**TRABALHO PRÁTICO DE GESTÃO DE VERSÕES**

**Entrega:** 11/06/09/2020 no campus virtual

**Trabalho Individual**

**Objetivo**: Propiciar a familiarização prática dos alunos do curso de graduação em Ciência da Computação e Sistemas de Informação da UFLA com a ferramenta de controle de versões o Git e GitHub (GitLab ou similar).

**O QUE ENTREGAR:** O aluno deverá entregar um relatório provando ter feito a atividade. O relatório deverá conter o enunciado de cada uma das atividades descritas no roteiro (abaixo). Para cada item do roteiro o aluno provará o feitio do trabalho mostrando um ou mais comandos executados e um ou mais *printscreens*. Nestes *printscreens*, deverá ser possível ler o comando executado, o resultado do comando retornado pela ferramenta e a identificação de quem fez o trabalho. No caso de usar console e comandos de linha, todo printscreen deve conter nome do usuário ou informação que possibilite identificar o aluno. Se o aluno utilizar interface gráfica, o aluno deverá mostrar um *printscreen* com o menu aberto e a opção do comando marcada pelo mouse, bem como *printscreen* de outras interfaces que demonstram o resultado da execução do comando.

**Preparação para realizar o trabalho**:

- Assista aos 8 vídeos do curso de GIT disponível em: <https://www.youtube.com/watch?v=WVLhm1AMeYE&list=PLInBAd9OZCzzHBJjLFZzRl6DgUmOeG3H0>

- Ao assistir os vídeos, tome nota dos comandos utilizados e para que eles servem. Faça um pequeno tutorial para você mesmo.

- Assista ao vídeo que faz uma boa introdução sobre o GITHUB: <https://www.youtube.com/watch?v=UMhskLXJuq4>

- Assista ao vídeo que exemplifica merge sendo feito com o Visual Studio Code. <https://www.youtube.com/watch?v=oXMgyQt0ce0>

**Preparação do ambiente para realizar o trabalho:**

1. Instale a ferramenta de Gerencia de Versões GIT do site oficial.
2. A partir do site oficial do GIT, instale uma GUI (Graphic User Interface), interface do usuário, para ter o recurso de interface gráfica além das linhas do prompt de comando. Exemplos de GUIs são TortoiseGit, Git Extensions, SourceTree etc.
3. Abra uma conta no GitHub de estudante, usando o email com domínio da UFLA.

**ENUNCIADO DO ROTEIRO**

Para cada instrução dada abaixo, o aluno deve tirar um ou mais *printscreens* para mostrar o que fez e o resultado.

**Seção criação de projeto e clonagem:**

1. Crie um repositório no GITHUB chamado “Projeto\_<primeironome>\_<segundoNome>”. Se fosse eu, professor, criando o repositório, ele se chamaria “Projeto\_Antonio\_Resende”.
2. Crie duas classes na linguagem que preferir, uma chamada A e a outra B. A classe A tem os atributos int A1 e float A2. A classe B tem os atributos int B1, float B2. O aluno pode alterar o tipo de variável para adequar a declaração a linguagem escolhida (int, integer, Integer,... ou float, real, decimal,....). Crie os gets e sets para esses atributos. A classe A deve ter os métodos MA1() e MA2(). A classe B deve ter os métodos MB1() e MB2(). Dentro de cada método coloque uma instrução de comando que manda imprimir o nome do método. Adicione (upload) dessas duas classes, A e B, no GitHub. **Atenção**, não se esqueça de apresentar o código das duas classes no relatório e a tela mostrando os dois arquivos inseridos no repositório do Github.
3. Faça o clone do repositório Criado no GitHub para sua máquina, isto é, o projeto criado no servidor do GitHub será copiado para sua máquina.

**Seção alteração e atualização do projeto:**

1. Edite a Classe A clonada no seu computador, inserindo o método MA3() com uma linha de comando imprimindo na tela a string ”Alteração a classe A partir do clone”. Crie uma classe C com os atributos String C1 e Int C2, os métodos MC1() e MC2(), cada um deles com um comando que imprime na tela o nome do método.
2. Verifique o Status do repositório explicando a mensagem enviada pelo comando de quais arquivos estão desatualizados, atualizados, sob controle, sem controle etc.
3. Garanta que todas as classes estejam sob controle de versão, adicionando as que não estiverem.
4. Aplique um commit na classe C enviando a mensagem “Criação da Classe C”
5. Aplique um commit na classe A enviando a mensagem “Criado o Método MA3()”
6. Se houver algum arquivo desatualizado faça atualização com commit e descarregue todos os arquivos no servidor.

**Seção de rotulação (tag) de releases do projeto e uso de tags:**

1. Crie uma Tag chamada “release 1.0”. Esta tag deve ser usada para rotular o release da solução contendo a última versão das classes A, B e C.
2. Mostre no GitHub se o rótulo foi criado
3. Altere localmente a Classe B e C, criando, respectivamente, os métodos MB3() e MC3() para aquelas classes. Os métodos deverão ter o comando para imprimir na tela o nome do método.
4. Crie uma classe D com os atributos D1 e D2, e os 2 métodos MD1() e MD2(). As assinaturas dos métodos e tipos de dado são de sua escolha.
5. Atualize o servidor com as alterações feitas, fazendo commit com mensagem adequada para cada classe.
6. Crie uma nova Tag chamada “release 2.0”. Esta tag deve ser usada para rotular o release da solução contendo a última versão das classes A, B, C e D.
7. Configure na sua máquina local que você quer trabalhar com o release 1.0 e faça o clone do projeto pelo release 1.0. Mostre a release que está ativa, quais arquivos fazem parte deste release e o conteúdo deles.

**Seção de resolução de conflitos (merge):**

1. Clone o release 2 na sua máquina local.
2. Abra a classe D e inclua o novo método MD3() nela, imprimindo o nome do método.
3. Salve localmente, SEM atualizar o servidor. (isto simula um usuário que está alterando um arquivo numa máquina)
4. Acesse a classe D da release 2 pelo github (browser). Crie o método MD4(), imprimindo o nome do método. Salve essa alteração no github. (isto é para simular outro usuário alterando o mesmo arquivo em outra máquina)
5. Volte para o terminal ou interface gráfica que você está utilizando para trabalhar no repositório local. Atualize o servidor com as alterações feitas localmente na classe D (foi criado o método MD3() ). Mostre a mensagem retornada ao tentar atualizar o servidor.
6. Faça o merge e verifique se os métodos MD3 e MD4 estão presentes na classe D ou se houve alguma perda de informação.

**Seção de verificar diferença entre releases e arquivos em diferentes versões:**

1. Mostre a diferença entre as duas últimas versões salvas da classe B e D
2. Mostre a diferença entre as releases 1.0 e 2.0.

**Seção de verificar diferença entre releases e arquivos em diferentes versões:**

1. A ferramenta disponibiliza um relatório que mostra todas as alterações já feitas no repositório?
2. A ferramenta disponibiliza um relatório que mostra todas as versões/alterações de um único componente?
3. A ferramenta disponibiliza um relatório contendo as diferenças entre as versões?
4. Quais outros relatórios a ferramenta disponibiliza?
5. A ferramenta disponibiliza uma interface gráfica para gerenciar a árvore de versões?
6. Mostre uma interface gráfica que apresente como está a evolução do seu projeto. Imagem exemplo de uma interface gráfica está abaixo.
7. 