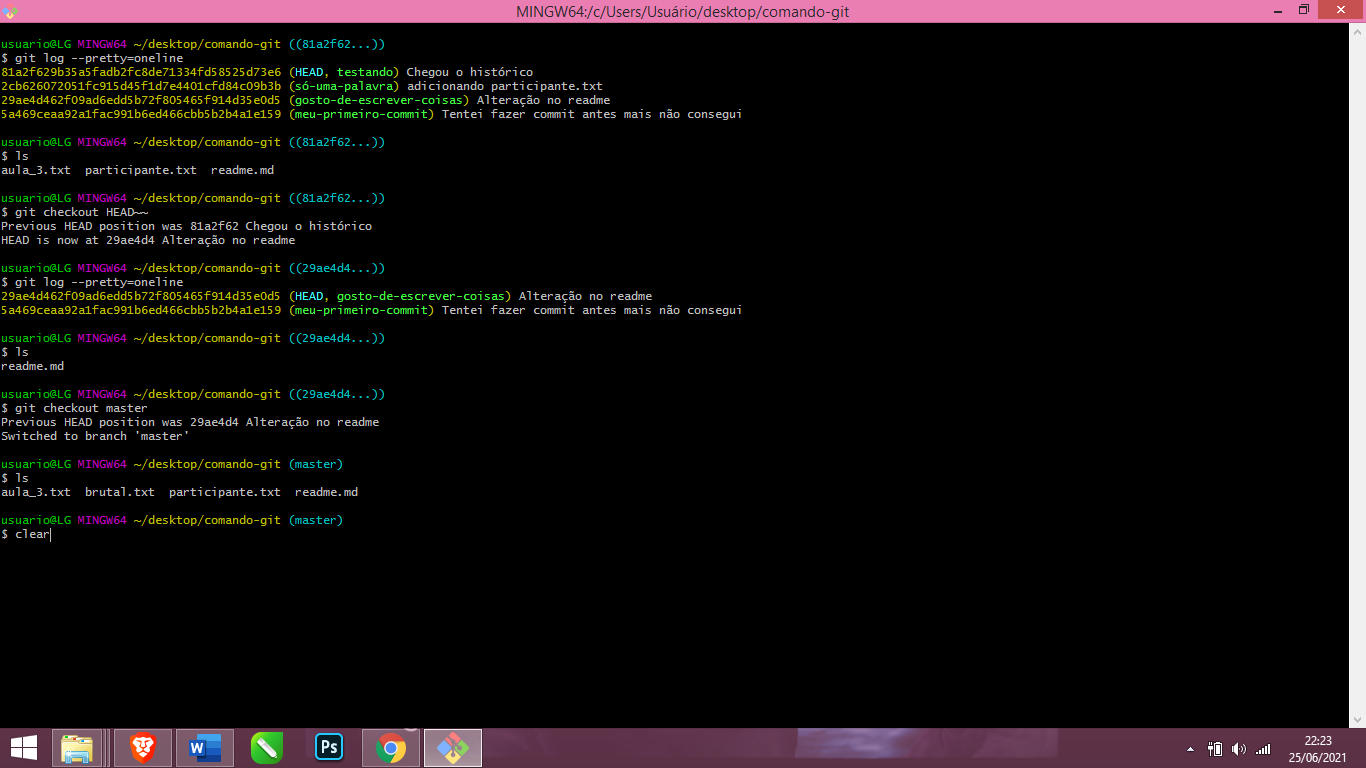
**- git checkout HEAD~~:** Volta também dois commits.

**- git checkout master:** Para voltar a nossa branch padrão.

**- git branch ‘nome-do-alguma-coisa’:** Cria uma branch com esse nome.

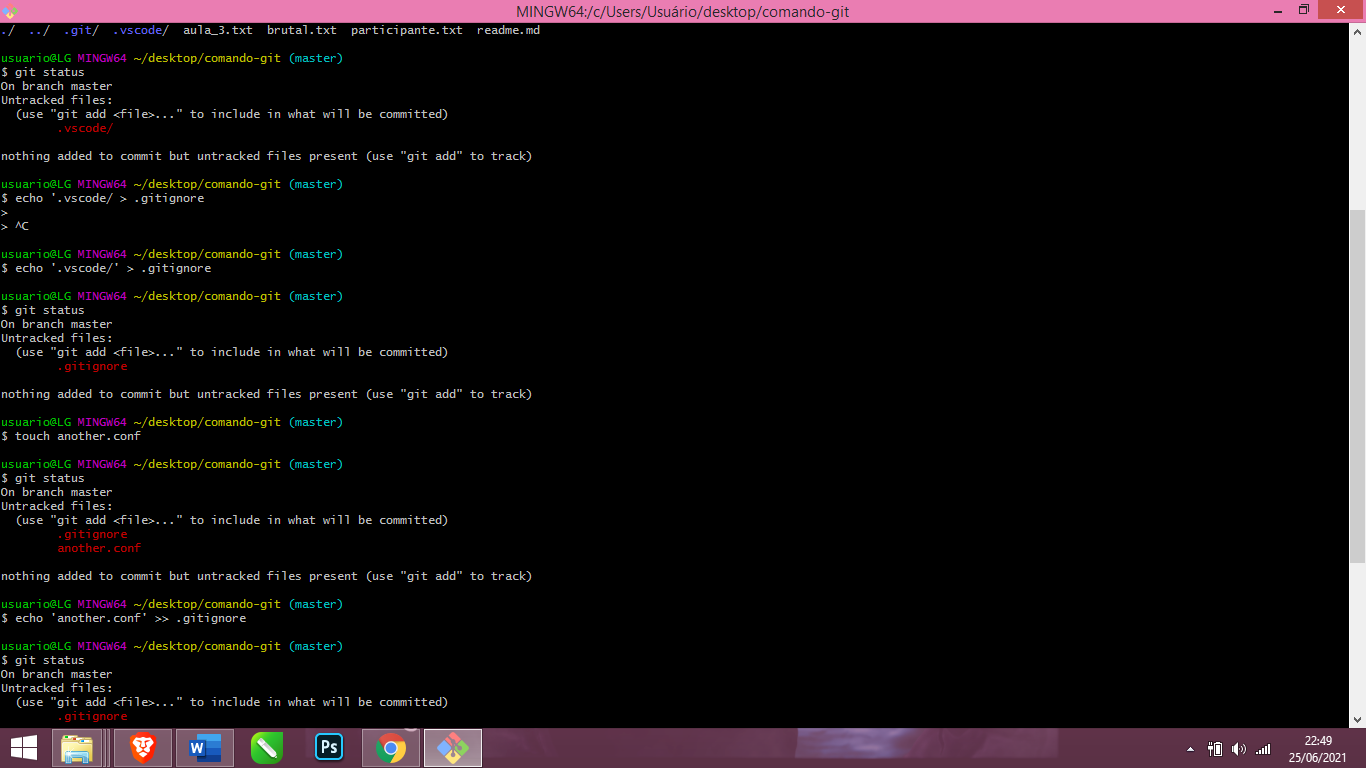
**- git reset –soft HEAD~:** Volta para o commit anterior (o pai do commit que vai depois) este arquivo volta tambem a seu estado anterior (staged).

**- git reset –hard HEAD~:** Voltou ao commit anterior ELIMINANDO o commit que estava depois, e também o arquivo. :c



Em quanto faz **git checkout HEAD~n,** pode observar que o comando **ls** identifica menos arquivos no git.

**Ignorar arquivos**



- Cria pasta com **mkdir .vscode/**

- usa echo **‘.vscode/’ > .gitignore**,   
para começar a ignorar essa pasta.

- Cria arquivo usando **touch config.conf**

- Pode colocar no mesmo arquivo usando  
o echo **’.vscode/’ >> .gitignore**

**- git add .gitignore**

**- git commit -m ‘adicionando arquivo .gitignore’**

**Branchs (vantagens)**

Desenvolvimento independente

- Facilmente acoplável e desacoplável.

- Evita conflitos:

- Multiplas funcionalidades sendo desenvolvidas.

- Multiplas pessoas trabalhando no mesmo projeto

- Organização

**-git branch:** mostra as branchs disponíveis.

**- git checkout -b ‘feature’:** Cria uma branch nova (cambia master a feature: . (git branch ‘loco’)

- **git checkout -b feature2 master:** Cria uma nova branch baseada no master (?)

- **git checkout -d feature2:** Apaga a branch **‘feature2’**

**- git branch -d [nome-da-branch]:** Apaga a mencionada branch, mais tem que estar no momento situado em uma branch diferente.

**Usando o MERGE (fundir)**

**Vantagens:**

- Operação não destrutiva

- Histórico real

**Desvantagens;**

- Commit extra

- Histórico poluído.

- git merge [nome-da-branch]: Uso do merge

- git log –pretty=oneline –graph:

**Usando o REBASE (reescrever):**

Vantagens:

- Sem commit extra.

- Histórico linear

Desvantagens:

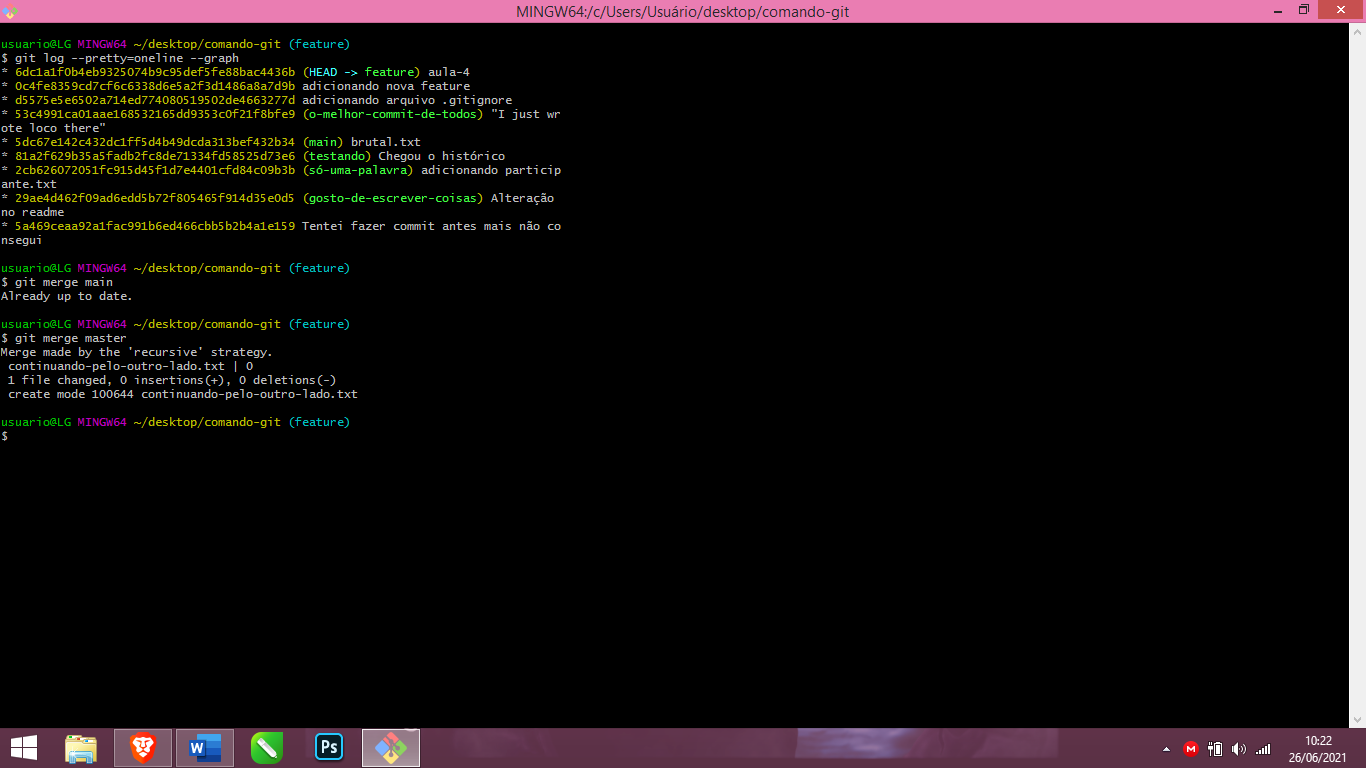
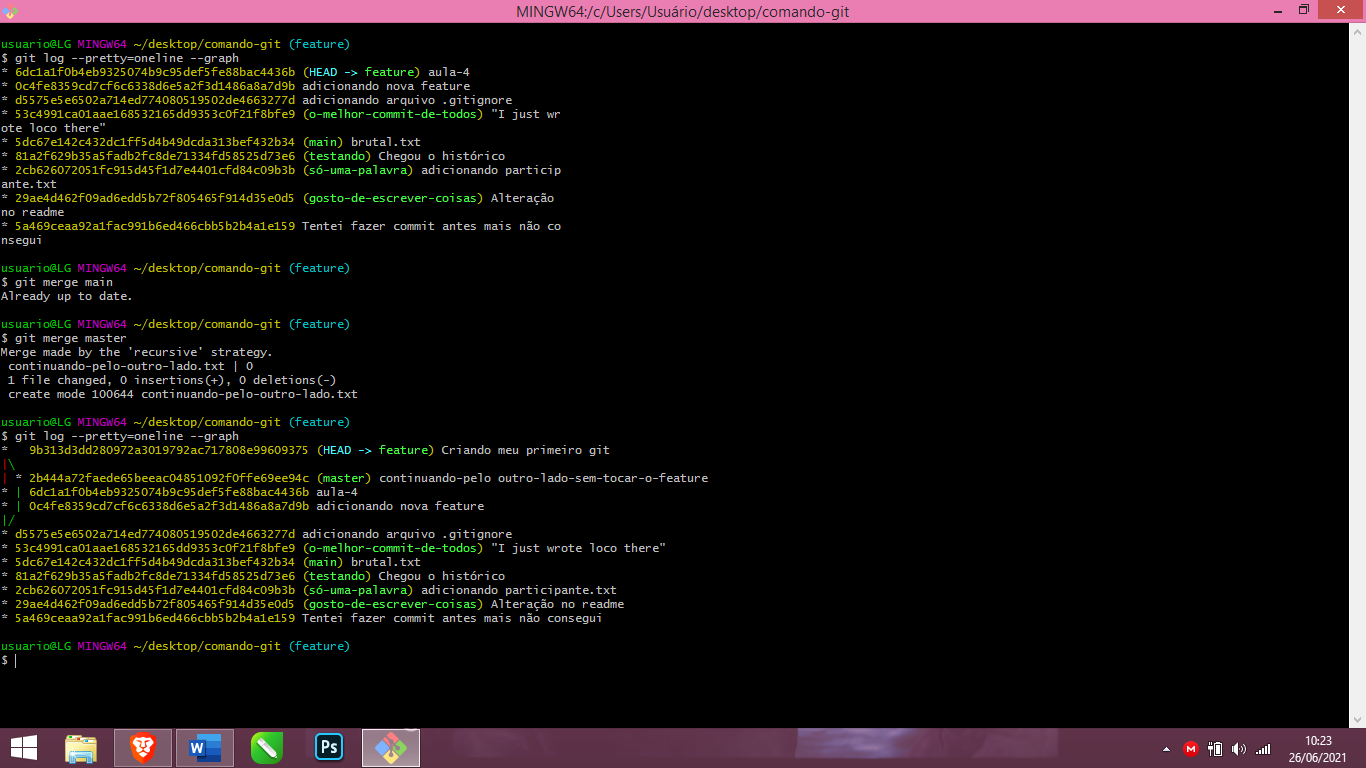
- Perde ordem cronológica

- Pode ter conflitos com outros desenvolvedores.

- git rebase [nome-da-branch]

- git log –pretty=oneline –graph

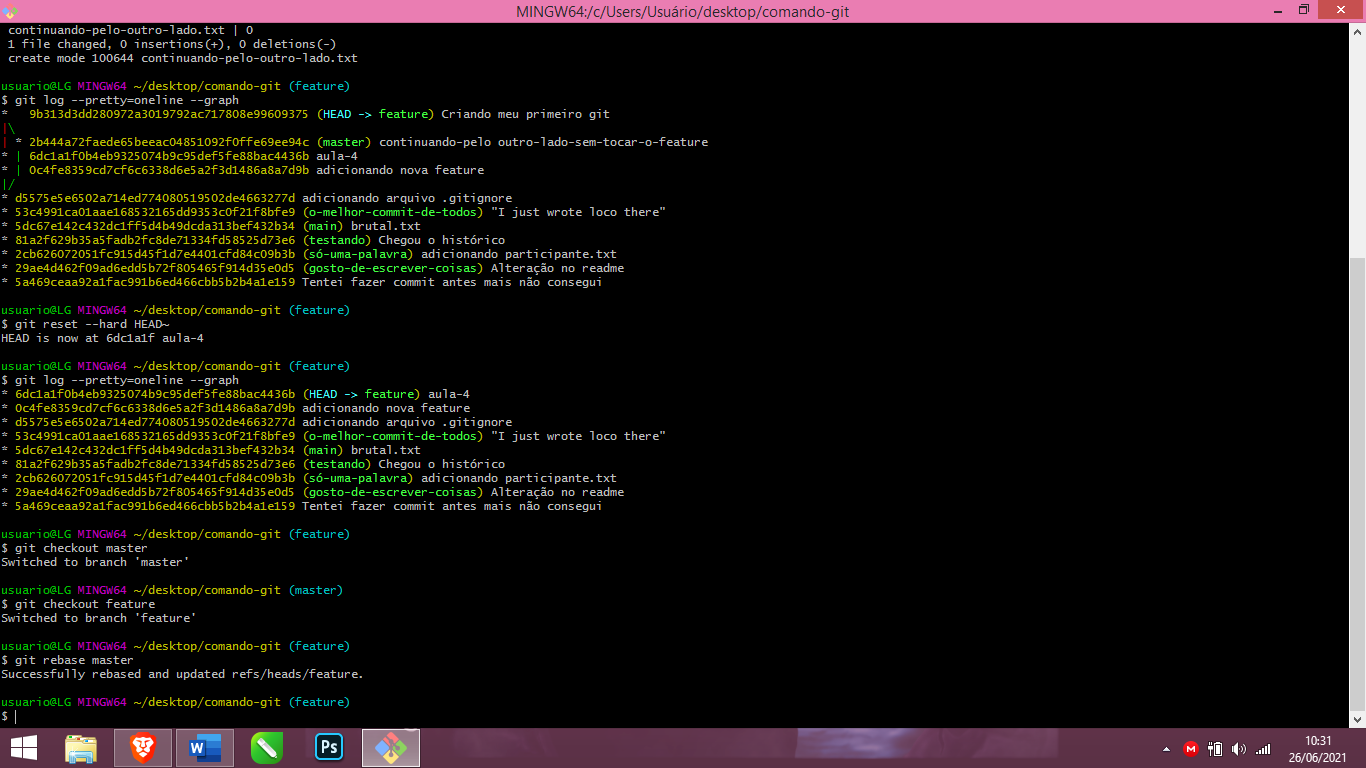
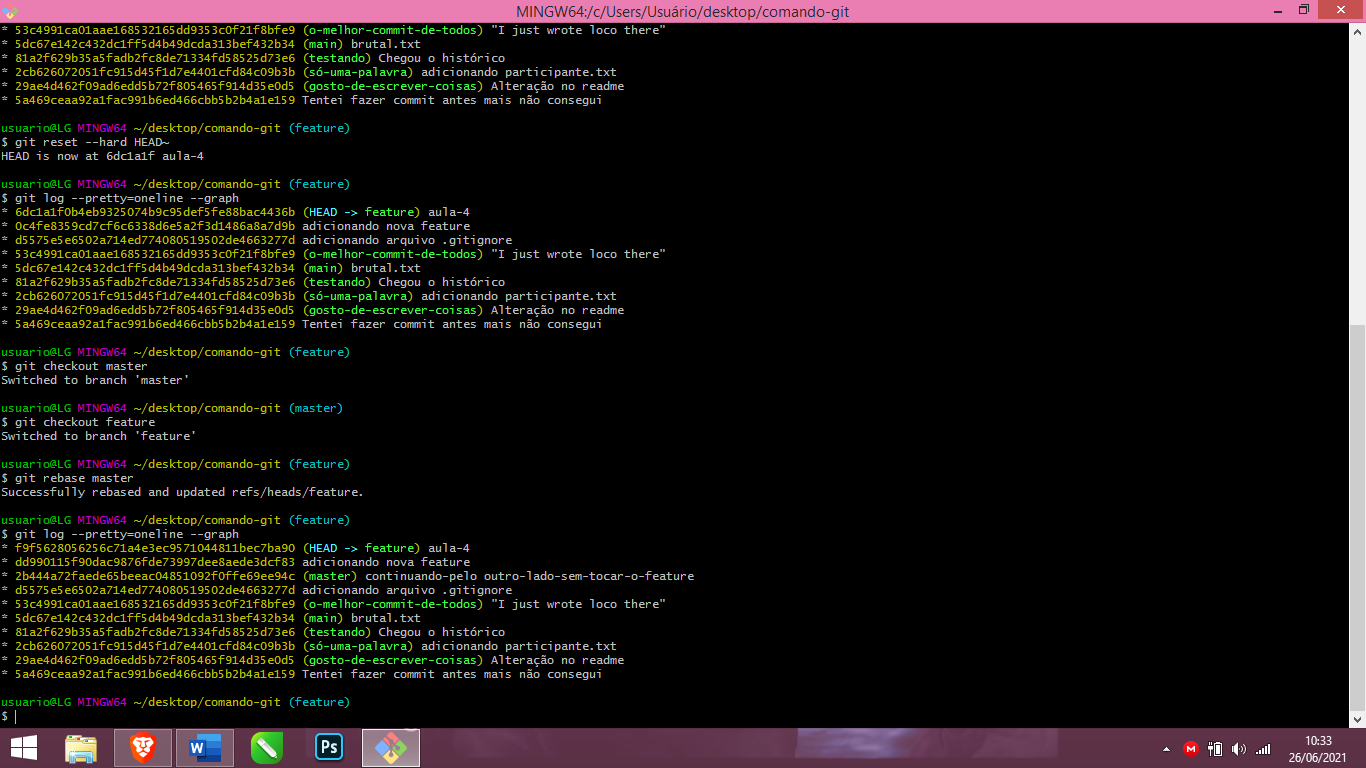
Exemplo gráfico:

 **git log Depois do Merge**

**git log Antes do merge**

Retrocedi este procedimento usando **git reset –hard HEAD~**

Depois, usei Rebase

 **git log depois do rebase**

**git log Antes do rebase**





Neste exercício

**- git branch -f main c6:** Movimenta o Branch “main” a o ‘c6’

**- git checkout HEAD~1 ou git checkout c1:** retrocede ou leva o HEAD para o c1

**- git branch -f bugFix HEAD~1:** Movimenta o bugFix atrás do HEAD

**GITHUB**

O github é uma ferramenta que permite conectar nossos arquivos, commits do git para um repositório no site.

**- git remote add origin [link]**

**- git push -u origin master**

**- git remote:** para ver se o repositório já existe.

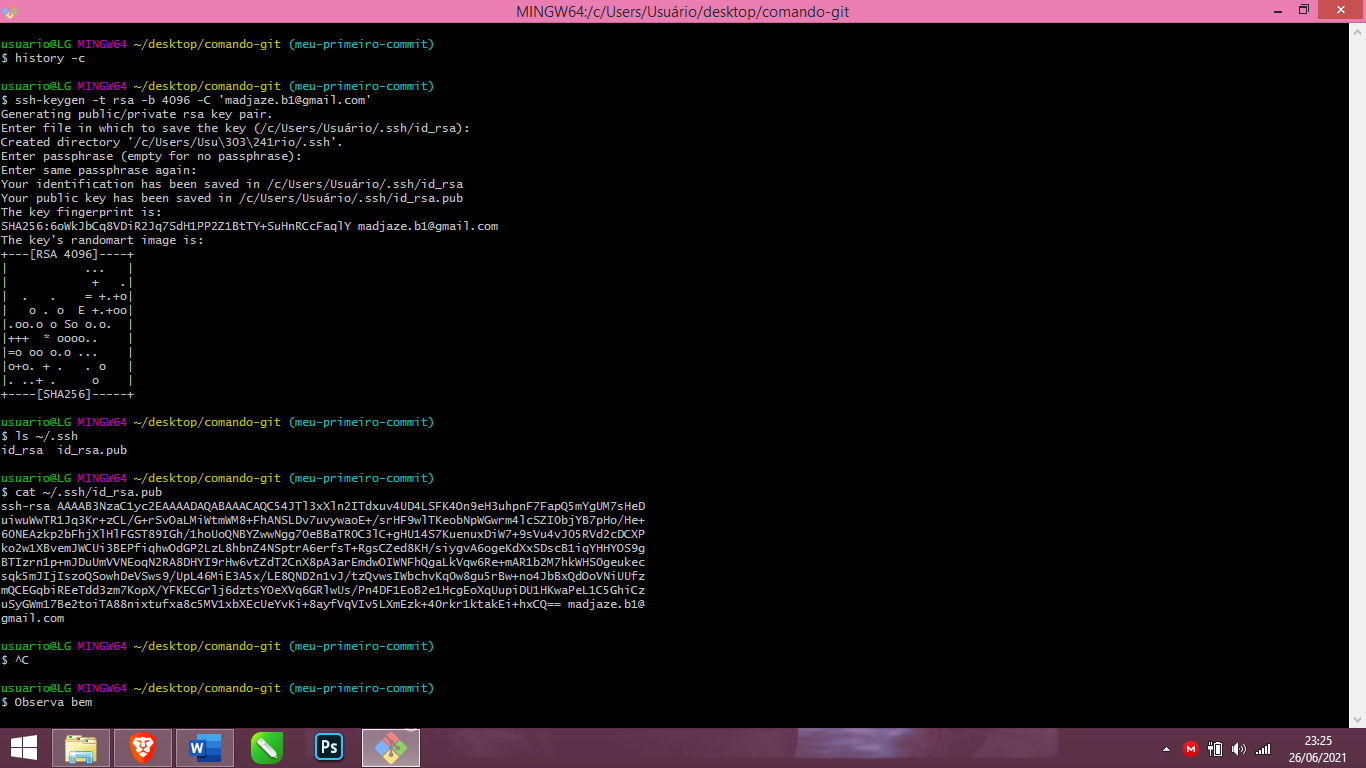
**- git pull origin master:** baixar as atualizações do repositório remoto.

**- git push origin master –tags:** Enviando tags para o repositório remoto.

- Criar uma chave SSH: **ssh-keygen -t rsa -b 4096 -C “[Seu@email.com]”** (O mesmo do GITHUB)

**- ls ~/.ssh**

**- cat ~/.ssh/id\_rsa.pub**



**- git clone [chave ssh]**