Materiais

- Uma introdução ao Git e Gitflow - TerraLAB
- Terraclub - Hands on Git
- Conventional Commits - Um guia sobre como escrever commits

Questões teóricas

1. O que é Git?
   R: Git é uma ferramenta de trabalho utilizada para organização e sincronismo de arquivos muito utilizada por nós desenvolvedores para desenvolvimento de códigos em equipe.

2. O que é a staging area?
   R: A staging area é um estado de arquivo para arquivos em fase de teste, ou seja, modificados desde o último commit, que estão sendo avaliados quanto à inúmeros fatores, como funcionalidade, otimização, compatibilidade, etc, para serem adicionados em um commit posterior

3. O que é o working directory?
   R: working directory é um diretório, local ou no sistema git na nuvem, que armazena todos os arquivos que estão sendo trabalhados, além de informações importantes, como o status de commit atual, o registro (log) de commits anteriores, e arquivos com funções git importantes, como o reset, ou revert.

4. O que é um commit?
   R: Existem tipos de commit diferentes, existe o commit “upload” e o commit “reverse”, onde o commit update seria para enviar arquivos novos ou modificados para o repositório git e o commit reverse seria para commitar uma versão anterior desejada, ou seja basicamente um “CTRL + Z” no repositório git.

5. O que é uma branch?
   R: Na tradução direta, branch é uma ramificação, e se enquadra bem na utilidade de tal função, uma vez que o branch serve para realizar alterações ou correções no arquivo prévio sem alterá-lo, o que deixa o processo de coding mais seguro e mais organizado, visto que se todos os desenvolvedores alterassem o mesmo arquivo simultaneamente ocorreriam incoerências, mesmo na equipe mais entrosada possível.

6. O que é o head no Git?
   R: Heads são versões de commits, sendo elas atuais ou passadas, e é possível alternar entre elas

7. O que é um merge?

R: Merge é a ação de fundir commits de dois branches diferentes, por exemplo, um branch “staging” com um branch “master” para criar uma nova versão funcional do branch master.

8. Explique os 4 estados de um arquivo no Git.

   R: Untracked é um arquivo novo, que não foi submetido a nenhum commit, e nem à fase de preparação com o comando git add

9. Explique o comando git init.

R: O comando git init em teoria serve para “demarcar” um diretório como sendo um repositório git. Na prática, o comando git init adiciona um diretório no diretório aberto que contém as informações necessárias para o bom funcionamento da ferramenta git.

10. Explique o comando git add.

    R: Git add é o comando que passa arquivos desejados para o estágio “staged”, e os prepara para um commit

11. Explique o comando git status.

R: git status é o comando que basicamente mostra o status de todos os arquivos que ainda não foram “commitados”.

12. Explique o comando git commit.

    R: Git commit é o comando que sincroniza os arquivos preparados previamente para commit, com o git add, no branch desejado.

13. Explique o comando git log.

    R: Git log é o comando que mostra uma lista de todos os commits anteriores, bem como os autores, data e horário.

14. Explique o comando git checkout -b.

    R: O comando git checkout -b cria uma nova branch e movimenta o head para ela

15. Explique o comando git reset e suas três opções.

    R: Existem três opções de git reset, sendo elas soft, mixed e hard.

Hard: retorna totalmente à um commit antigo, alterando todos os arquivos e modificações de arquivos do branch atual, substituindo-os pelos arquivos do commit desejado e sobrescrevendo todas as alterações não salvas atuais. Opção brusca, irreversível

Mixed: retorna parcialmente à um commit antigo, copiando apenas o staging do commit desejado, mantendo o working directory atual e sobrescrevendo a staging area.

Soft: retorna HEAD à um commit antigo, mas não altera o commit atual, possibilitando o retorno total à ele, visto que as alterações realizadas entre o commit antigo e o atual ficam na staging area..

16. Explique o comando git revert.

    R: É um commit que desfaz as alterações feitas entre o commit de escolha e o atual, não impossibilitando o retorno ao atual.

17. Explique o comando git clone.

R: Copia todos os arquivos e diretórios de um repositório git local ou remoto, e cola num destino desejado dentro da máquina atual.

18. Explique o comando git push
    R: Realiza o upload, para o repositório remoto, das alterações realizadas no clone de um repositório remoto na sua máquina.
19. Explique o comando git pull.
    R: é uma função contrária ao git push, que possibilita atualizar o seu código perante alterações realizadas por outros desenvolvedores.
20. Como ignorar o versionamento de arquivos no Git?
    R: criar um arquivo chamado .gitignore no repositorio git e colocar o nome dos arquivos que devem ser ignorados no commit dentro de tal arquivo, podendo ser executado pelo comando linux echo “nome_do_arquivo” >> .gitignore
21. No terralab utilizamos as branches master ou main, develop e staging. Explique o objetivo de cada uma.
    R: A branch master é a versão já testada e que tem melhor funcionamento e estabilidade até a etapa atual de desenvolvimento, a branch develop é a branch mais básica, utilizada para a maior parte do desenvolvimento do programa, desde a construção inicial até a correção de bugs. Após estabilização de várias branches develop diferentes, elas são submetidas ao processo de merge e introduzidas na branch staging, onde serão realizados diversos testes finais para a implementação da versão staging estável e funcional como uma master, e se repete o ciclo.

Questões práticas

1. A essa altura, você já deve ter criado a sua conta do GitLab, não é? Crie um repositório público na sua conta, que vai se chamar Atividade Prática e por fim sincronize esse repositório em sua máquina local.

2. Dentro do seu reposotorio, crie um arquivo chamado README.md e leia o artigo como fazer um readme.md bonitão e deixe esse README.md abaixo bem bonitão: README.md onde o trainne irá continuamente responder as perguntas em formas de commit.
   https://raullesteves.medium.com/github-como-fazer-um-readme-md-bonit%C3%A3o-c85c8f154f8

3. Crie nesse repositório um arquivo que vai se chamar calculadora.js, abra esse arquivo em seu editor de códigos favoritos e adicione o seguinte código:

node calculadora.js a b

```
const args = process.argv.slice(2);
console.log(parseInt(args[0]) + parseInt(args[1]));
```

Descubra o que esse código faz através de pesquisas na internet, também descubra como executar um código em javascript e dado que o nome do nosso arquivo é calculadora.js e você entendeu o que o código faz, escreva abaixo como executar esse código em seu terminal:

RESPOSTA: node “calculadora.js”

EXPLICAÇÃO: a primeira linha declara uma matriz com os argumentos passados ao script node.js, ignora os dois primeiros elementos da matriz (o primeiro é o caminho para o executável do Node.js e o segundo é o caminho para o script), retornando apenas os argumentos relevantes. Finalmente realiza a conversão das strings em inteiros para realizar a operação soma e imprimir na console.

4. Agora que você já tem um código feito e a resposta aqui, você precisa subir isso para seu repositório. Sem usar **git add .** descubra como adicionar apenas um arquivo ao seu histórico de commit e adicione calculadora.js a ele.
   R: git add calculadora.js

   Que tipo de commit esse código deve ter de acordo ao conventional commit.

   R: o commit utilizado para o push do arquivo calculadora.js tem o tipo feat

   Que tipo de commit o seu README.md deve contar de acordo ao conventional commit. Por fim, faça um push desse commit.

   R: o commit do push do arquivo README.md tem o tipo style, ja que apenas customiza esteticamente o repositório.

5. Copie e cole o código abaixo em sua calculadora.js:

```
const soma = () => {
    console.log(parseInt(args[0]) + parseInt(args[1]));
};

const args = process.argv.slice(2);
```

```
soma();
```

Descubra o que essa mudança representa em relação ao conventional commit e faça o devido commit dessa mudança.

R: A mudança representa uma modularização do código, transformando o que era parte do “main” do código em uma função que realiza uma soma, o que pode ser interessante para a organização do código, o que facilita sua leitura e interpretação bem como reduz seu tamanho. O tipo de commit é o refactor.

6. João entrou em seu repositório e o deixou da seguinte maneira:

```
const soma = () => {
    console.log(parseInt(args[0]) + parseInt(args[1]));
};

const sub = () => {
    console.log(parseInt(args[0]) - parseInt(args[1]));
}

const args = process.argv.slice(2);

switch (args[0]) {
    case 'soma':
        soma();
    break;

    case 'sub':
        sub();
    break;

    default:
        console.log('does not support', arg[0]);
}
```

Depois disso, realizou um git add .
e um commit com a mensagem: "Feature: added subtraction"
faça como ele e descubra como executar o seu novo código.

R: A diferença desse código para os anteriores é que este realiza duas funções diferentes de acordo com a escolha do usuário de realizar soma ou subtração, escrevendo, respectivamente, soma ou sub, além disso caso seja escrito algo diferente de soma ou sub, será exibida uma mensagem de erro.

Nesse código, temos um pequeno erro, encontre-o e corrija
para que a soma e divisão funcionem.
R: Erro: o primeiro argumento utilizado tanto na soma quanto na subtração é geralmente onde estaria armazenado a escolha de operação fornecida pelo usuário, o que invalidaria o programa.

Por fim, commit sua mudança.

7. Por causa de joãozinho, você foi obrigado a fazer correções
na sua branch principal! O produto foi pro saco e a empresa perdeu
muito dinheiro porque não conseguiu fazer as suas contas, graças a isso
o seu chefe ficou bem bravo e mandou você dar um jeito disso nunca acontecer.

   Aprenda a criar uma branch, e desenvolva a feature de divisão nessa branch.

8. Agora que a sua divisão está funcionando e você garantiu que
não afetou as outras funções, você está apto a fazer um merge request
Em seu gitlab, descubra como realizá-lo de acordo com o gitflow.

9. João quis se redimir dos pecados e fez uma melhoria em seu código,
mas ainda assim, continuou fazendo um push na master, fez a seguinte alteração no código e fez o commit com a mensagem:

   "feat: add conditional evaluation"

```
var x = args[0];
var y = args[2];
var operator = args[1];

function evaluate(param1, param2, operator) {
  return eval(param1 + operator + param2);
}

if ( console.log( evaluate(x, y, operator) ) ) {}
```

Para piorar a situação, joão não te contou como executar esse novo código,
enquanto você não descobre como executá-lo lendo o código,
e seu chefe não descobriu que tudo está comprometido, faça um revert
através do seu gitlab para que o produto volte ao normal o quanto antes!

10. Descubra como executar esse novo código e que operações ele é capaz de realizar.
Deixe sua resposta aqui, e explique o que essas funções javascript fazem.
R: Este novo código é executado digitando node calculadora.js A B C, onde A e C são números e B é um operador qualquer que é englobado pela linguagem js, exceto mutliplicação, por alguma razão.