**95. Introdução ao Git e Github**

O Git é um sistema de versionamento que permite controlar as versões de um projeto ou de um dado documento. Imagine, por exemplo, você tendo que controlar as N versões do seu TCC da faculdade.

Cada versão é controlada pelo que nós chamamos de Commit e cada um deles possui um número identificador único. Através dessa identificação que é possível controlar as versões, voltar a uma versão anterior, comparar versões etc.

Já o GitHub é um sistema online para gerenciamento de repositórios GIT. Em outras palavras é também conhecido como uma rede social para programadores, onde você consegue disponibilizar repositórios remotos para que sejam clonados localmente e assim trabalhados em uma equipe, por exemplo.

Dessa forma podemos dizer que existem dois tipos de repositórios, os locais e os remotos. Os remotos são os que estão centralizados em serviços online como no Github, Bitbucket, etc. E os locais é o que você usa para trabalhar em sua máquina local.

Diagrama

Descrição gerada automaticamenteQuando não temos um projeto, o primeiro passo é cloná-lo para isso usamos o comando **git clone.** Após fazer nossas alterações fazemos um **commit** para salvar a versão em nosso repositório local. Por fim, enviaremos as modificações para o servidor através do **git push**. Para que todo esse mecanismo funcione é necessário ter o GIT instalado em sua máquina e uma conta no GitHub.

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

**99. Configurando sua identificação**

A identificação é composta pelo nome de usuário e o e-mail do serviço usado, no caso o que foi cadastrado no GitHub. Para definir isso via comando, basta abrir um git bash e informar os comandos:

$ **git config –global user.name “Nathan Cirillo”**

$ **git config –global user.email “nathan\_cirillo@hotmail.com”**

Para verificar se a identificação ficou correta use:

$ **git config –list**

**102. Salvando a primeira versão de um projeto no Github**

O primeiro passo é criar um repositório no Github, ou seja, o local para onde os seus arquivos ficarão hospedados.

Depois iremos criar localmente a pasta com o projeto ou os arquivos que queremos versionar em nosso computador. Devemos informar que essa pasta será um repositório usando o comando:

**$ git init**

Para salvar um versão sempre precisaremos adicionar o arquivo a uma área especial chamada de Stage e depois realizar o Commit. Para isso usamos os seguintes comandos:

**$ git add .**

**$ git commit -m “Meu commit inicial”**

Sempre que possível busque verificar qual é o status do seu repositório local como, por exemplo, conferir se está atualizado. Para isso:

**$ git status**

Para conseguir enviar os arquivos para o Github é necessário associar o repositório local ao remoto. Essa associação é feita com o comando abaixo:

**$ remote add origin git@github.com:seuusuario/seurepositorio.git**

Por fim para enviar os arquivo de fato use:

**$ git push -u origin master**

**103. Salvando um novo commit**

Toda e qualquer modificação será rastreada. Então conforme citado sempre verifique o status (git status), adicione os arquivos em Stage (git add .), faça o Commit (git commit -m “mensagem”) com uma mensagem explicativa e por fim envie os arquivos/modificações (git push -u origin master).

**104. DEMO – clonar e modificar um projeto**

Partindo do pressuposto que os seus projetos já se encontram atualizados no repositório remoto, basta você navegar até o local onde deseja criar/baixar esse repositório em seu sistema. Informe o seguinte código:

**$ git clone <endereço\_repositório>**

Após o projeto ser clonado podemos fazer modificações e trabalhá-las conforme instruções do tópico 103. Outro comando interessante é a possibilidade de ver os commits realizados em um projeto. Para isso use o comando:

**$ git log**

**105. Git log para verificar histórico de versões**

Além do comando visto anteriormente **git log** que permite ver o histórico de versões (commits) de forma detalhada, pode-se também usar a diretiva **git log –oneline**, assim será uma forma resumida:

**$ git log –oneline**

Esse commando torna-se útil em projetos grandes que já possuem uma grande quantidade de commits. Dessa forma iremos ver apenas o seu identificador e a mensagem, facilitando a visualização.

**106. Entendendo o git status e stage**

Ao dar um git status existem três situações possíveis para os arquivos do projeto: modified, untracked e deleted. Modificado é qualquer arquivo que já está sendo rastreado e por algum motivo foi alterado. Não rastreado são os arquivos novos que ainda não estão sendo rastreados, ou seja, não foram adicionados e comitados. E por fim, o deleted é quando temos um arquivo rastreado e deletamos o mesmo. Toda e qualquer modificação deve ser adicionada a área de Stage para que possamos fazer o commit.

Texto

Descrição gerada automaticamente com confiança média

A área de Stage é importante pois podemos controlar os arquivos que queremos adicionar a cada versão. Por exemplo ,ao modificar 10 arquivos, 5 podem estar em uma commit e os outros 5 em outro commit.

Um outro comando bastante interessante é o que permite tirar os arquivos da área de Stage, ou seja, voltam os arquivos para o estado original. Para isso usamos:

$ **git reset**

**107. Git diff e recurso source control do VS Code**

O **git diff** é o comando responsável por mostrar a diferença entre arquivos modificados. A dica é usar o VS Code, pois mostra as diferenças graficamente.

**108. Git checkout**

Modifica os arquivos do projeto ao estado de um dado Commit ou Branch. Como vimos cada commit possui um código usado para referenciá-lo. O último commit do Branch corrente pode ser chamado pela palavra HEAD.

Para voltar ao último commit realizado, ou seja, ao seu estado original podemos usar o comando:

$ **git checkout master**

Dessa forma conseguimos navegar e testar as versões de um projeto existente.

Essa trata-se de uma modificação não necessária.