GIT

https://itau.udemy.com/course/git-completo-do-basico-ao-avancado/learn/lecture/13897466#overview

Sobre o controle de versão

* Histórico de alterações
  + É possível restaurar versão anterior sem prejuízo
* Pequenos ou grandes projetos
  + Pode ser utilizado por frentes diferentes de trabalho
* Ramos (branches)
  + Diferentes níveis à partir de um mesmo ponto do projeto
  + Mesclar funcionalidades após testadas
* Rastreabilidade
  + Identifica quando e quem realizou a mudança no código em determinado momento

Tipos de Versionadores

Centralizado

Nesse modelo o servidor contém todo o histórico e foi padrão por muitos anos. Ex.: Subversion (SVN) criado em 2000, GIT criado no final de 2005

Diagrama

Descrição gerada automaticamente

Se o servidor não tiver a proteção de backup, etc., e ocorrer um problema no servidor, o projeto é impactado.

Distribuído

Nesse modelo o servidor e as máquinas contêm todo o histórico do versionamento.

Diagrama

Descrição gerada automaticamente

Toda máquina possui uma cópia do repositório

Mudanças são realizadas localmente

Não depende de uma única máquina

Informações sobre o funcionamento

STATUS

Texto

Descrição gerada automaticamente

Untracked files: arquivos não rastreados pelo GIT (ainda), para adicionalos ao versionamento, executar “git add <nome do arquivo>”

Texto

Descrição gerada automaticamente

No exemplo abaixo existe uma versão já adicionada do arquivo “commands\_GIT.txt” para ser commitada, no entanto, ainda há mudanças nesse mesmo arquivo que não foram adicionadas.

Texto

Descrição gerada automaticamente

Tela preta com letras brancas

Descrição gerada automaticamente

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente



Logotipo

Descrição gerada automaticamente



Texto

Descrição gerada automaticamente

Estagios de um arquivo

Diagrama

Descrição gerada automaticamente

Untracked: arquivos ainda não rastreados

Unmodified: arquivo commitado

Modified: arquivo que após o commit, sofreu alguma alteração

Staged: arquivo após modificado, adicionado na área para realizar o commit

Para comparar as alterações realizadas:

Texto

Descrição gerada automaticamente

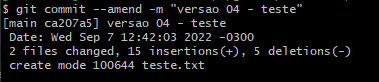
Para listar os commits

Os comits são listados em ordem decrescente

Texto

Descrição gerada automaticamente





Texto

Descrição gerada automaticamente

Movendo entre os commits

Texto

Descrição gerada automaticamente

DESFAZER Alterações no repositório

Diagrama

Descrição gerada automaticamente

GIT IGNORE

https://github.com/github/gitignore

Conceito de **FORK**

 O fork serve para realizar uma “copia” individualizando o projeto para dominio proprio sendo possível realizar alterações e contribuir (ou não) com o porjeto origem

Conceito de **PULL REQUEST**

 Proposta de mudanças que são encaminhadas como contribuição para o projeto origem qual foi realizado o fork/branch.

Conceito de **ISSUES / Labels / Milestone**

 Problema capturado por qualquer um, que é reportado e assim fica visível para comunidade toda que acompanha o projeto.

 Usado para categorizar os problemas capturados

 Usado para definir um determinado marco / backlog/ agrupamento de Issues

Tela de celular com aplicativo aberto

Descrição gerada automaticamente

Conceito de **README**

Fornecer um Resumo/Documento/Apontamento incial do Projeto ou subdiretorios para que tenha maior facilidade de se obter informação sobre o repositorio.

<https://dillinger.io/>

Conceito de **SSH**

Utilizado para o GIT\_BASH conseguir realizar a comunicação com o REPO

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

Nas configurações do REPO é possivel realizar o cadastro da chave SSH gerada.

A configuração é gerada por conta e não por repositório, uma vez gerada, não será necessário fazer para cada um dos repositorios existentes.

----------------

gerar chave SSH

----------------

ssh-keygen

cd ~/.ssh/

start ~/.ssh/

id\_rsa.pub <- chave gerada (caso pode ser renomeado e adicionado posteriormente)

eval $(ssh-agent)

ssh-add ~/.ssh/id\_rsa

Conceito de **BRANCHES**

Branch é uma ramificação no projeto que permite que funcionalidades sejam desenvolvidas separadamente sem impactar as funcionalidades estáveis no projeto.

Gráfico, Esquemático, Gráfico de bolhas

Descrição gerada automaticamente



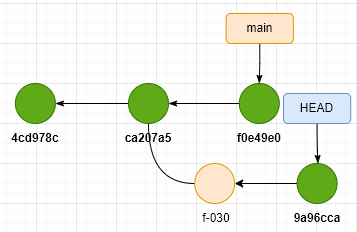


Diagrama

Descrição gerada automaticamente

Diagrama, Esquemático

Descrição gerada automaticamente



Texto, Tabela

Descrição gerada automaticamente

Conceito de **MERGE**

Integrar os fontes desenvolvidos em um ramo na branch atual. Ou seja, caso esteja na branch develop e execute o merge da branch main, significa que vai mesclar os dados da main para develop.

Mais natural o caminho de ter uma branch “PRODUCAO, HOMOLOGACAO E DESENVOLVIMENTO”; Na branch HOMOLOGACAO é realizado merge da DESENVOLVIMENTO e em PRODUCAO realizado o merge da HOMOLOGACAO.

O Merge trás as mudanças realizadas em determinado ramo para o ramo atual.

Interface gráfica do usuário, Texto

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente com confiança média

Tela de celular com aplicativo aberto

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

Ou seja, temos o ramo “develop” e o “main” no mesmo ponto independente do HEAD apontando para qualquer um que seja após o MERGE, diferente do ramo “testehead”

Conflitos

Os conflitos podem ocorrer quando existirem alterações em branchs que diferem em um mesmo arquivo.

Tela de computador com jogo

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

Linha do tempo

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

É possível constatar que após a resolução do conflito, o fonte desenvolvido no ramo ‘testehead’ agora aparece junto ao projeto novamente na ‘develop’, mas ainda não na ‘main’ pois esta está atras da ‘develop’

Conflito Maior

Texto

Descrição gerada automaticamente com confiança média

Tela preta com letras brancas

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

Interface gráfica do usuário

Descrição gerada automaticamente com confiança baixa

Texto

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

Interface gráfica do usuário

Descrição gerada automaticamente com confiança média

Interface gráfica do usuário, Texto

Descrição gerada automaticamente

Interface gráfica do usuário, Texto

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Uma imagem contendo Texto

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

Após o PUSH dos fontes dos ramos ‘develop’ e ‘main’ os remotos foram para o mesmo ponto. O ramo ‘testehead’ uma vez implementado no ramo ‘develop’ e com novas implementações posterior a evolução do projeto, pode ser removido para não causar equívocos ou podemos mantê-lo por determinado tempo por questão de rastreabilidade.

Para um caso de linha de produção, os branchs são criados a partir de ‘develop’ que contêm o fonte completo, é então desenvolvido a funcionalidade e realizado o PULL REQUEST para MERGE do fonte destacado de ‘develop’ para implementar na esteira, assim, é encaminhado para o servidor de desenvolvimento o fonte após as validações da esteira. Se todas as validações ocorrerem com sucesso, automaticamente é criado no final um PULL REQUEST para esteira de ‘homologação’ responsável por entregar os fontes no servidor de homol.

Antes de chegar em Produção todo fonte deve ser testado e validado, bem como passar pelo processo de GMUD.

Uma vez realizada com sucesso e passado o tempo de validação ai o MERGE REQUEST deve ser aceito em PRODUÇÃO (main) caso contrário deve ser realizado rollback nos fontes para versão funcional anterior.

Conceito de TAG

TAG é um ponteiro em commit específico na história do projeto independente de branch ou HEAD. Sua função é marcar um ponto importante no projeto.

Diagrama, Esquemático

Descrição gerada automaticamente

Tela de celular com aplicativo aberto

Descrição gerada automaticamente

Cada TAG pode representar uma versão do projeto e sendo atrelada ao commit fica facil identificar e rastrear o ponto do projeto com a devida TAG

Tela de celular com fundo preto

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

Tela de celular

Descrição gerada automaticamente

Conceito de STASH

O stash serve para salvar um estado do fonte sem a necessidade de efetuar um commit, para poder aplicar em momento oportuno.

Texto

Descrição gerada automaticamente

Quanto maior o numero, mais anterior o momento do stash.

Apendice A – Comandos

# lista as configurações globais

git config --global -l

# gerenciamento das configurações globais

git config --global user.name "Fulano de Tal"

git config --global user.email "fulano@de.tal"

git config --global init.defaultBranch main

git config --global core.editor "code --wait"

# inicializa o versionamento de um repositório

git init

# exibe o status do repositório

git status

# adiciona um ou mais arquivos ou todos (., -a, --all)

git add <file>

git add .

git add -a

git add --all

# exibe o que há de modificado entre as versões do arquivo "modified" e o ultimo conhecido, se houver na área "staged" a comparação é realizada por lá, senão é realizada na "unmodified"

git diff

# para realizar a comparação entre a "staged" e "unmodified"

git diff --cached

git diff --staged

# exibe os comits de forma decrescente

git log

git log --oneline

git log <branch> --oneline

git log -2

# exibe as modificações do ultimo commit

git log -p -1

# exibe um resumo dos arquivos alterados nos commits

git log --stat

git log --shortstat

# modifica a mensagem do commit (na realidade o commit cria um novo commit baseado no ultimo HEAD)

git commit --amend -m "alteracao da mensagem do commit"

git commit --amend --no-edit

# DESFAZENDO ALTERAÇÕES

# reverte / navega entre os commits

git checkout <numero do commit>

git checkout main

git checkout <file>

git checkout .

# remove arquivos não rastreados

git clean -f

# remove um ou mais arquivos ou todos

# --cached opção para 'unstage' e remover caminhos apenas do indice se não houver um HEAD

git rm --cached <file>

git rm --cached -rf .

# sem a opção --cached vai apagar literalmente o arquivo

git rm 'd\*'

git rm 'd/\*'

# salva uma versão do arquivo até o momento do commit

# é possivel comentar em multiplas linhas se não encerrar o (") caso precise detalhar melhor o commit

git commit -m "versao teste"

# descarta as alterações após uma versão do arquivo até o momento do ultimo commit (precisa de um HEAD para funcionar)

git restore <file>

# opção para 'unstage' -> remove a alteração da área preparada para o commit

git restore --staged <file>

# remove todas as alterações realizadas voltando ao ultimo commit (trackeds)

git reset --hard

# para parar de rastrear um arquivo caso adicionado após no gitignore ou rastrear

git update-index --skip-worktree <file>

git update-index --no-skip-worktree <file>

# realizar uma copia de um repositorio local/remoto

git clone repositorio novoRepositorio

git clone https://github.com/usuario/gitcursocompleto.git

# verificar se o repositório está vinculado a um remoto ou não

git remote -v

## origin https://github.com/usuario/gitcursocompleto.git (fetch) -> busca informações

## origin https://github.com/usuario/gitcursocompleto.git (push) -> envia informações

# adicionar links de repositorios remotos

git remote add <alias> <url do repositorio>

git remote add origin https://github.com/usuario/gitcursocompleto.git

# alterar links de repositorios remotos

git remote set-url <alias> <url do repositorio>

git remote set-url origin https://github.com/usuario/gitcursocompleto.git

# enviar commit para repositorio remoto

git push

git push --set-upstream origin <nome do ramo>

git push -u origin <nome do ramo>

# restaurar do repositorio remoto

git pull

# listar os ramos no projeto

git branch

git branch --list

# criar um ramo no projeto

git branch <nome do ramo>

git checkout -b <nome do ramo>

git switch -c <nome do ramo>

# renomear um ramo no projeto

git branch -m <branchAnterior> <novaBranch>

git branch -m <novaBranch> (caso esteja na branch a ser renomeada)

# trocar para um ramo no projeto

git checkout <nome do ramo>

git switch <nome do ramo>

# trocar para um ramo no projeto desfazendo alterações realizadas na branch anterior

# caso tenha adicionado e não commitado, as alterações serão perdidas

git checkout -f <nome do ramo>

# trocar para o ramo anterior usado no projeto

git switch -

# remover um ramo no projeto LOCAL

git branch -d <nome do ramo>

git branch -D <nome do ramo>

# remover um ramo no projeto REMOTO

## quando executado a remoção remota, é necessário excluir de todos os demais locais distribuidos para garantir que um PUSH não retorne a branch

git push --delete origin <nome do ramo>

# realizar MERGE de um ramo no projeto

git merge <nome do ramo>

# abortar o MERGE de um ramo no projeto

git reset --hard

git merge --abort

# localizar ramo que não foram ou foram MERGEadas no projeto respectivamente

git branch --no-merged

git branch --merged

----------------

gerar chave SSH

----------------

ssh-keygen

cd ~/.ssh/

start ~/.ssh/

id\_rsa.pub <- chave gerada