SEGUNDO AULÃO DESIGN PARA WEB

Professora: Joseneuza Aguiar

Aluno: Lucas Barbosa

CONTROLE DE VERSÃO

Introdução 📖:

Controle de versão é um tema da Engenharia de Software, disciplina que irá compor futuramente o currículo de vocês presumindo que estão no primeiro semestre.

> **Engenharia de software** é uma área da **engenharia** e da **computação** voltada à **especificação, desenvolvimento, manutenção e criação de software**, com a aplicação de **tecnologias** e **práticas** de **gerência de projetos** e outras disciplinas, visando **organização**, **produtividade** e **qualidade**. Atualmente, essas tecnologias e práticas englobam **linguagens de programação, banco de dados, ferramentas, plataformas, bibliotecas, padrões de projeto de software, processo de software e qualidade de software**. Além disso, a engenharia de software deve oferecer **mecanismos** para se **planejar** e **gerenciar** o **processo de desenvolvimento de um sistema computacional de qualidade** e que **atenda às necessidades de um requisitante de software**.

Uma “pista” do motivo de controle de versão ser um tópico em Engenharia de Software:

A eficácia do controle de versão de software é comprovada por fazer parte das exigências para **melhorias do processo de desenvolvimento** de certificações tais como **CMMI** e **SPICE (ISO 15504)**.

O que é controle de versão?

Um **sistema de controle de versões** (ou versionamento), **VCS** (do inglês version control system) ou ainda **SCM** (do inglês source code management) na função prática da Ciência da Computação e da Engenharia de Software, é um **software que tem a finalidade de gerenciar diferentes versões no desenvolvimento de um documento qualquer**. Esses sistemas são comumente utilizados no desenvolvimento de software para **controlar as diferentes versões** — **histórico e desenvolvimento** — dos **códigos-fontes e também da documentação**.

Esse tipo de sistema é muito presente em empresas, **instituições de tecnologia e instituições de desenvolvimento de software**. É também muito comum no

desenvolvimento de software livre. É útil, em diversos aspectos, tanto para projetos pessoais pequenos e simples como também para grandes projetos comerciais.

Entre os mais comuns encontram-se as soluções livres: **CVS, Mercurial, Git e SVN** (Subversion); e as comerciais: **SourceSafe, TFS, PVCS** (Serena) **e ClearCase**. O **desenvolvimento de software** livre utiliza mais o **Git** (com repositórios no GitHub), que vem substituindo o SVN (Subversion), que por sua vez é um sucessor do CVS (Concurrent Versions System).

Mais sobre controle de versão:

O **desenvolvimento de software** envolve um **processo contínuo de evolução de código**. Baseado nesse paradigma, surgiu a necessidade do desenvolvimento de uma solução que **gerenciasse o controle de versões dos códigos fonte**, **da documentação** e **do compartilhamento de trabalho**. Assim, surgiram os softwares responsáveis de controle de versão.

Importância de controlar versões:

1. Possibilitar compartilhamento de código fonte e desenvolvimento distribuído
2. Permitir controle de modificações e trabalho em paralelo
3. Auxiliar na qualidade do código e em sua manutenção
4. Atrair novos colaboradores para projetos abertos e expor sua evolução

Controle de versão em outras áreas:

1. Histórico de modificações em documentos de texto (wikis)
2. Configurações de um sistema computacional (snapshots ou profiles)
3. Gerenciamento de configurações de software (SCM)
4. Alteração de desenhos feitos em CAD
5. Protocolos de comunicação
6. Interfaces de programação (APIs)

Algumas perguntas que aparecem com frequência com relação à VCS:

1. Já perdeu alguma versão anterior do código do projeto?
2. Tem problemas em manter diferentes versões do sistema ao mesmo tempo?
3. Alguém da equipe já sobrescreveu o código de outra pessoa por acidente e acabou perdendo as alterações?
4. Tem dificuldades em saber quais as alterações efetuadas em um programa, quando foram feitas e quem fez?

Porque usar um VCS:

1. Mantém e disponibiliza cada versão já produzida de cada item do projeto.
2. Possui mecanismos para gerenciar diferentes ramos de desenvolvimento, possibilitando a existência de diferentes versões ao mesmo tempo.
3. Possibilidade de se ter um histórico do desenvolvimento, bem como a possibilidade de rastrear as alterações feitas durante um projeto.
4. Oportunidade de possibilitar a divisão de tarefas de forma que não comprometa a integridade total do projeto. Desse modo, pode-se dividir uma equipe de programadores sem que nenhum atrapalhe o projeto do outro, tudo isso sem problemas de sincronização de arquivos.
5. A utilização dos Sistemas de Controle de Versão facilita a restauração de versões funcionais. Desse modo, é possível trabalhar com segurança o desenvolvimento de uma versão de software sem o comprometimento da versão estável.

GIT

O Git é o único VCS que existe?? 🤔:

V·T·E	Version control software	[hide]
Years, where available, indicate the date of first stable release. Systems with names *in italics* are no longer maintained or have planned end-of-life dates.		
Local only	Free/open-source	RCS (1982) · SCCS (1973)
	Proprietary	PVCS (1985) · *QVCS* (1991)
Client–server	Free/open-source	CVS (1986, 1990 in C) · CVSNT (1998) · QVCS Enterprise (1998) · Subversion (2000)
	Proprietary	AccuRev SCM (2002) · Azure DevOps (Server (via TFVC) (2005) · Services (via TFVC) (2014)) · ClearCase (1992) · *CMVC* (1994) · Dimensions CM (1980s) · *DSEE* (1984) · Endevor (1980s) · Integrity (2001) · Panvalet (1970s) · Perforce Helix (1995) · SCLM (1980s?) · Software Change Manager (1970s) · StarTeam (1995) · Surround SCM (2002) · Synergy (1990) · Team Concert (2008) · Vault (2003) · *Visual SourceSafe* (1994)
Distributed	Free/open-source	*ArX* (2003) · BitKeeper (2000) · Breezy (2017) · *Code Co-op* (1997) · Darcs (2002) · *DCVS* (2002) · Fossil (2007) · Git (2005) · *GNU arch* (2001) · GNU Bazaar (2005) · Mercurial (2005) · Monotone (2003)
	Proprietary	Azure DevOps (Server (via Git) (2013) · Services (via Git) (2014)) · *TeamWare* (1992) · Plastic SCM (2006)

Fonte: https://en.wikipedia.org/wiki/Version_control

O que é o GIT? Historia 📖 e funcionamento ⚙:

Primeiramente, para explorar com maior profundidade toda a capacidade do software Git, é altamente recomendável a leitura do e-book gratuito disponibilizado no site.

https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control

E-book gratuito dentro do próprio Git ensinando como usá-lo (Exercitem o Inglês!!)

Breve história do GIT

Como muitas coisas na vida, o Git começou com um pouco de destruição criativa e uma ardente controvérsia.

O núcleo (kernel) do Linux é um projeto de código aberto com um escopo bastante grande. A maior parte da vida da manutenção do núcleo o Linux (1991-2002), as mudanças no código eram compartilhadas como correções e arquivos. Em 2002, o projeto do núcleo do Linux começou usar uma DVCS proprietária chamada BitKeeper.

Em 2005, a relação entre a comunidade que desenvolveu o núcleo do Linux e a empresa que desenvolveu BitKeeper quebrou em pedaços, e a ferramenta passou a ser paga. Isto alertou a comunidade que desenvolvia o Linux (e especialmente Linux Torvalds, o criador do Linux) a desenvolver a sua própria ferramenta baseada em lições aprendidas ao usar o BitKeeper. Algumas metas do novo sistema era os seguintes:

1. Velocidade
2. Projeto simples
3. Forte suporte para desenvolvimento não-linear (milhares de ramos paralelos)
4. Completamente distribuído
5. Capaz de lidar com projetos grandes como o núcleo o Linux com eficiência (velocidade e tamanho dos dados)

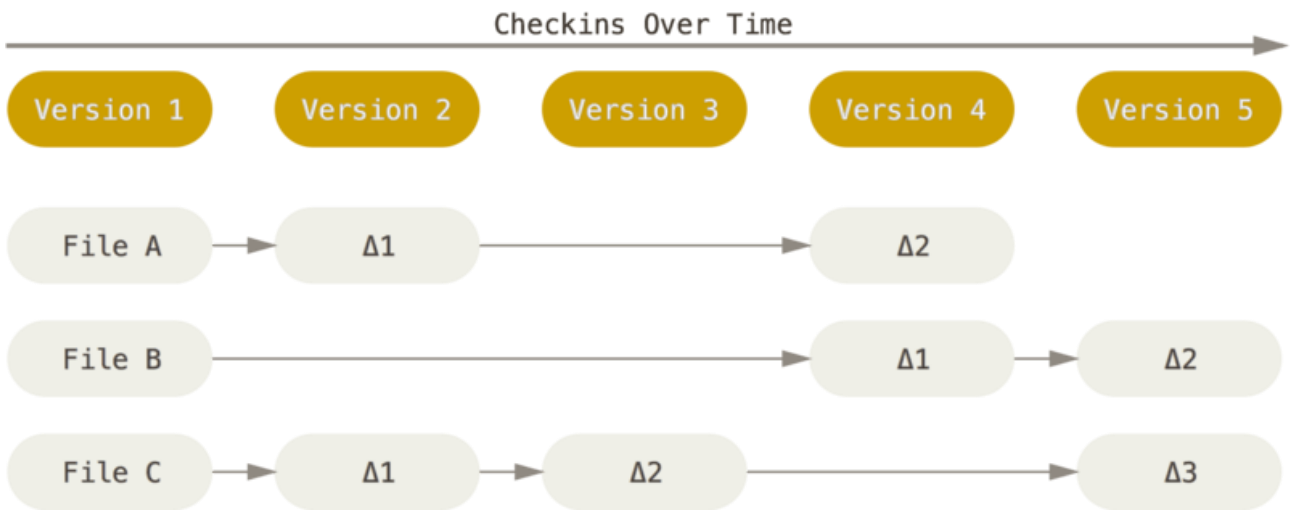
Desde seu nascimento em 2005, Git evoluiu e amadureceu para ser fácil de usar e ainda reter essas qualidades iniciais.

Trecho retirado do site **git-scm** (https://git-scm.com/book/pt-br/v2/Come%C3%A7ando-Uma-Breve-Hist%C3%B3ria-do-Git)

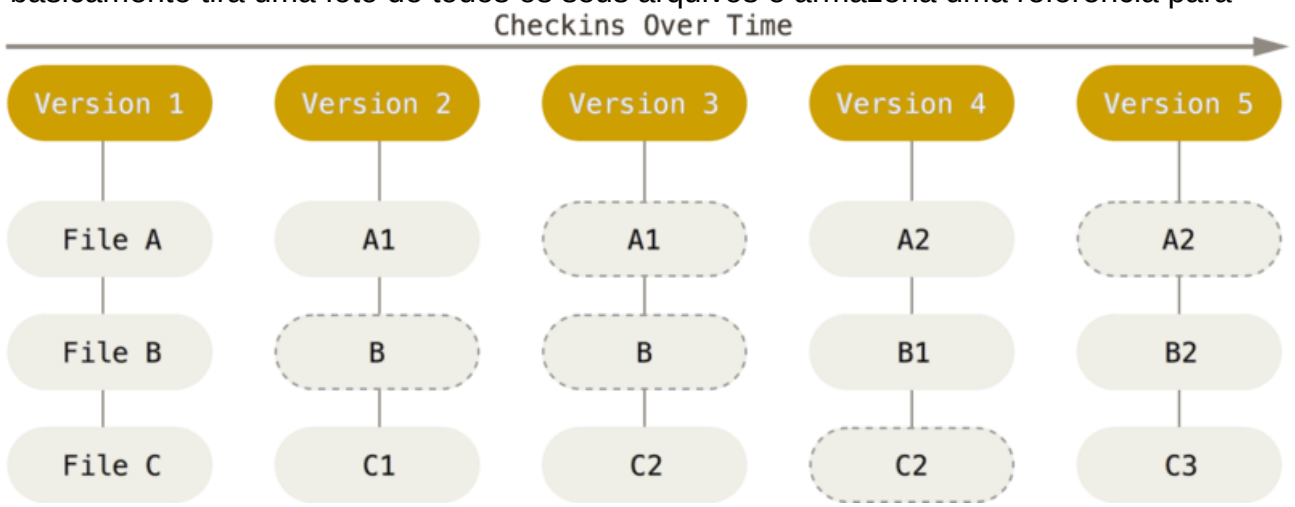
Funcionamento básico do GIT

Imagens, Não Diferenças

A principal diferença entre o Git e qualquer outro VCS (Subversion e similares) é a maneira como o Git trata seus dados. Conceitualmente, a maioria dos outros sistemas armazenam informação como uma lista de mudanças nos arquivos. Estes sistemas (CVS, Subversion, Perforce, Bazaar, e assim por diante) tratam a informação como um conjunto de arquivos e as mudanças feitas em cada arquivo ao longo do tempo.



O Git não trata nem armazena seus dados desta forma. Em vez disso, o Git trata seus dados mais como um conjunto de imagens de um sistema de arquivos em miniatura. Toda vez que você fizer um commit, ou salvar o estado de seu projeto no Git, ele basicamente tira uma foto de todos os seus arquivos e armazena uma referência para



esse conjunto de arquivos. Para ser eficiente, se os arquivos não foram alterados, o Git não armazena o arquivo novamente, apenas um link para o arquivo idêntico anterior já armazenado. O Git trata seus dados mais como um fluxo do estado dos arquivos.

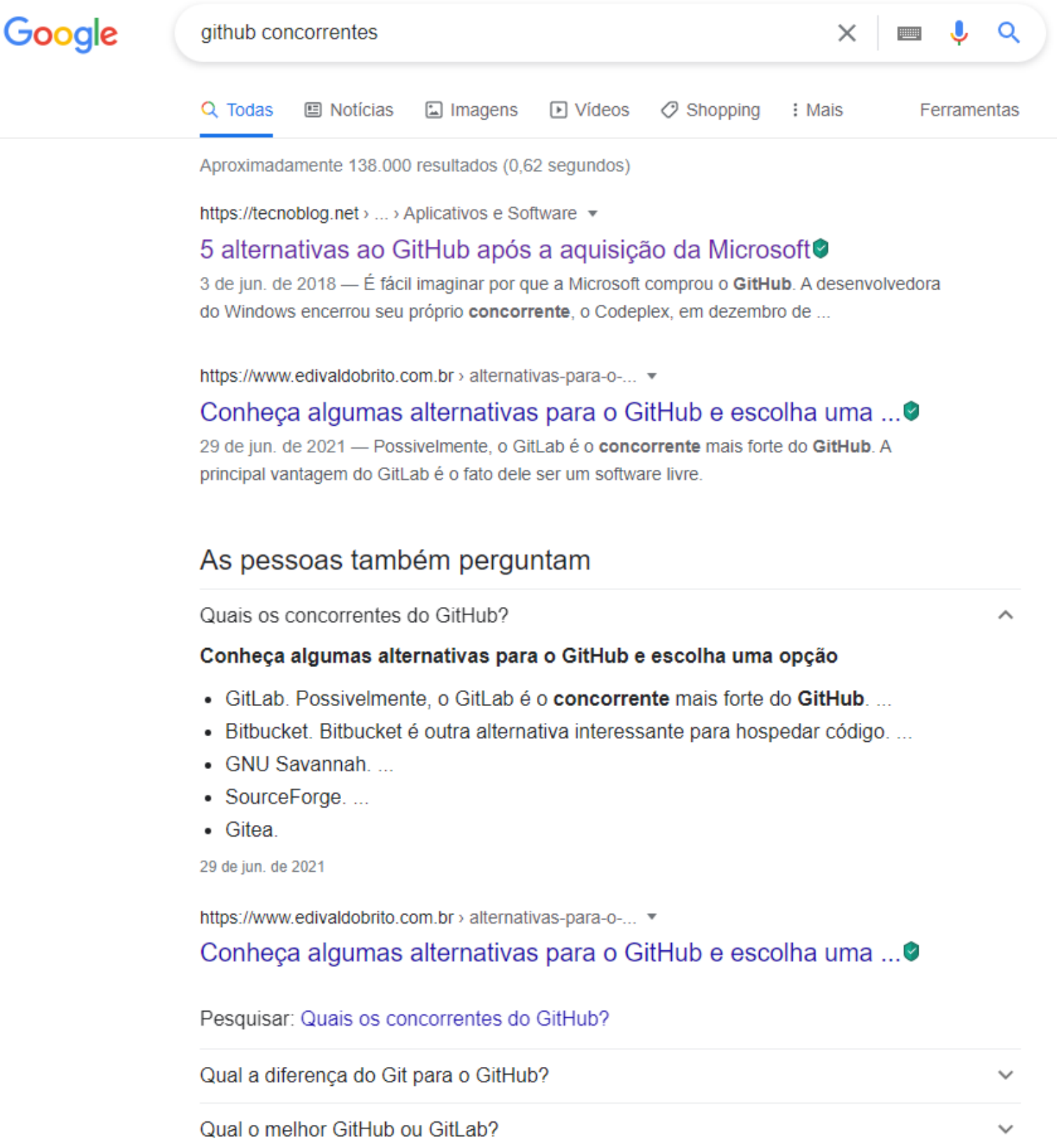
Esta é uma diferença importante entre o Git e quase todos os outros VCSs. Isto faz o Git reconsiderar quase todos os aspectos de controle de versão que a maioria dos outros sistemas copiaram da geração anterior. Isso faz com que o Git seja mais como um mini sistema de arquivos com algumas ferramentas incrivelmente poderosas, ao invés de simplesmente um VCS.

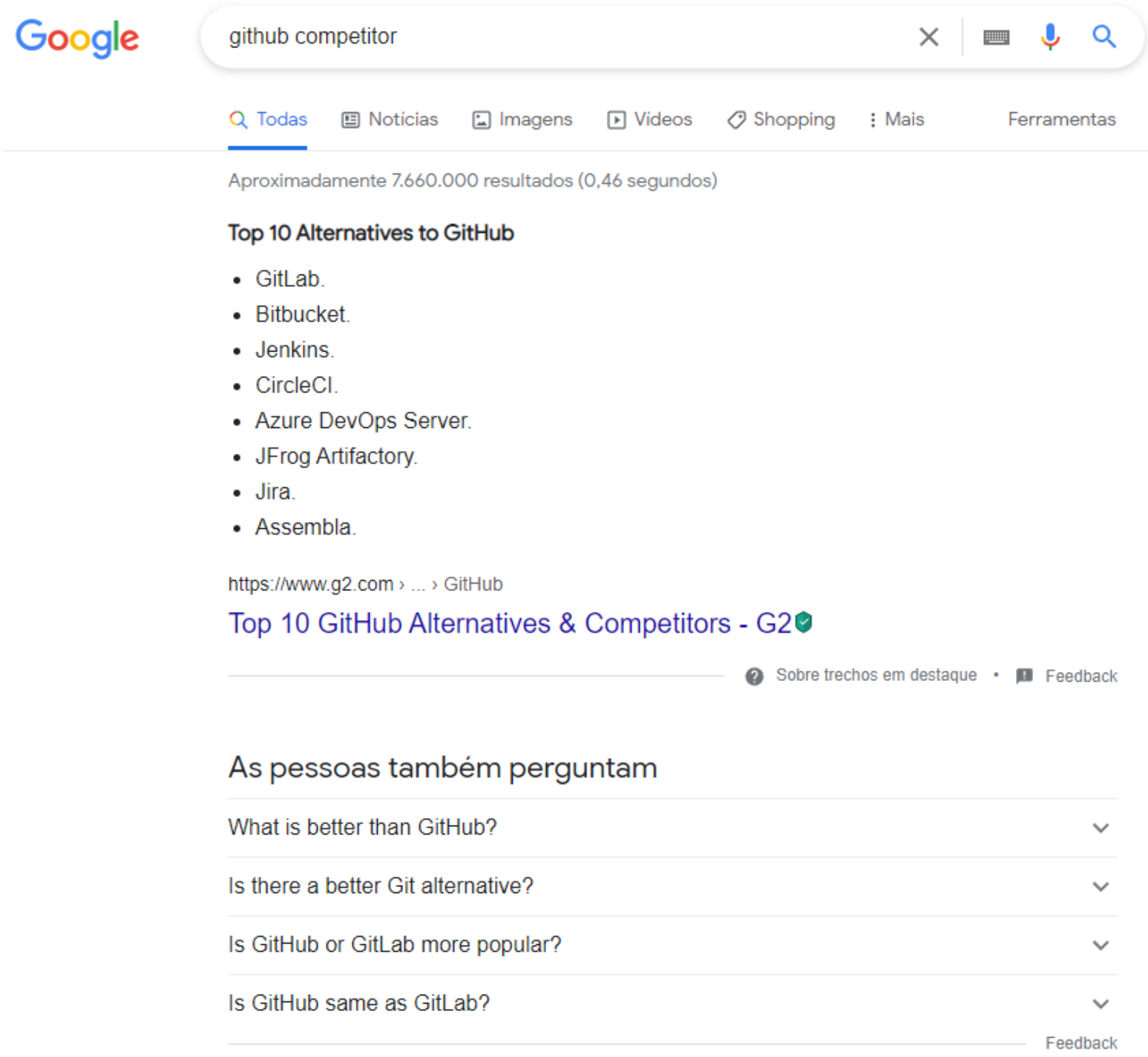
Trecho retirado do site **git-scm** (https://git-scm.com/book/pt-br/v2/Come%C3%A7ando-O-B%C3%A1sico-do-Git#:~:text=O%20fluxo%20de%20trabalho%20b%C3%A1sico,para%20o%20diret%C3%B3rio%20do%20Git.)

GITHUB

O GitHub é a única plataforma de repositório remoto que existe??

🤔:





O que é esse tal de GitHub? 🤔:

GitHub é uma espécie de "rede social para programadores". O site tem uma fama especial nesse nicho, sendo também um serviço de publicação e compartilhamento de códigos de programação. Lançada em 2008, a plataforma é usada mundialmente e é, desde 2018, de propriedade da Microsoft.

Segundo o site oficial, o GitHub já conta com números superiores a 65 milhões de desenvolvedores, 3 milhões de organizações cadastradas e 200 milhões de repositórios. Na lista a seguir, veja perguntas e respostas sobre o que é o GitHub, para que serve e como usar o site.

O slogan do GitHub é "Social Code Host" (hospedeiro social de códigos, em tradução livre). A base do site é justamente essa: armazenar códigos de programação, produzidos por desenvolvedores do mundo todo, e compartilhá-los como se fosse uma rede social. Dessa forma, é possível que quaisquer usuários cadastrados na plataforma divulguem seus trabalhos e que outros membros da comunidade façam contribuições.

Com isso, o GitHub também pode funcionar como um serviço de colaboração de projetos pessoais e até comerciais. Grandes empresas como Google e WordPress também usufruem do site no que diz respeito às possibilidades de suporte a problemas e até novos desenvolvimentos para suas plataformas.

Trecho retirado do site TechTudo (https://www.techtudo.com.br/listas/2021/05/o-que-e-o-github-veja-para-que-serve-a-rede-social-de-programadores.ghtml)

ROTEIRO 📖

Durante a aula 😳:

1- Criando e alterando um repositório local (Git):

1. Criar pasta qualquer com um nome desejado
2. Abrir o GIT Bash
3. Executar os seguintes comandos
    a) git init – Começar um repositório local
    b) git status – Checar o Status do repositório
    c) git log – Checar os Logs do repositório
    d) echo “Escreva aqui qualquer mensagem” >> arquivoNovo.txt
    e) git status
    f) git log
4. Dentro da pasta, criar um arquivo LEIAME.md e escrever qualquer conteúdo dentro dele, sendo que seu primeiro caracter deve ser um #(Exemplo: #Esta é uma mensagem)
5. Voltar ao GIT Bash
6. Executar os seguintes comandos
    a) git status
    b) git log
    c) git add . – Comando responsável por selecionar quais arquivos serão versionados no repositório
    d) git commit -m “Escreva aqui a sua mensagem” – Comando responsável por alterar a versão de um repositório por meio da manipulação dos arquivos contidos neste

2- Criando branches em um repositório local (Git):

1. Estar na pasta do exercício 1
2. Estar com o GIT Bash aberto
3. Executar os seguintes comandos
    a) git branch – Listar branches
    b) git status – Checar o Status do repositório
    c) git log – Checar os Logs do repositório
    d) git branch ramificacao1 – Criar nova branch
    e) git checkout ramificacao1 – Ir até a branch
    f) Crie algum arquivo ou tradicionalmente ou com o comando “echo”
    g) git add .
    h) git commit -m “Escreva aqui qualquer mensagem”
4. Pronto, você criou uma branch e ela tem arquivos diferentes da branch principal!

3- Criando e alterando um repositório remoto (GitHub):

1. Criar uma conta no GitHub, se ainda não tem.
2. Ir para a página inicial
3. Clicar em “Repositories”
4. Clicar em “New”

5. Definir as configurações do repositório: Nome do repositório (Repository name), Descrição (Description), Visibilidade (Deixar “Public” selecionado), Inicializar este repositório com um arquivo README file(Marcar a opção), Escolher a licença MIT ou não escolher (Choose a license)
6. OBS: É altamente recomendável ler mais sobre as licenças (MIT License, Apache License 2.0, GNU General Public License v3.0,...)
7. Ir até o repositório criado.
8. Clicar em “Add File”
9. Clicar em “Create new file”
10. Nomear o arquivo como “arquivoCriadoNoGitHub.txt” e escrever qualquer conteúdo nele.
11. Ir até “Commit new file” e escrever em “Create new file” qualquer descrição.
12. Clicar em “Commit new file”

4- Forkando e clonando um repositório remoto (GitHub):

1. Ir até o repositório https://github.com/ldsbarbosa/AulaoGitGitHub.git
2. Clicar na opção “Fork”
3. Preencher o que for necessário
4. Ir até seu repositório
5. Clicar no repositório replicado
6. Clicar na opção “Code”
7. Copiar o link na opção HTTPS
8. Criar uma pasta qualquer com um nome desejado para hospedar o repositório
9. Abrir o GIT Bash
10. Executar os seguintes comandos
    a. git clone link.git – O link deve corresponder ao link descrito na opção HTTPS
    b. git log
11. O repositório está no seu computador!

5- Trabalhando com o clone na máquina local (Git):

1. Uma vez com o repositório clonado na máquina, você pode executar os mesmos comandos e trabalhar no mesmo ritmo das atividades 1 e 2

Exceção 1: Removendo nome e e-mail do Git local:

https://www.codegrepper.com/code-examples/shell/remove+git+user.name

https://stackoverflow.com/questions/6243407/delete-username-from-a-git-repository

Exceção 2: Removendo credenciais do GitHub no Git local (Cuidado com push em máquinas públicas!!):

https://stackoverflow.com/questions/44246876/how-to-remove-cached-credentials-from-git

https://stackoverflow.com/questions/15381198/remove-credentials-from-git

Após a aula 🏠:

Primeiro Dia - Entendendo GIT | (não é um tutorial):

https://www.youtube.com/watch?v=6Czd1Yetaac&list=PLUYfChmF6AM9U4pqfCaZ-baDO65EsJLpf&index=1

Segundo Dia (parte 1) - O QUE É GIT E GITHUB? - definição e conceitos importantes 1/2:

https://www.youtube.com/watch?v=DqTITcMq68k&list=PLUYfChmF6AM9U4pqfCaZ-baDO65EsJLpf&index=2

Segundo Dia (parte 2) - COMO USAR GIT E GITHUB NA PRÁTICA! - desde o primeiro commit até o pull request! 2/2:

https://www.youtube.com/watch?v=UBAX-13g8OM&list=PLUYfChmF6AM9U4pqfCaZ-baDO65EsJLpf&index=3

https://github.com/rafaballerini/GitTutorial (Link do repositório complementar à aula)

Terceiro Dia - Curso de Git e Github COMPLETO 2021 [Iniciantes] + Desafios + Muita Prática:

https://www.youtube.com/watch?v=kB5e-gTAl_s&list=PLUYfChmF6AM9U4pqfCaZ-baDO65EsJLpf&index=4

Quarto Dia - Curso de Git e GitHub: grátis, prático e sem usar comandos no terminal:

https://www.youtube.com/playlist?list=PLHz_AreHm4dm7ZULPAmadvNhH6vk9oNZA

Quinto Dia em frente – Continuem estudando!

Existem várias playlists, tanto em inglês quanto em português, falando tanto sobre Git (Git Bash e outras ferramentas), quanto GitHub (Expondo código via repositório na plataforma e interagindo na plataforma com Issues. Forks e Pull Requests).

Também existem materiais escritos sobre vários sistemas de controle de versão e sobre várias plataformas de repositórios remotos, o que inclui Git e GitHub

.

COMANDOS IMPORTANTES (GIT)

Comandos básicos:

git --version:

Checa a versão do software Git na sua máquina.

git log:

Lista os commits que podem ser acessados seguindo os links pai do(s) commit(s) fornecido(s), porém não inclui os commits que podem ser acessados daqueles fornecidos com um ^ na frente deles. A saída é dada em ordem cronológica inversa por padrão.

git reflog:

Os logs de referência, ou "reflogs", registram quando as branches e outras referências foram atualizadas no repositório local.

Reflogs são úteis em vários comandos do Git, para especificar o valor antigo de uma referência. Por exemplo, HEAD@{2} significa "onde HEAD costumava estar há duas posições", master@{one.week.ago} significa "onde a branch “master” costumava a estar há uma semana atrás neste repositório local" e assim por diante.

git reset –hard *idFornecidoPeloGitReflog*:

Acessar todas as versões do repositório, incluindo acessar as passadas e voltar para a atual.

git log nomeDa*branch1*...nomeDa*branch2*:

Mostra os commits em uma “branchA" que não estão em uma “branchB”.

git log --follow *nomeDoArquivo*:

Mostra os commits que alteraram o arquivo, mesmo que sejam somente renomeações.

Comandos essênciais para versionamento local:

git init:

Inicializa um novo repositório Git. Se você deseja colocar um projeto sob controle de revisão, este é o primeiro comando que você precisa aprender.

git add:

Este comando atualiza o índice usando o conteúdo atual encontrado na árvore de trabalho, para preparar o conteúdo preparado para o próximo commit. Ele normalmente adiciona o conteúdo atual dos caminhos existentes como um todo, mas com algumas opções também pode ser usado para adicionar conteúdo com apenas parte das alterações feitas nos arquivos da árvore de trabalho aplicadas.

git commit:

Crie um novo commit contendo o conteúdo atual do índice (git add e git status) e a mensagem de log fornecida descrevendo as alterações.

git branch:

Vê a branch atual em que está.

git branch *qualquerNome*:

Cria uma branch com o nome desejado.

git checkout *nomeDeUmaBranch*:

Navega até uma branch existente, seja ela a principal(main ou master) ou uma branch alternativa dedicada para desenvolvimento de uma funcionalidade, correção de um erro, etc.

git checkout -b *nomeDesejadoParaBranch nomeDeUmaBranchBase*:

Cria uma branch e já entra nesta mesma branch tendo como base uma certa branch já existente, por exemplo as branchs master/main.

git merge *nomeDaBranchASerIncorporada*:

Incorpora as alterações dos commits nomeados (desde o momento em que seus históricos divergiram da ramificação atual) na ramificação atual.

echo "Qualquer mensagem que desejar escrever" >> *qualquerArquivo*:

Cria um arquivo textual com a extensão desejada (podendo ser .txt ou .md, preferencialmente .txt) já inserindo dentro dele a mensagem desejada.

Exemplo: echo "Este é meu primeiro arquivo." >> arquivoNovo.txt

cat *qualquerArquivo*:

Lê o conteúdo de arquivo textual com a extensão desejada (podendo ser .txt ou .md, preferencialmente .txt).

Exemplo: cat arquivoNovo.txt

Output: Este é meu primeiro arquivo.

Comandos essênciais para versionamento remoto:

git remote:

Gerencia o conjunto de repositórios remotos cujas ramificações você rastreia a partir do software de versionamento local. Se executado sem argumentos (git remote), ele listará os repositórios remotos os quais você aponta.

Para adicionar um novo repositório remoto, use o comando ***git remote add nomeDesejado link.git*** no terminal do diretório no qual seu repositório está armazenado.

O comando git remote add usa dois argumentos: Um nome para o repositório remoto (por exemplo: origin) e uma URL remota (por exemplo https://github.com/user/repo.git)

git push:

Atualiza referências remotas usando referências locais, enquanto envia objetos necessários para completar as referências fornecidas.

git pull:

Incorpora alterações de um repositório remoto na ramificação atual. Se a ramificação atual estiver atrás do controle remoto, por padrão, ele avançará rapidamente a ramificação atual para corresponder ao controle remoto.

git clone:

Clona um repositório em um diretório recém-criado, cria ramificações de rastreamento remoto para cada ramificação no repositório clonado (visível usando git branch --remotes) e cria e verifica uma ramificação inicial que é bifurcada da ramificação atualmente ativa do repositório clonado.

touch .gitignore:

Cria um arquivo *.gitignore*.

O Git vê cada arquivo em sua cópia de trabalho em um de três estados:

rastreado – um arquivo que já passou pelo staging ou commit;

não rastreado – um arquivo que não passou pelo staging ou commit; ou

ignorado – um arquivo que o Git foi forçado a ignorar.

Os arquivos ignorados costumam ser artefatos de desenvolvimento e arquivos gerados por máquina que podem ser derivados da fonte do seu repositório ou que não devem passar por commit. Exemplos comuns incluem:

caches de dependência, como o conteúdo de /node_modules ou /packages

código compilado, como arquivos .o, .pyc e .class

diretórios de saída de build, como /bin, /out ou /target

arquivos gerados no período de execução, como .log, .lock ou .tmp

arquivos de sistema ocultos, como .DS_Store ou Thumbs.db

arquivos pessoais de configuração do IDE, como .idea/workspace.xml

Os arquivos ignorados são rastreados em um arquivo especial chamado *.gitignore*, que é verificado na origem do seu repositório.

Trecho retirado do site *Atlassian* (https://www.atlassian.com/br/git/tutorials/saving-changes/gitignore)

CRÉDITOS/FONTES

Controle de versão:

Material disponibilizado pelo prof. Fábio Lúcio Lopes de Mendonça na disciplina de Engenharia de Software ministrada remotamente no segundo semestre do ano de 2021

https://pt.wikipedia.org/wiki/Engenharia_de_software

https://pt.wikipedia.org/wiki/Sistema_de_controle_de_vers%C3%B5es

Git:

https://git-scm.com/

GitHub:

https://www.techtudo.com.br/listas/2021/05/o-que-e-o-github-veja-para-que-serve-a-rede-social-de-programadores.ghtml

https://github.com/

Comandos Git:

https://education.github.com/git-cheat-sheet-education.pdf

https://www.atlassian.com/git/glossary

Vídeos apresentados no Roteiro

LINKS ÚTEIS

Git: https://git-scm.com/

GitHub Desktop: https://desktop.github.com/

Source Tree: https://www.sourcetreeapp.com/

Armazenar suas credenciais do GitHub no Git: https://docs.github.com/pt/get-started/getting-started-with-git/caching-your-github-credentials-in-git