**Curso GIT Completo - do Básico ao Avançado**

**>> 26/01/2022 <<**

**Cap. 3 - Controle de Versão:**

Os softwares de Controle de Versão permitem o gerenciamento das versões, com documentação, podendo ser qualquer tipo de arquivo (.xml, .xls, .docx, .css, .txt, etc.).

* Muito difundido em empresas de software;
* Pode ser usado em pequenos ou grandes projetos;
* Usado para controlar as etapas de qualquer tipo de trabalho feito em um computador (trabalhos acadêmicos, textos diversos do Word, entre outros diversos tipos de arquivos).

**Funcionalidades:**

* Histórico de versões;
* Alternar entre versões;
* Backup – a versão anterior fica armazenada;
* Trabalho em Equipe;
* Desenvolvimento em paralelo em ambientes diferentes;
* Ramificação – podem-se criar diversas versões a partir de um ponto;
* Junção das funcionalidades após finalizadas;
* Rastreabilidade – identifica quais foram as mudanças efetuadas e quem as efetuou.

**Cap. 4 – Tipos de Controle de Versão:**

**Centralizado:**

* O histórico das alterações fica centralizadas em um único servidor;
* O usuário dever ter acesso a esse servidor para acessar o histórico;
* Cada versão deve ser baixada individualmente;
* Quando uma versão está sendo utilizada por um usuário, outros não poderão acessá-la.

**Distribuído:**

* Distribuído nas máquinas que estão operando no repositório;
* Também possui servidor;
* Todo histórico do projeto é trazido para a máquina local;
* Melhor performance de programação;
* Réplica de repositórios em cada máquina, assim, caso haja algum problema com o servidor, o projeto pode ser novamente obtido com os usuários envolvidos.

**Características do GIT:**

* Operações locais;
* Navegação pelo histórico;
* Criação de Branch (ramificação) - troca de versões sem que seja necessário comunicar o servidor para tal, o que dá maior eficiência ao programador;
* Somente adição de conteúdo ao histórico (não há remoção de itens do histórico);
* Integridade – um arquivo, uma vez rastreado, será rastreável permanentemente, até ser removido do projeto, e todas as alterações ficarão no histórico;
* Autonomia – várias pessoas realizam alterações sem dependência de arquivos (porém, quando os usuários forem integrar o projeto, devem verificar se há conflitos).

**Cap. 5 – Instalação do GIT:** <https://git-scm.com> e seguir com a instalação padrão.

**Cap. 8 – Configuração do Usuário e alguns Comandos:**

* git --version: Retorna a versão do GIT instalada;
* git config --global user.name “nome\_do\_usuario”: Configura o nome do usuário;
* git config --global user.email “e-mail\_do\_usuario”: Configura o e-mail do usuário;
* git --list: Lista as configurações do GIT.

**Cap. 9 – Criação de um Repositório Local:**

* dir: Lista o que tem na paste;
* git init: Inicializa um repositório vazio;
* “master”: É a Branch (ramificação) principal do projeto – a partir dessa branch, poderão ser feitas novas ramificações;
* dir -a: Lista os arquivos e pastas ocultos;
* A pasta “.git” armazena todas as informações do repositório (todo o histórico);
* Comando “cd ..”: volta à pasta acima da atual.

**Cap. 10 – Versionamento - Introdução:**

* git status: Retorna o estado do arquivo;
* git add nome\_do\_arquivo: Faz o rastreamento do arquivo;
* git add --all ou git add . ou git add -A: Adiciona todos os arquivos da pasta atual.

**Cap. 11 – Versionamento – Salvando Alterações:**

* git commit -m “Mensagem para o Commit”: Salva uma nova versão do banco de dados local do GIT.

**Cap. 12 – Funcionamento do GIT:**

**Working Directory (working tree):**

* Criado a partir do “git init” – é a pasta onde estamos trabalhando com o projeto;
* Arquivos Modificados;
* Arquivos Excluídos;
* Arquivos Adicionados.

**Staging Area:**

* Área de preparação;
* Staged – adicionado com “git add .”;
* Contém os arquivos preparados para serem versionados.

**Committed:**

* Arquivos salvos (versionados) e prontos para fazer upload.

**Cap. 13 – Visualizar Alterações sem abrir o projeto:**

* git diff: Retorna as informações sobre as alterações no diretório;
* git diff –cached ou git diff --staged: Retorna as diferenças na área de preparação (cache).

**Cap. 14 – Histórico de Alterações:**

* git log: Lista os commits feitos no projeto;
* git log --oneline: Lista os commits de forma resumida;
* CHA: Algoritmo que gera um número de associação com o commit, como, por exemplo: 23d199865904a9123043123f0de753754dea1acc;
* HEAD: Aponta a última modificação em uma Branch;
* Para mais variações de log, verificar o arquivo “Git+log+-+Variações.pdf”.

**Cap. 15 – Usando Commits Anteriores:**

* git checkout numero\_do\_CHA: Aponta o HEAD para o commit referente ao CHA e volta ao estado do projeto do commit informado;
* Não é necessário informar o número completo do CHA;
* Importante para verificação de alterações e cada commit do projeto;
* git checkout master: Aponta o HEAD para a última versão da Master.

**Cap. 16 – Desfazendo Alterações:**

* git checkout nome\_do\_arquivo: Desfaz as alterações feitas no arquivo informado. Também recupera o arquivo informado, caso tenha sido excluído e se queira recuperá-lo (antes de um novo commit);
* git reset --hard: Desfaz todas as alterações de todos os arquivos do projeto, desde que não tenha sido feito um novo commit.

**Obs.:** O “git checkout” não funciona em arquivos novos.

**Cap. 17 – Desfazendo Alterações Não Rastreadas:**

* git clean -f: Força a remoção de arquivos não rastreados com “git add”, caso haja, pois o “git reset” não funcionar é nesse caso.

**Cap. 18 – Ignorando Arquivos (arquivo “.gitignore”):**

Há momentos em que não queremos versionar arquivos;

* Por exemplo, arquivos de configuração da máquina local; ou
* Arquivos de senha; ou
* Arquivos .NET (.dll), etc..

**Criando um Arquivo .gitignore:**

* Na pasta do projeto, crie um arquivo .txt vazio e com qualquer nome;
* Entre no CMD e vá até a pasta do projeto;
* Renomeio o arquivo com: ren nome\_do\_arquivo.txt .gitignore (para Mac e Linux, utilizamos o comando “mv” ao invés do “ren”;
* Dentro desse arquivo, inserir informações sobre o que o Git deve ignorar quando for fazer os versionamentos. Por exemplo: “teste.bmp” (ignora esse arquivo), ou “NomeDaPasta/\*.bmp” (ignora todos os arquivos .bmp da pasta “NomeDaPasta”.

**Obs.:** O arquivo “.gitignore” poderá ser criado quando for criado o repositório no GitHub.

**Quais Arquivos Ignorar?**

* No site do <https://github.com/github/gitignore>, podemos verificar os arquivos .gitignore para todos os tipos de projetos, para todos os tipos de plataformas;
* Copiar e colar para o arquivo local .gitignore, conforme necessário.

**Cap. 19 – Clonando Repositórios:**

* git clone nome\_da\_pasta/ novo\_nome\_de\_pasta: Clona a pasta de um projeto do computador local, cria uma nova pasta com o nome desejado e insere essa clonagem nessa nova pasta;
* git clone <https://github.com/DevMasterTeam/GitCompleto.git>: Clona um repositório do GitHub (a partir da URL informada);

**Obs.:** O repositório <https://github.com/DevMasterTeam/GitCompleto.git> contém Templates Bootstrap para baixar gratuitamente.

**Cap. 20 – Introdução ao GitHub:**

* Introdução