UTFPR **Especialização em Desenvolvimento Web com *Frameworks* Modernos**

**WEB11 — Versionamento de código com Git** **Lista de exercícios 03**
**2022**

**Aluno: Kauan Marques de Moraes Polydoro** **Nota:**

1. Julgue as afirmações a seguir:
(marque V para verdadeiro, F para falso) ( / 10 )

(V) No *detached HEAD state*, não há *branch* atualmente selecionado para *checkout* (o *HEAD* não aponta para nenhum *branch*, mas sim diretamente para um *commit*)
(V) Uma opção para sair do *detached HEAD state* é criar e fazer *checkout* em um *branch*
(F) O *HEAD* não se move automaticamente para o novo *commit* ao realizarmos *commit*
(V) A *tag* não se move automaticamente para o novo *commit* ao realizarmos *commit*
(V) O *branch* atual se move automaticamente para o novo *commit* ao realizarmos *commit*
(V) O comando **checkout** possui uma *flag* que serve de atalho para criar um *branch* e em seguida fazer *checkout* nele
(V) *Branches* são úteis para isolar alterações, ou seja, nos permitem modificar arquivos e reverter essas modificações sem risco de danificarmos o código estável ou atrapalharmos o fluxo de desenvolvimento de outras funcionalidades independentes
(F) Um *merge* automático ocorre quando mesclamos dois *branches* que introduziram alterações diferentes nas mesmas linhas de um mesmo arquivo
(F) O *branch* padrão, isto é, aquele criado no momento do **git init**, é o **def**
(V) As principais diferenças entre *branches* e *tags* são quanto à mobilidade e à necessidade de *flags* especiais ao enviá-los ao repositório remoto

2. Julgue as alternativas a seguir quanto à finalidade de cada comando:
(marque V para verdadeiro, F para falso) ( / 6 )

(V) **branch** cria, exclui e lista “ponteiros” para *commits*
(V) **checkout** restaura uma certa versão (*snapshot*) do repositório
(V) **checkout** em um *branch* faz com que o HEAD aponte para esse *branch*
(F) **fetch** envia os “ponteiros” locais para o *remote*
(F) **merge** inicia uma divergência entre *branches* (apesar de não gerar o primeiro *commit* divergente, que deve ser criado especificamente por **commit**)
(V) **merge** une *branches* divergentes, resultando em um novo *commit* após a resolução de possíveis conflitos

3. Explique brevemente o significado e a utilidade de divergir e unir *branches*, citando exemplos de uso desse recurso. ( / 5 )

Divergir: Cria uma nova branch baseado no ponto atual do repositório, muito útil para que desenvolvedores diferentes possam trabalhar em funcionalidades ou versões diferentes de uma maneira controlada, podendo rastrear as alterações realizadas em cada branch e uni-las futuramente quando finalizadas.

Exemplo de uso: Criar uma nova branch para o desenvolvimento de um novo botão para uma versão futura do sistema.

Unir: Utilizado para unir branches, juntando as modificações realizadas em um único ponto. Útil para unir funcionalidades finalizadas em outras branchs em branchs principais ou novas versões do sistema

Exemplo de uso: Juntar branch de novas features já finalizadas e testadas na branch da nova release, após isso, juntar a branch de release na branch main para o lançamento da nova versão do sistema

4. Explique brevemente o que são *merge conflicts*, em qual comando ocorrem, como detectá-los, como abortá-los e como resolvê-los. ( / 5 )

Merge conflicts ocorrem no momento em que o desenvolvedor utiliza o comando “merge” para a junção de duas branches e existem alterações conflitantes entre elas, ou seja, o mesmo pedaço de código foi alterado em ambas as branches. Nesse cenário, os arquivos conflitantes ficarão marcados com conflict markers e o desenvolvedor deverá tratar o conflito, analisando o código e identificando o que deverá permanecer no código final, realizando o comando git commit para que o git identifique que o conflito foi resolvido.. Caso o desenvolvedor não consiga realizar a tratativa no momento, ele deverá abortar o merge utilizando o comando git merge –abort.

5. Liste os comandos utilizados para remover o *remote* origin2 de um repositório, adicionar o *remote* fictício https://github.com/unavailable/remote.git, enviar as *tags* locais para esse repositório remoto, e então fazer *checkout* no *branch* issue100 (existente apenas no referido *remote*, sem *branch* local até o momento). ( / 5 )

```
git remote remove origin2
git remote add origin https://github.com/unavailable/remote.git
git push –tags
git fetch
git checkout issue100
```

6. Marque as alternativas com **M**, **P** ou **R**, de acordo com as categorias a seguir:
(  / 9 )

**M**: um conflito ocorreu; como proceder com a **m**esclagem?
**P**: o conflito ainda não ocorreu; como **p**revenir que venha a ocorrer?
**R**: conflitos podem ocorrer no dia-a-dia; como **r**eduzir a dificuldade de resolução quando um conflito vier a acontecer?

(R) Manter os *commits* pequenos, limpar o diretório de trabalho antes do *merge*, indicar o objetivo das alterações na mensagem de *commit*, não mexer em segmentos desnecessários
(M) Abster-se de alterar o que não está relacionado ao conflito, testar o código ao término da resolução, tomar o código *theirs* como base para refazer as alterações *ours* sobre ele
(P) Dividir o projeto em partes funcionais independentes, dividir claramente as tarefas na equipe, efetuar *pull* constantemente e estabelecer padrões de código previamente às atividades