Universidade Federal de Ouro Preto
ICEB – Departamento de Computação
Processo Trainee – TerraLab

Estudo Dirigido - Git

Júlia Eduarda Miranda de Sousa 18.1.4084

1 - O que é Git?

O Git é uma ferramenta SCM (controle de código fonte). Tal ferramenta permite rastrear modificações em um repositório, seja ele local ou remoto, permitindo e encorajando o uso de ramificações. Desse modo, a criação, modificação, mescla e deleção de linhas de desenvolvimento podem ser feitas de forma controlada e rápida.

Além disso, o Git é uma ferramenta de código aberto e permite o controle de versão de projetos dos mais simples aos mais complexos.

2 – O que é a Staging Area?

A Staging Area é um local temporário em que as alterações e arquivos a serem adicionados no próximo commit ficam armazenadas no repositório local.

3 – O que é Working Directory?

O Working Directory é composto pelos arquivos atuais que fazem parte do projeto, ou seja, o diretório de trabalho.

4 – O que é Commit?

Um Commit é composto por uma captura das modificações realizadas e do estado do projeto em um determinado momento de sua linha de tempo. Essa captura é semelhante a um snapshot, de modo que é gerada uma versão segura das mudanças realizadas até aquele momento.

5 – O que é uma Branch?

Uma Branch consiste em uma ramificação do código fonte do projeto, estando presente em diversas ferramentas SCM. A ramificação permite criar novos recursos ou correções de bugs de forma encapsulada e independente do código principal, assim facilitando a mescla desse código posteriormente.

6 – O que é Head no Git?

No Git o Head consiste em um ponteiro para uma branch ou um commit, funcionando, de modo geral, como uma referência para a versão atual em que se está trabalhando no momento.

7 – O que é um Merge?

Um merge consiste na mescla de ramificações do código do projeto. Ele combina vários commits em um histórico unificado, mesclando as alterações de duas branches diferentes em uma única.

8 – Explique os quatro estados de um arquivo no Git

No Git os arquivos podem assumir quatro estados diferentes: untracked, unmodified, modified e staged.

Os arquivos do tipo **untracked** (não rastreados) são queles que não constam no último commit, do tipo **unmodified** (não modificados) são aqueles que não foram modificados desde o último commit, do tipo **modified** (modificados) são aqueles que sofreram alterações desde o último commit e **staged** são aqueles que foram adicionados e preparados para serem commitados.

9 – Explique o comando git init

O comando git init é responsável por criar um repositório local git, de modo que cria um subdiretório do tipo. git e todos os metadados necessários.

10 – Explique o comando git add

O comando git add altera o estado de um arquivo para o tipo staged, de modo que ele é preparado para fazer parte do próximo commit.

11 – Explique o comando git status

O comando git status mostra o estado do repositório e da staging area, exibindo as alterações dos arquivos e diferenças entre os arquivos e o commit atual do Head. Além disso, exibe os caminhos da working tree e os estados dos arquivos, se estão sendo rastreados ou não, se há arquivos para serem adicionados à staging area, entre outras informações.

12 – Explique o comando git commit

O comando git commit é utilizado para criar um commit dentro do projeto, de modo que esse commit combina o conteúdo corrente das alterações na working tree assim como seu índice e uma mensagem de log descrevendo as mudanças. O novo commit criado é adicionado à Head, de modo que sua ponta é atualizada para ele.

13 – Explique o comando git log

O comando git log exibe um histórico dos commits do projeto. É possível procurar por commits por meio de seu identificador, ver quais são os contribuintes do projeto, reverter mudanças entre outros.

14 – Explique o comando git checkout -b

O comando git checkout -b permite atualizar a referência da Head para o nome da branch especificada após a flag -b. Desse modo, é trocada a branch em que se está desenvolvendo.

15 – Explique o comando git reset e suas três opções

O comando git reset permite desfazer um commit, apresentando três opções: soft reset, mixed reset e hard reset.

Na opção de **soft reset** o Head é movido para o commit desejado sem alterações na staging area e no working directory.

Na opção de **mixed reset** o Head é movido para o commit desejado gerando alteraçãos apenas na staging area, mantendo o working directory inalterado.

Já na opção de **hard reset** o Head é movido para o commit desejado alterando tanto como a staging area e o working directory, de modo que todas as alterações realizadas após o commit para o qual o Head foi movido são perdidos.

16 – Explique o comando git revert

O comando git revert também permite desfazer commits. Dado um ou mais commits, reverte as mudanças e cria um novo commit para registrar as mudanças. Esse novo commit é utilizado para reverter os efeitos de commits anteriores e, para que funcione corretamente, requer que a working tree esteja limpa.

17 – Explique o comando git clone

O comando git clone clona um repositório git em um novo diretório na máquina, criando rastreamento remoto para cada branch do repositório clonado e faz o check out para a branch inicial. O clone pode ser feito por meio de protocolo HTTPS, por SSH ou pela CLI do Github, por exemplo.

18 – Explique o comando git push

O comando git push atualiza as refs remotas utilizando as refs locais, ou seja, envia as mudanças do repositório local para o repositório remoto. Após adicionarmos as mudanças para o stage e realizar seu commit, o comando push é responsável por fazer o envio dessas mudanças.

19 – Explique o comando git pull

O comando git pull é responsável por incorporar as mudanças do repositório remoto no repositório local. Desse modo, é interessante sempre utilizá-lo no início do desenvolvimento para que a branch atual esteja sempre atualizada e compatível com o repositório remoto, assim evitando conflitos e problemas futuros.

20 – Como ignorar o versionamento de arquivos no Git

Para que arquivos ou diretórios sejam ignorados pelo versionamento de arquivos no Git, é necessária a criação de um arquivo .gitignore. Ao adicionarmos o nome e/ou extensão de arquivos assim como de diretórios o Git passará a ignorar alterações nos itens configurados no arquivo .gitignore.

21 – No TerraLab utilizamos as branches master ou main, develop e staging. Explique o objetivo de cada uma.

A **branch master ou main** é a branch principal da versão, contendo o código mais estável assim como alterações testadas e homologadas. A **branch develop** é criada a partir da branch main, de modo que a partir dela que são desenvolvidas todas as melhorias e evoluções de funcionalidades ou bug fixes.

Já a branch **staging** é criada a partir da branch develop e é utilizada para colocar em teste as modificações geradas pelos desenvolvedores. Após a aprovação das modificações, que ocorre após uma rotina de testes, a versão da branch staging é mesclada à branch main. Após esse merge temos uma nova versão do software.

Referencial Bibliográfico

http://www2.decom.ufop.br/terralab/uma-introducao-ao-git-e-gitflow/

https://git-scm.com/about

https://www.atlassian.com/br/git/tutorials/what-is-git

https://www.devmedia.com.br/como-usar-os-comandos-do-git/33665

https://www.atlassian.com/br/git/tutorials/saving-changes/git-commit

https://git-scm.com/docs/git-status

https://git-scm.com/docs/git-log

https://git-scm.com/docs/git-checkout

https://git-scm.com/docs/git-revert

https://git-scm.com/docs/git-push