Exercício 1 Gitlab

PARTE A

No servidor GITLAB (repositório remoto):

- Crie um projeto chamado **Project**
- Crie o arquivo **changelog.txt** no ramo principal **main**
- Edite o arquivo **changelog.txt** para que contenha esta primeira linha:
  "Versão 1.0: Primeira versão"
- Crie 2 ramificações a partir do ramo **principal** principal: **dev** e **testar**
- Crie 2 ramificações a partir do **ramo main** de desenvolvimento: **dev1** e **dev2**
- Crie e depois exclua um branch **do Dev3**

PARTE B

Na sua máquina:

Abra um terminal (ou console X) e prepare o diretório de funcionamento:

```
$ mkdir ~/TP_GIT
$ CD TP_GIT
$ mkdir DEV1@paris
$ mkdir DEV2@nancy
$ mkdir DEV@metz
$ mkdir TEST@strasbourg
$ mkdir FONCTION1
```

Recupere **manualmente os arquivos sf1a.c**, **sf1b.c**, **sf1.c** do https://wiki.morere.eu/doku.php?id=brasil:courses:git_gitlab:2_git_gitlab_course#useful_files no diretório

Recupere o script profile.sh do https://wiki.morere.eu/doku.php?id=brasil:courses:git_gitlab:2_git_gitlab_course#useful_files no diretório **~/TP_GIT**. Esse script permite que você alterne entre perfis na mesma máquina digitando os comandos dev1, **dev2** ou **dev**.

```
$ mv ~/DOSSIER_TELECHARGEMENT/profile.sh ~/TP_GIT
```

Execute o script **profile.sh**

```
$ CD ~/TP_GIT
profile.sh fonte $
```

Verifique com o **comando de alias** se o script funcionava

Configurando seu GIT:

```
$ git config --global user.email "votre_email"
$ git config --global user.name "votre_pseudo"
$ mais ~/.gitconfig
```

Digite **CTRL-C** para sair do comando more

Aumente o tempo de cache do ID para 7200 (2 horas) segundos, que é definido por padrão para 15 min:

```
git config --global credential.helper "cache --timeout=7200"
```

Crie os repositórios locais para cada ambiente DEV1 e DEV2:

```
$dev 1
```

Estamos em **DEV1@paris**.

```
DEV1@paris$ clone git git@gitlab.com:USER/project.git
$DEV 1@paris projeto CD
```

Verificamos se a origem do alias aponta para o projeto para simplificar a escrita.

```
git remote add origin git@gitlab.com:USER/project.git
```

Uma mensagem indica que o pseudônimo já existe.

```
$ dev2
```

Estamos em **DEV2@nancy**.

```
DEV2@nancy$ clone git git@gitlab.com:USER/project.git
```

Também criamos repositórios para os perfis DEV e TEST, digitamos os mesmos comandos.

PARTE C

COMO DESENVOLVEDOR DEV1, PREPARO A SUBFUNCIONALIDADE 1A, ENVIO PARA MEU BRANCH LOCALMENTE E DEPOIS PARA O REPOSITÓRIO REMOTO PARA O DESENVOLVEDOR USAR:

```
$dev 1
```

Em **DEV1@paris** :

```
$ projeto CD/
$ ls
$ ramo git
$ git branch -a
$ git checkout dev1
$ cp ~/TP_GIT/FONCTION1/sf1a.c . # Preste atenção ao ponto
$ gcc -o sf1a.o -c sf1a.c #on compila a fonte C em um arquivo objeto
$ gcc -o sf1a sf1a.o #on realiza a ligação para gerar o programa executável
$ ./sf1a # rodamos o programa a partir do diretório atual
$ git add sf1a*
$ git commit -m "adicionando sf1a"
$ git push -u
```

PARTE D

COMO DESENVOLVEDOR DO DEV2, PREPARO A SUBFUNCIONALIDADE 1B, ENVIO PARA MEU BRANCH LOCALMENTE E DEPOIS PARA O REPOSITÓRIO REMOTO PARA O DESENVOLVEDOR USAR:

```
$ dev2
```

Em **DEV2@nancy** :

```
$ projeto CD/
$ cp ~/TP_GIT/FONCTION1/sf1b.c . # Preste atenção ao ponto
$ ramo git
$ git checkout dev2
$ gcc -o sf1b.o -c sf1b.c #on compila a fonte C em um arquivo objeto
$ gcc -o sf1b sf1b.o #on realiza a ligação para gerar o programa executável
$ ./sf1b # rodamos o programa a partir do diretório atual
$ git add sf1b*
$ git commit -m "adicionando sf1b"
$ git push -u
```

PARTE E

COMO DESENVOLVEDOR DE DESENVOLVIMENTO, EU JUNTO O TRABALHO DO DEV1 E DO DEV2 E PREPARO UM NOVO SCRIPT DE BUILD ADAPTADO À ARQUITETURA REAL DO PROCESSADOR:

```
$ dev
```

Em **DEV@metz** :

```
$ projeto CD/
$ ramo git
$ git branch -a
$ git checkout dev
$ pit pull
$ ls
$ git merge origem/dev1
$ ls
$ git merge origin/dev2
$ ls
```

O DEV constrói um novo arquivo **sf1.c** concatenando os 2 arquivos **sf1a.c** e **sf1b.c** com um editor e depois o empurra para seu branch localmente e para o repositório remoto para que o TEST possa usá-lo (ou eu baixo o arquivo já concatenado no arch).

NOTA: este arquivo sf1.c já foi criado para simplificar o exercício, recupere-o e coloque-o no diretório do projeto DEV

```
$ gcc -o sf1-build.o -c sf1.c -march=nativo
$ gcc -o sf1-build sf1-build.o
$./sf1-build
$ git add sf1-build
$ git push -u
$ git commit -m "adicionando sf1-build"
$ git push -u
$ pit pull
```

PARTE F

COMO TESTADOR DE TESTE, EU EXECUTO O PROGRAMA **SF1-BUILD** PREPARADO PELO DESENVOLVEDOR PARA VERIFICÁ-LO ANTES DE ENTRAR EM PRODUÇÃO:

```
Teste $
```

Em **TEST@strasbourg** :

```
$ projeto CD/
$ git branch -a
$ teste de verificação git
$ ls
$ pit pull
$ git branch -a
$ git merge origem/dev
$ ls
$./sf1-build
```

PARTE G

COMO TESTADOR DEV1, VERIFICO O HISTÓRICO DE COMMIT DO MEU BRANCH USANDO OS SEGUINTES COMANDOS:

- **git status** (verifique para qual commit o HEAD está apontando)
- **git log --graph --oneline --decorate** (verifique o histórico de commits)
- **git show n°COMMIT** (conhecer os detalhes sobre um commit)
- **git checkout n°COMMIT** (mover HEAD para um commit específico (HEAD destacado))
- **git check dev1** (mova a HEAD para dev1)

```
$dev 1
```

2 SOLUÇÕES:

```
$ alias gl="git log --graph --oneline --decorate" #CRIANDO UM ALIAS PARA GIT LOG
```

OU

```
$ gitk --todos  os registros git #EQUIVALENT com interface gráfica
```

Em **DEV1@paris** :

```
Projeto CD $
$ git status //verifique onde o HEAD aponta
$ gl ou gitk --all ANÁLISE DE LOGS GIT
```

```
$ touch module1.h CRIAR UM NOVO ARQUIVO
$ git add module1.h
$ gl ou gitk --all NOTE SE HOUVER ALGUMA DIFERENÇA EM RELAÇÃO AO GIT LOG ANTERIOR
$ git commit -m "adicionando módulo1"
$ gl ou gitk -all NOTE SE HOUVER DIFERENÇA EM RELAÇÃO AO GIT LOG ANTERIOR
```

```
$ git push -u
$ gl ou gitk --all NOTE SE HOUVER ALGUMA DIFERENÇA EM RELAÇÃO AO GIT LOG ANTERIOR
```

```
$ git show dc591f7 => NOTE AQUI SEU NÚMERO DE COMMIT  VEJA OS DETALHES DE UM COMMIT
$ git checkout dev1 SUBSTITUIR CABEÇA PARA O COMMIT ORIGINAL
```

Crie outro module2.h e verifique se o conteúdo do diretório muda com ls -l

Como exemplo, aqui está uma interpretação do resultado do primeiro **log git** :

1. **cc140e8 (HEAD -> dev1, origin/dev1) adicionou sf1a** :
    - Este é o último commit que você fez no branch dev1.
    - **"HEAD -> dev1"** indica que seu **HEAD**, ou seja, seu ponteiro atual, está no branch **dev1**.
    - **"Origin/Dev1"** significa que esse commit também é o último commit no branch **dev1** do repositório remoto (**Origin**).
2. **dc591f7 (origin/main, origin/test, origin/HEAD, origin/dev2, origin/dev, main, dev) adicionou o changelog** :
    - Este commit é o pai direto do commit cc140e8.
    - Ele é compartilhado por múltiplos ramos tanto no repositório remoto (**origem/principal**, **origem/test**, **origem/dev2**, **origem/dev)** quanto localmente (**principal, dev**).
    - Isso sugere que esse commit é uma semelhança recente entre todos esses ramos.
3. **869a814 Commit inicial** :
    - Este é o primeiro commit do repositório.
    - Todos os ramos começam a partir desse commit.

```
$ etiqueta git -l //LIST TAGS
$ etiqueta git -a v1.0 -m "minha versão 1.0" CRIAR UMA TAG ANOTADA
$ gl //VERIFICAR PARA ADIÇÃO DE TAG
$ git checkout v1.0 VERIFICAR HEAD MOVE
```

Quando executo o comando "**git checkout DEV1",** o sistema mostra esta mensagem:

```
Branch 'dev1' configurado para rastrear branch remoto 'dev1' de 'origin'Mudou para um novo branch
'dev1'
```

O que isso significa?

Quando você executa o **comando git checkout dev1** ou **git switch dev1**, duas coisas acontecem:

1. . Rastreamento de branch remoto: A mensagem **"Branch 'dev1' configurado para rastrear branch remoto 'dev1' a partir de 'origin'"** significa que o Git configurou seu **branch dev1** local para rastrear o **branch dev1** remoto que está no repositório de origem remota. Isso informa o Git que esse branch

local corresponde a um branch no servidor. O rastreamento facilita fazer coisas como **git pull** ou **git push**, porque o git sabe de onde puxar atualizações ou onde enviar mudanças.

2. Mudança de ramo: A mensagem **"Mudou para um novo branch 'dev1'"** indica que o Git mudou seu branch atual para **dev1**. Isso significa que quaisquer commits que você fizer a partir de agora serão adicionados a esse branch até que você troque branch com outro comando **de checkout do git**.

Então, você mudou os branches para funcionar no **dev1**, e esse branch está configurado para seguir seu equivalente no repositório **de origem remota**.

Análise da HEAD no caso de uma fusão :

Em **DEV@metz**

```
$ projeto CD/
$ gl
* fb86245 (HEAD -> dev, origin/DEVdev Merge branch 'dev' de git@gitlab.com:USER/projet em dev
|\
| * f376f57 Excluir sf1b.o
| * 6583fe9 Excluir sf1b.c
| * 0b4b52b Excluir sf1b
| * 9d1db25 Excluir sf1a.o
| * 199a54c Excluir sf1a.c
| * 9c550fc Excluir sf1a
* | 198CD28 Adicionando a Construção SF1
|/
* 87f58ee Muna o branch de rastreamento remoto 'origin/dev2' no dev
|\
| * 25bf076 (origem/DEV2) adicionado sf1b
* | CC140E8 (origin/DEV1) adicionou SF1A
|/
* dc591f7 (origem/principal, origem/test, origem/CABEÇA, principal) adicionou o registro de
alterações
* 869a814 Compromisso inicial
```

RECURSOS ÚTEIS:

Estágio: https://www.freecodecamp.org/news/undo-git-add-how-to-remove-added-files-in-git/#:~:text=There%20are%20two%20major%20commands,git%20rm%20%2D%2Dcached%20commands.
Diário: https://www.ionos.fr/digitalguide/sites-internet/developpement-web/git-log/
Empurrão: https://gdevops.gitlab.io/tuto_git/commands/push/push.html

Exercício 3 Gitlab

PARTE A

1. Crie um repositório git vazio fora do repositório onde você trabalhou até agora, por exemplo, em uma nova subpasta da sua casa.
2. Configure o repositório para ignorar arquivos com a extensão ".o".
3. Crie um arquivo e diga **ao git** que você quer revertê-lo.
4. Mova esse arquivo para outro local e avise **o git**.
5. A saída do **status git** deve falar apenas de dois arquivos no final, aquele que você acabou de mover e o que foi criado na pergunta 2.

Veja o status final do depósito. Ignorando a parte "No ramo principal" por enquanto, você entende as outras mensagens exibidas?

PARTE B

1. Adicione alguns arquivos do conteúdo que você deseja e depois salve um commit contendo esses arquivos.
2. Edite pelo menos dois dos arquivos que você fez backup e depois crie um commit que inclua apenas as alterações em um único arquivo.
3. Veja as mudanças que ainda não foram salvas.
4. Use um atalho git para criar um commit contendo as alterações nos outros arquivos sem precisar nomeá-las nos comandos que você usa.
5. Veja a diferença entre os dois primeiros commits que você fez dentro do repositório.
6. Verifique se o diretório de trabalho está limpo.
7. Mude para o avô do commit atual, anexe uma tag nele, depois volte para o commit original.

PARTE C

1. Crie uma cópia do repositório usado na Parte A.
2. Visite o pai do commit ativo e crie um branch que vá a partir dele.
3. Mude para esse ramo e crie um commit adicionando um novo arquivo.
4. Visualize o branch principal e o que você acabou de criar com uma ferramenta gráfica, por exemplo, código e a extensão git-graph que você vai instalar da seguinte forma: No vscode, use o atalho "CTRL-P" para o quick launch e digite o comando ext install mhutchie.git-graph. Uma vez instalada a extensão, você terá uma visão geral gráfica da evolução dos ramos. (cf. https://marketplace.visualstudio.com/items?itemName=mhutchie.git-graph)
5. Unir as trocas de mãos dentro da filial atual.
6. Substitua o conteúdo de um arquivo por dois textos diferentes dentro do ramo principal e do novo, fazendo essas alterações.
7. Unam as mudanças do novo ramo dentro do principal e resolvam o conflito de fusão que se segue.

PARTE D

1. Envie seu depósito para o Gitlab.