**CURSO TÉCNICO EM DESENVOLVIMENTO DE SISTEMAS**

**VERSIONAMENTO**

**TURMA:** T04\_ARIOT\_23

**ETAPA ONLINE:** 21/11/2023 a 30/11/2023

**ETAPA PRESENCIAL:** 06/12/2023 CFP 5.01 CAMPINAS

**Sumário**

[1 INTRODUÇÃO 3](#_Toc179026478)

[1.1 OBJETIVOS: 3](#_Toc179026479)

[2 CONCEITOS 3](#_Toc179026480)

[2.1 git E github: versionamento e colaboração 3](#_Toc179026481)

[2.2 VERSIONAMENTO 4](#_Toc179026482)

[2.3 REPOSITÓRIO 4](#_Toc179026483)

[2.4 COMMIT 4](#_Toc179026484)

[2.5 BRANCH 4](#_Toc179026485)

[2.6 merging 4](#_Toc179026486)

[2.7 FORK 4](#_Toc179026487)

[2.8 PULL REQUEST 5](#_Toc179026488)

[3 instalação do git 5](#_Toc179026489)

[3.1 Validação da instalação 5](#_Toc179026490)

[3.2 Repositório Local 5](#_Toc179026491)

[3.1 CONFIGURAÇÕES INICIAIS DO REPOSITÓRIO 7](#_Toc179026492)

[3.2 Executando o 1º commit 9](#_Toc179026493)

[3.2.1 Comando git.add 9](#_Toc179026494)

[3.3 Efetuando alterações no repositório 9](#_Toc179026495)

[4 TREINAMENTO DE VERSIONAMENTO 11](#_Toc179026496)

[4.1 CONCEITOS & DEFIINIÇÕES 11](#_Toc179026497)

[4.1.1 Ramificação (Branch) e Mesclagem (Merge) 11](#_Toc179026498)

[4.2 SSITEMA DE CONTROLE DE VERSIONAMENTO 11](#_Toc179026499)

[4.2.1 DISTRIBUTED VERSION CONTROL SYSTEMS (DVCS) 3](#_Toc179026500)

[4.2.2 SNAPSHOTS – DIFERENÇAS ENTRE GIT x Outros VCS 1](#_Toc179026501)

[4.2.3 ATIVIDADES: 1](#_Toc179026502)

[5 REFERÊNCIAS 1](#_Toc179026503)

# INTRODUÇÃO

## OBJETIVOS:

Bem-vindo ao curso do ProEducador - Versionamento de Código!

Em um mundo cada vez mais voltado para a tecnologia, a gestão eficaz do código fonte se tornou um elemento crucial no desenvolvimento de software. Por isso, o curso de versionamento com Git e GitHub, será uma peça fundamental para o sucesso de suas disciplinas de programação no curso de desenvolvimento de software.

O versionamento é a espinha dorsal do desenvolvimento de software. Com o Git, você poderá acompanhar as mudanças em seu código, colaborar de forma eficiente com colegas de equipe, controlar versões anteriores e futuras de seus projetos e garantir a estabilidade e integridade do código-fonte. Tudo isso é essencial para o sucesso nas disciplinas de programação e, consequentemente, no desenvolvimento de software como um todo.

Este curso é constituído por duas etapas:

1 - Módulo on-line assíncrono autoinstrucional, com carga horária de 8h, onde aprenderemos o básico sobre Git e GitHub.

2 - Módulo on-line síncrono para abordarmos temas mais avançados e tirarmos dúvidas gerais.

# CONCEITOS

## git E github: versionamento e colaboração

O Git é um sistema de controle de versões utilizado pelos desenvolvedores de software para registrar as mudanças do desenvolvimento ao longo do tempo, além de permitir o trabalho colaborativo entre equipes de trabalho.

Foi criado em 2005 por Linus Torvalds e tornou-se uma ferramenta essencial para as equipes de desenvolvimento de software.

## VERSIONAMENTO

Através do versionamento podemos manter um histórico sobre as mudanças sofridas pelo código, assim, é possível recuperarmos versões anteriores ou comparar com as versões atuais.

O Git opera de forma distribuída, ou seja, cada integrante da equipe de desenvolvimento possui uma cópia completa do projeto em sua máquina local, permitindo trabalho offline. Assim, a colaboração é mais flexível e evita perda de dados.

## REPOSITÓRIO

O repositório Git é o local onde os arquivos, os metadados e o histórico do projeto ficam armazenados. O repositório existe localmente e remotamente.

Para armazenamento remoto utilizamos o GitHub, GitLab e BitBucket.

## COMMIT

O commit representa um ponto de alteração no projeto.

As alterações efetuadas nos arquivos são armazenadas nesse ponto.

Os commits ficam associados a uma mensagem que descreve as mudanças a partir daquele ponto em relação ao conteúdo anterior.

## BRANCH

Quando criamos um repositório com o comando “git init” é criada uma branch principal chamada “main”.

A brach main possui a linha de base do código e é considerada a versão mais estável do projeto.

Além da branch main, podemos criar branches adicionais que são ramificações que possuem a cópia completa da branch principal.

As alterações feitas em uma branch não afeta a branch principal, permitindo que as equipes trabalhem em novos recursos ou correções de forma independente.

Fonte: https://blog.betrybe.com/git/git-branch/

## merging

* O merging é responsável pela mesclagem do conteúdo inserido nas branches para a branch principal. É neste momento que o trabalho de todos é unificado para obtermos o projeto final.

## FORK

* O fork permite que projetos hospedados em uma plataforma, como o GitHub, sejam copiados para outra pessoa, permitindo que alterações sejam feitas sem afetar o projeto original.
* Posteriormente, as alterações feitas na cópia poderão ser atualizadas no repositório original através de uma requisição (pull request) ao proprietário.

## PULL REQUEST

* O pull request é uma requisição de aceite de alterações efetuadas em um repositório que foi “forcado”. O dono do repositório original pode aceitar a requisição de modo a efetivar as mudanças sugeridas.

# instalação do git

A utilização do git depende de instalação em nosso computador. O download do git pode ser feito no link abaixo:

[Download do git](https://git-scm.com/downloads)

Escolha o download de acordo com a versão do seu Sistema Operacional.

<https://git-scm.com/downloads>

## Validação da instalação

Para verificarmos se o git foi instalado corretamente vamos executar o seguinte comando:

git version

O resultado do comando acima deverá se parecer com a imagem ao lado.

Após a instalação do git, teremos à disposição todos os comandos necessários para gerenciar nosso repositório local, além dos comandos para sincronização com um repositório remoto.

Os comandos git serão inseridos a partir do terminal de comandos do seu sistema operacional (SO).

## Repositório Local

A utilização do git depende da criação de um repositório local, ou seja, é necessário configurarmos uma pasta em nosso computador como sendo o repositório do nosso projeto. Esse repositório poderá, mais tarde, ser sincronizado com um repositório remoto.

Para praticarmos a utilização do git, vamos criar uma pasta em nosso sistema de arquivos que será a pasta raiz do nosso projeto.

Portanto, crie uma pasta com o nome “Projeto\_Git”, em uma pasta da sua preferência.

Crie os seguintes arquivos na pasta “Projeto\_Git” que você acabou de criar, conforme indicado na figura abaixo:

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

## CONFIGURAÇÕES INICIAIS DO REPOSITÓRIO

Antes de começarmos as configurações do repositório local, vamos verificar qual é o status inicial.

Através do comando “git status” podemos ver algumas informações úteis sobre o repositório, tais como nome da branch, arquivos não rastreados, dentre outras informações.

Digite o comando abaixo no terminal para vermos o status do nosso repositório:

* “git status” podemos ver algumas informações úteis sobre o repositório, tais como nome da branch, arquivos não rastreados, dentre outras informações.
* Interface gráfica do usuário, Texto, Aplicativo

  Descrição gerada automaticamente

Dependendo da versão do git que você instalou, o nome da branch principal será “master”. Veremos como mudar o nome da branch mais adiante.

Antes de continuarmos é importante efetuarmos algumas configurações em nosso repositório, como mudar o nome da branch principal e fornecer os dados do usuário que fará os commits.

Para mudar o nome da branch, digite o comando abaixo:

* git branch -M main

Outra mudança importante, é em relação ao nome e e-mail do usuário que fará os commits. Quando trabalhamos em equipe precisamos saber quem são os responsáveis por “subir” alguma alteração. Para isso, execute os comandos abaixo substituindo o nome e e-mail pelos seus dados:

* git config –global user.name “André Shiguemitsu Aoki”
* git config –global user.email “andre.aoki@sp.senai.br

Verificar as configurações do repositório

git config -l

Texto

Descrição gerada automaticamente

## Executando o 1º commit

1º) Preparação do repositório local – ok

2º) adicionar os arquivos que queremos registrar as alterações ao próximo “commit”

### Comando git.add

git add index.html

ou adicionar todos os objetos do repositório

git add .

Quando executamos o comando “git add” estamos enviando os arquivos para uma área chamada de “staging area”, agora devemos confirmar o registro das alterações realizando o commit. Para isso execute o seguinte comando:

git commit -m “Projeto inicial”

Quando executamos um commit, precisamos fornecer uma mensagem através do parâmetro “-m”. A mensagem é importante para descrevermos, de forma breve, o que a nova versão está implementando.

Um recurso bastante útil é a visualização dos logs dos commits efetuados. Através dos logs podemos ver, por exemplo, o identificador do commit, o autor, a data e mensagem do commit.

Para visualizar o log de um repositório após a realização dos commits devemos executar o comando abaixo:

git log

## Efetuando alterações no repositório

Para compreendermos melhor o uso do git, vamos efetuar algumas alterações em nosso repositório

Quando fazemos o commit estamos “tirando uma fotografia” do arquivo naquele momento. Com o passar do tempo o arquivo sofre alterações e devemos “fotografar” o estado dele novamente. O repositório git é como o álbum de fotos da sua família, você tem várias versões das pessoas que foram fotografadas ao longo do tempo e você pode se lembrar delas olhando as fotos mais antigas.

O arquivo “index.html” sofreu alterações desde o primeiro commit, então, devemos tirar uma nova fotografia para que tenhamos a versão marcada no primeiro commit e a versão mais nova com a alteração.

Após a alteração no arquivo rode o comando “git status” novamente e observe a sua saída abaixo.

Devemos executar o comando “git add .” novamente para adicionar todos os arquivos que sofreram alteração ao “staging”.

Em seguida, execute o comando “git status” para conferir se o arquivo alterado foi adicionado ao “staging”. O resultado deverá se parecer com a imagem abaixo:

- Diferença entre commits

É bastante comum precisarmos ver o que há de diferente entre uma versão e outra de um arquivo, portanto, é possível verificarmos quais são as diferenças entre os diversos commits em nosso repositório.

Para isso utilizamos o comando “git diff”. A sintaxe deste comando é a seguinte:

git diff [<opções>]<origem><destino>

Onde,

<origem> e <destino> são os identificadores dos commits que desejamos comparar.

Assim, antes de utilizarmos o “git diff” é necessário localizarmos os identificadores dos commits que serão comparados através do comando “git log”.

O GitHub é uma plataforma utilizada pelos desenvolvedores de software para hospedagem de repositórios Git com a finalidade de controle de versões e trabalho colaborativo em projetos privados e Open Source de qualquer lugar do mundo.

# TREINAMENTO DE VERSIONAMENTO

@Prof. Celso Marcos Furtado \_ SENAI JANDIRA

## CONCEITOS & DEFIINIÇÕES

### Ramificação (Branch) e Mesclagem (Merge)

Quase todo Sistema de Controle de Versionamento tem alguma forma de suporte a ramificações (Branches). Ramificação significa que você diverge da linha principal de desenvolvimento e continua a trabalhar sem alterar essa linha principal. Em muitas ferramentas versionamento, este é um processo um tanto difícil, geralmente exigindo que você crie uma nova cópia do diretório do código-fonte, o que pode demorar muito em projetos maiores.

3 POSSIBILIDADES DE MERGE

1. Mundo ideal

- Nunca mexe na brach main

## SSITEMA DE CONTROLE DE VERSIONAMENTO

Version Control System (VSC) é um sistema que grava as alterações realizadas em um arquivo ou grupo de arquivos onde você consegue recuperar versões anteriores. (“Este é o princípio aplicado no one drive, que salva todas as versões dos arquivos de acordo com as alterações realizadas”).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **SISTEMA DE CONTROLE DE VERSIONAMENTO (VERSION CONTROL SYSTEM – VSC)** | | | | |
| **Time-Stamped directory** | **Local Version Control Systems** | **Centralized Version Control Systems (CVCSs)** | **Distributed Version Control Systems** |  |
| User copy files into another directory.  <problems> Forget directory, accidentally write to the wrong file. | Simple database that  kept all the changes to files under revision control.  Single server that contains  all the versioned files, and a number of clients that check out files from that central place | This setup offers many advantages, especially over local VCSs. For example, everyone knows to a  certain degree what everyone else on the project is doing. Administrators have fine-grained control  over who can do what, and it’s far easier to administer a CVCS than it is to deal with local databases  on every client. | Exemplo: Git, Mercurial, Darcs  - Checagem das imagens (snapshots) das últimas versões dos arquivos;  - Espelhamento completo do repositório, incluindo os históricos;  - Todo clone será um backup completo das informações; |  |

**LOCAL VERSION CONTROL SYSTEM**

Qual a motivação de utilizar um VSC?

* Concentrar as versões dos arquivos em um único repositório;
* Evitar edições sobre versões desatualizadas;

O fundamento é simples. Um arquivo dentro de um ambiente denominado Checkout possui Versões (Version Database). Uma das ferramentas mais populares dentro do VSC são os sistemas chamados RCS. RCS works by keeping patch sets (that is, the ormato n between files) in a special ormato n disk; it can then re-create what any file looked like at any point in time by adding up all the patches.

Diagrama

Descrição gerada automaticamente

Desvantagens

The most obvious is the single point of failure that the centralized server represents. If that server goes down for an hour, then during that hour nobody can collaborate at all or save versioned changes to anything they’re working on. If the hard disk the central database is on becomes corrupted, and proper backups haven’t been kept, you lose absolutely everything — the entire history of the project except whatever single snapshots people happen to have on their local machines. Local VCSs suffer from this same problem — whenever you have the entire history of the project in a single place, you risk losing everything.

### DISTRIBUTED VERSION CONTROL SYSTEMS (DVCS)

**VANTAGENS**

- Clientes enxergam um espelho do repositório, incluindo o histórico completo;

- Se o servidor cair, pode-se recuperar o repositório de qualquer cliente;

- Todo clone é um backup completo de todos os dados salvos;

-

**DESVANTAGENS**

-

-

-

**POSSIBILIDADES DE APLICAÇÕES**

**-** Possibilidade de trabalhar com diversos repositórios remotos;

- Possibilidade de trabalhar simultaneamente com o mesmo projeto;

**-** Possibilidade de utilizar o hierarchical models/

**EXEMPLOS DE DVCS**:

- Git;

- Mercurial;

- Darcs;

### SNAPSHOTS – DIFERENÇAS ENTRE GIT x Outros VCS

Enquanto grande parte dos VCS armazenam as informações com base em uma lista de alterações dos arquivos, como exemplo: CVS, Subvsersion. Perforce e outros.

### ATIVIDADES:

1. VERIFICAR SE O GIT ESTÁ INSTALADO:
   1. Abrir o cmd
   2. Digitar git –version
   3. SAÍDA ESPERADA:
      1. Linha mostrando a versão inslada.

O Git bash é um ambiente Linux, enquanto o CMD é ambiente Windows.

1. CONFIGURAÇÃO DO NOSSO REPOSITÓRIO
2. QUERO COMEÇCAR UM PROJETO
3. Criar uma pasta: md turma3
4. No Bash, coloco

Quando tem dr = diretório

Tudo o que tem l é um atalho

Azul claro é diretório.

Branco é arquivo.

Turma3 não é um repositório git.

Se for um repositório, no cmd eu vou rodar o git status. Vouter a respota se a pasta é um repositório ou não.

No Git bash não preciso rodar o comando.

Como transformo a pasta em repositório?

Posso digitar cd .

Dir/ a = mostra se tem algo oculto.

Comando no CMD

git init

Se você apagar a pasta .git, perde-se tudo.

Quando você não quiser que a pasta seja mai sum repositório, é só apaga-la.

|  |  |  |  |
| --- | --- | --- | --- |
| **COMANDO – no GIT BASH** | **VIA .CMD** | **PONTO DE CHECAGEM** |  |
| git branch -M main |  |  |  |
| mkdir turma3-teste |  |  |  |
| git init |  | Precisa ser executado dentro da pasta que eu quero transformar em |  |
| git init dev |  | Já transforma uma pasta chamada dev de repositório |  |
| touch index.html |  | Cria um arquivo texto vazio |  |
| ls |  | Mostra a lista que temos salvo |  |
| git status |  | Comando mais utilizado para verificar status de commits. |  |
| git commit -m |  | Antes de rodar, preciso rodar um comando |  |
| git config -–global user.name “André Aoki” |  |  |  |
| git config -–local user.name “André Aoki |  | Reflete somente no local, no repositório que eu estou. |  |
| git config -–global e-mail “andre.aoki@sp.senai.br” |  |  |  |
| git config –global –unset user.mial |  | Quando voc~e criar algo errado |  |
|  |  |  |  |

Git Wordkflow:

Untracked: Ainda não rastreáveis;

Staged: Após o comando “git add”, os arquivos estão disponíveis para o próximo commit

Commited: Após o comando “git commit -m” os arquivos atualizados

A partir do segundo commit, os arquivos alterados possuem os seguintes estados:

- Modified: estão diferentes em relação o último “commit”;

- Unmodified: Não possuem diferenças em relação ao último “commit”;

Diagrama

Descrição gerada automaticamente

Ctrl + X

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | LINUX FEDORA | LINUX UBUNTO | MACOS | WINDOWS |
| Instalação do Git | $ sudo dnf install git-all | $ sudo apt install git-all | $ git –version  <https://git-scm.com/download/mac> | <https://git-scm.com/download/win> |
|  | $ sudo dnf install dh-autoreconf curl-devel expat-devel gettext-devel \  openssl-devel perl-devel zlib-devel  $ sudo apt-get install dh-autoreconf libcurl4-gnutls-dev libexpat1-dev \  gettext libz-dev libssl-dev |  |  |  |
|  | $ sudo dnf install asciidoc xmlto docbook2X  $ sudo apt-get install asciidoc xmlto docbook2x |  |  |  |

# PRIMEIROS PASSOS

- Criar uma pasta na área de trabalho;

- Vamos criar um repositório

- I rno profile > REpositóprios > New > Nome do repositório > ex: nome do projeto e o dash board >

- Entre na pasta onde está o Git Hub

- Na barra de pesquisa, digite cmd <Entrer>

- criei um arquivo na pasta do Git Hub

- digitar git add . no cmd (cuidado, tem que ter um espaço antes do “.”)

- digitar

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

# REFERÊNCIAS

Git: <https://git-scm.com/book/en/v2>