

Aula 2.1 Git & Github

• Git

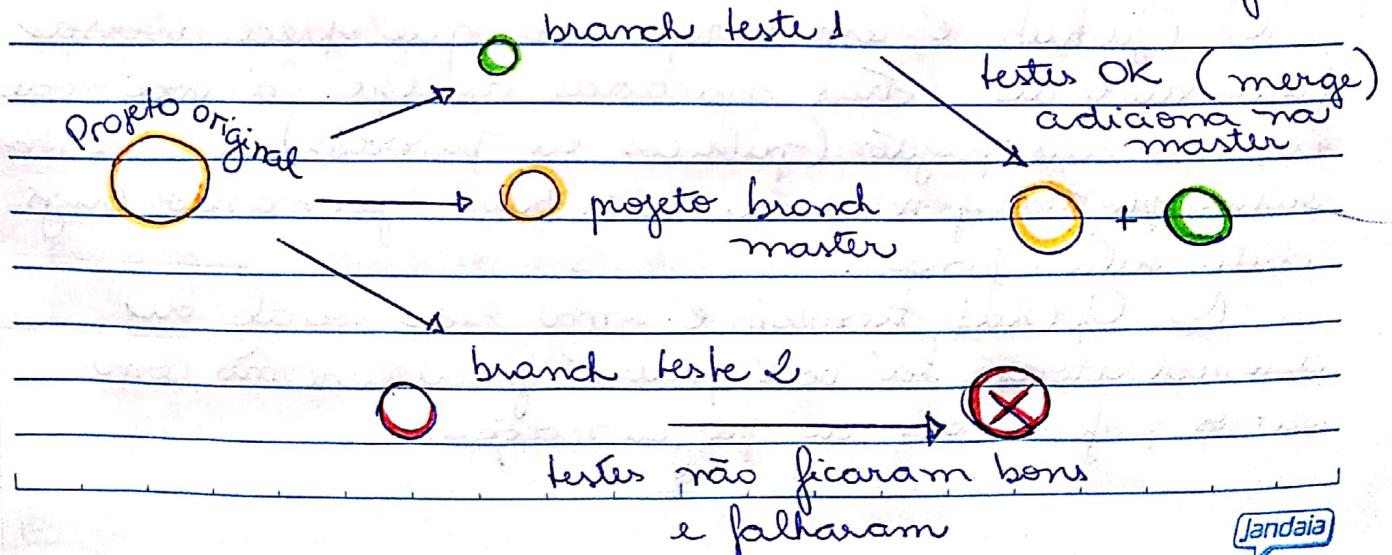
É um sistema open-source de controle de versão utilizado por desenvolvedores. Com ele é possível criar um histórico de alterações no código de um projeto e facilmente voltar para qualquer ponto.

Todas essas alterações na história do projeto são chamadas de commits. Cada commit tem uma descrição das alterações realizadas nesse ponto.

Essas informações e detalhes são importantes para você e outros colaboradores do projeto saberem as etapas.

Branches são ramificações de um projeto. Por padrão o Git cria automaticamente um branch chamado master. Às vezes queremos fazer alterações para testar novas funcionalidades em um projeto e é importante não fazer alterações/testes na master.

Às vezes estas alterações são apenas testes, que podem não dar certo, e ^{ainda} não serão adicionados a branch master. Contudo, podemos criar uma branch para trabalhar nela e caso estas alterações fiquem boas podemos colocar elas na master (através do merge).

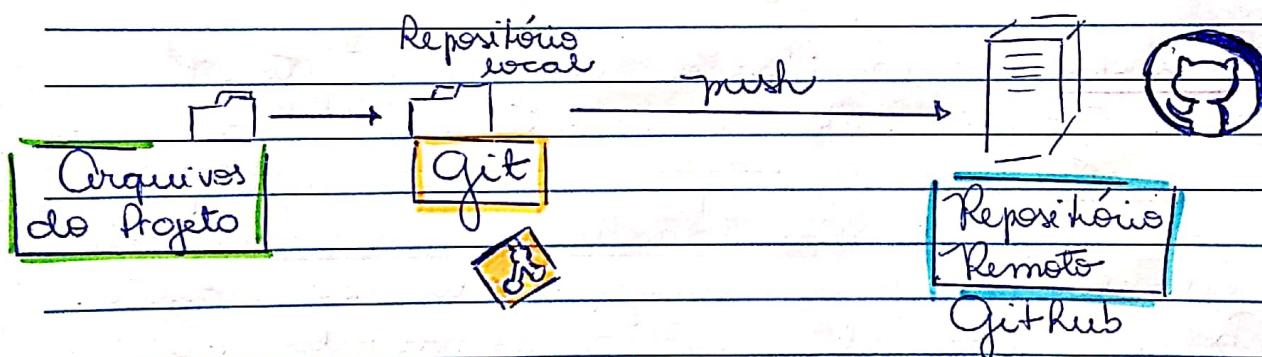


Como vimos o merge é uma fusão de duas branches. Os merges podem funcionar perfeitamente mas algumas vezes o git pode ficar confuso e gerar um conflito ao realizar um merge.

• Principais vantagens Git:

- Controle do histórico
- Trabalho em equipe (consegue juntar + merge)
- Ramificações do projeto
- Segurança
- Organização

Estrutura do Controle de versão



④ Não existe apenas o Github. Outros: GitLab, BitBucket, Phabricator, Gogs ...

O Github é um serviço web que oferece diversas funcionalidades extras aplicadas ao Git. Daí você pode hospedar seus projetos (públicos ou privados), acompanhar outros projetos open source, contribuir informando bugs entre outras funcionalidades.

O Github também é uma rede social de desenvolvedores. Daí você poderá fazer conexões com outros profissionais de programação.

data

S T Q Q S S D

Etapas Principais:

- Abrir uma conta Github
- Instalar Git e suas configurações

• O que é a chave SSH?

• Por padrão a conexão do Git com servidores como o Github é feita pelo protocolo HTTPS, contudo isso te obriga a digitar o usuário e senha toda vez que executar um comando como git pull ou git push. Usando o protocolo SSH, você pode se conectar a servidores remotos e se autenticar para utilizar seu serviços. O Github (por exemplo) permite ao Git se conectar via SSH em vez de HTTPS. A conexão então é feita com criptografia de chaves.

• Para verificar se você já possui a chave SSH:

```
% ls -lah ~/.ssh
```

Se ele informar que a pasta ~/.ssh não existe significa que você ainda não tem um par de chaves SSH.

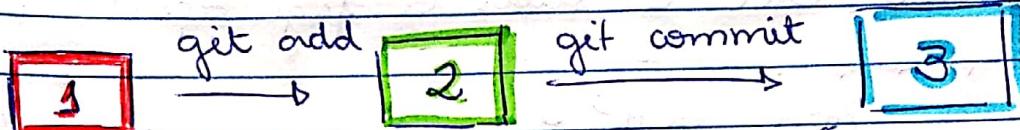
• Caso você já tenha você verá estes dois arquivos:

```
{ id_rsa (chave privada)
{ id_rsa.pub (chave pública)
```

Obs: A chave pública é adicionada nas configurações do Github

data • •
S T Q Q S S D

Q Git tem três estados principais que seus arquivos podem estar:



Modificado
(Os arquivos foram criados ou alterados)

Preparado
(separou os arquivos para fazer parte de um commit)

Comitado
(Os arquivos foram armazenados de forma segura no seu repositório local)

obs: O repositório remoto pode ser um servidor, um outro computador, Github, Gitlab ...

• Iniciando um repositório

Nas pastas do computador crie um diretório para seu projeto. Por exemplo:

```
% mkdir "Meu primeiro Projeto"
```

Pelo editor de textos (VS code) crie um arquivo. Por exemplo: index.html e salve na sua pasta. Verifique que você está na pasta do projeto e que o arquivo está na mesma.

```
% ls | → verifica os arquivos do diretório atual
```

• Para iniciar um repositório do Git nesta pasta: digite o comando

```
% git init
```

Obs Para verificar-se desta etapa, ao usar o comando:

% git status

Ele verifica as condições e informações do diretório.
No caso aparece no terminal

on branch master → mostra a branch que você está

No commits yet

Untracked files:

(use "git add <file>..." to include ...)

index.html → mostra que há um arquivo que foi modificado ou alterado

nothing added to commit ...

Neste caso como o arquivo index.html acabou de ser criado, ele aparece como ainda não adicionado para futuros commits.

Obs Caso não tenha nenhum arquivo o terminal mostra no comando git status:

terminal } on branch master

nothing to commit (working directory clean)

CONTINUANDO ...

Com o arquivo index.html na pasta precisamos separar ele e preparar para o futuro commit. Para isso primeiro vamos adicionar ele:

% git add index.html

Obs: E se tivermos vários arquivos para adicionar?
 Podemos usar o comando % git add.

Após adicionar ao fazer o comando % git status

on branch master

Terminal } No commits yet → mudanças pt comitá-las
 Changes to be committed:
 (use "git rm --cached <file>" to unstage)
 new file: index.html

• Primeiro Commit

O commit adiciona um ponto no histórico do projeto (uma alteração no arquivo, uma adição de outro arquivo, até mesmo quando deletámos). Dendo assim, neste ponto é necessário uma mensagem para saber o que aconteceu tanto para quando revisarmos o projeto, quanto para outros colaboradores saberem.

Prestar caso para adicionar um commit:

% git commit -m "Orição do arquivo.html"

O parâmetro -m serve para passar a mensagem que será colocada entre aspas.

Ao dar enter no comando ele salva o commit e ao verificar com o comando % git status irá aparecer que não há mais arquivos pt ser comitados.

Obs: As mensagens dos commits devem ser objetivas e claras.

Caso você faça alguma alteração neste arquivo ou a adição de novos arquivos ocorrerá o mesmo processo.

↓ % git add arquivo

↓ % git commit -m "Descrição do Commit"

Obs: É recomendável sempre verificar o estado dos arquivos com o comando %git status

• Para verificar o histórico de alterações:

% git log

→ Mostra no terminal detalhes como autor, data e hash do commit.

* O que é a hash do commit?

É um código com números e letras que identifica o commit. Quando fazemos o comando %git log ele aparece na frente de cada commit.

commit: 3a8ff6905a59...

Obs: Esta identificação é única. Cada commit tem uma #

• Para visualizar uma mensagem mais resumida do %git log é através do comando:

% git log --oneline

Ele passa as informações de cada commit em uma linha. O hash do commit também é resumido em apenas os primeiros caracteres.

- Para ver as informações de maneira detalhada:

% git log -p

→ mostra até as linhas que foram alteradas

* Git Ignore

É importante para aqueles arquivos que você não quer incluir no rastreamento ou não devemos comitar. Para evitar que isso ocorra criamos um arquivo especial que indica ao Git quais arquivos não queremos rastrear. O nome do arquivo criado será .gitignore e ele é um arquivo de texto que fica oculto. Ele é colocado no diretório raiz do projeto.

Quais arquivos costumam ser ignorados?

- Arquivos de log
- Arquivos com senhas ou informações confidenciais.
- Arquivos do sistema e temporários
- Arquivos gerados, como executáveis de teste
- Saídas compiladas (executáveis)
- Arquivos pessoais

Por exemplo, estamos em um projeto e um dos arquivos do diretório contém os dados pessoais dos colaboradores daquele projeto. Este arquivo é importante no meu projeto mas não queremos que o Git fique monitorando as alterações dele e não temos interesse de comitá-lo.

Passo a passo:

① Entrar no diretório do projeto

`cd ~ / home / debara / Documentos / Projeto`

② Criar o arquivo .gitignore:

`% touch .gitignore`

Para verificar se ele foi criado através do comando:

`% ls -a`

pois o .gitignore é um arquivo oculto

Obs: Vamos criar um arquivo exemplo para ser ignorado. Por exemplo o arquivo com dados pessoais dos colaboradores do projeto

`% touch informacoesColaboradores.txt`

Nell inserimos nomes, telefones, email e etc mas como foi visto este arquivo queremos que o Git ignore e não gosteie / comite ele.

`% git status`

On branch master

Untracked files:

• .gitignore

• informacoesColaboradores.txt

No terminal mostra que os dois arquivos foram criados e ainda não foram adicionados. Mas como queremos ignorar o arquivo `informacoesColaboradores.txt` vamos inserir o nome deste arquivo no conteúdo do .gitignore. Deste modo:



A primeira linha é um comentário

Na segunda linha tem o nome do arquivo que desejamos ignorar.

data . .

S T Q Q S S D

% cat >> .gitignore

Este é o arquivo que será ignorado:
informaçõesColaboradores.txt

[CTRL + D] para finalizar o envio de dados do comando

Neste caso o arquivo .gitignore também poderá ser editado em um editor de textos (por exemplo VS Code).

Poderemos ignorar apenas um arquivo ou poderemos ignorar um diretório, ou vários arquivos que possuem a mesma extensão por exemplo:

Exemplos de padrões

NovoArquivo.txt → ignora o arquivo

diretórioTeste/ → ignora o diretório e seu arquivos

*.log → ignora todos os arquivos extensão log

teste* → ignora todos os arquivos iniciados com teste

!dados.txt → NÃO ignora o arquivo dados.txt

Este sinal indica negação. Por exemplo, você ignorou todos os arquivos *.txt mas esse especificamente você não quer ignorar. É só fazer essa observação.

Continuando ...

Após inserir o informaçõesColaboradores.txt dentro do arquivo de texto do .gitignore ao realizar o comando % git status:

terminal
On branch master

Untracked files:

• .gitignore

data . . .

S T Q Q S S D

Verificamos assim que o arquivo `informações Colaboradores.txt` não está mais listado para adicionar e comitar.

Isto acontece pois o `.gitignore` já está ignorando o arquivo. Depois só falta add e comitar o arquivo `.gitignore`.

`% git add .gitignore`

`% git commit -m "Criado o arquivo .gitignore"`

Agora todos os arquivos e diretórios do projeto que desejamos ignorar podemos adicionar no Arquivo `.gitignore`.

Adicionando os arquivos no Repositório Remoto

Agora que já estamos versão nando nossa pasta do projeto no computador podemos também colocar ela em um repositório remoto. No caso o Github por exemplo.

①º Criar um repositório no Github.

Na aba Repositories tem um botão  **New** que abrirá uma página solicitando o nome do repositório, se deseja coloca-lo público ou privado, descrição... Cio final...

`Create repository`

Pronto!

②º Voltando ao terminal verifique se você está na pasta de projeto e se adicionou e comitou os arquivos. Com os comandos: `% ls`

`% git status`

`% git log`



Após add e comitar os arquivos precisamos adicionar este projeto ao novo repositório criado no Github

`% git remote add origin https://github.com/...`

Nesta etapa você informa o link do repositório. Caso você tenha criado a chave SSH clique na opção que tem o link SSH.

Após isso deve-se fazer o PUSH dos arquivos:

`% git push -u origin master`

→ vincula com o repositório
(Upstream)

não é obrigatório
desconectar

Para verificar abra a página do Github e abra o repositório e os arquivos estarão lá.

Caso altere os arquivos ou adicione novos a etapa é praticamente a mesma

`% git add arquivo`

`% git commit -m "Descrição do commit"`

`↓ % git push origin master`

Clonar um repositório

Caso você já tenha um projeto no Github com arquivos e na sua máquina ainda não tenha você poderá clonar este projeto.

O clone gera uma cópia identica do repositório na sua máquina (todos os arquivos).

`% git clone git@github.com:...`

link do projeto

Obs Mas se já temos o mesmo arquivo na nossa máquina porém ele não está atualizado?

Digamos que você está em um projeto com outros colaboradores e você tem a versão inicial do projeto da master. Porém há uma atualização do projeto no repositório remoto (Github) e você quer atualizar a cópia na sua máquina.

Para isso usamos o **git pull**

% git pull origin master

O comando pull, na verdade é a combinação de outros dois comandos: git fetch e git merge.

De um modo simples de exemplificar o git fetch busca as novas atualizações daquele projeto (as diferenças) e já o git merge faz a junção dessas diferenças (dos commits) com o seu branch.

Você poderia baixar todos os novos commits com o git fetch sem afetar seu código local (sem fundir com a sua cópia).

Branches

Projeto
(Master)



Projeto: Colaborador (João)

ex: vai trabalhar com o cabeçalho da página (Branch: João)



Projeto: Colaborador (Maria)

ex: vai trabalhar com o corpo da página (Branch: Maria)

Neste exemplo João e Maria dãoem trabalhar no projeto em diferentes áreas e para isso criam f braches do mesmo projeto (master).

Assim, eles não alteram as configurações iniciais da master. A master já é criada por padrão quando você inicia um repositório no git.

Criar uma nova branch:

% git checkout -b Nome-da-branch

por exemplo:

Maria

comando de forma
curta

As branches são ramificações do projeto. Um projeto pode ter várias branches.

Quais as vantagens das branches?

• É possível desenvolver novos recursos para nossa aplicação sem impedir o desenvolvimento na branch principal (por exemplo na master)

• É possível dividir as branches em versões de testes, em gitter para diferentes áreas de trabalho (por exemplo diferentes setores de uma página web), pelos colaboradores etc...

(*) Para visualizar as branches de um projeto:

Entre na pasta onde está o projeto versionado

Comando:

% git branch

Caso ainda não tenha criado uma nova branch o terminal mostrará apenas a branch master.

Para criar na nova branch:

% git branch Nome-da-branch

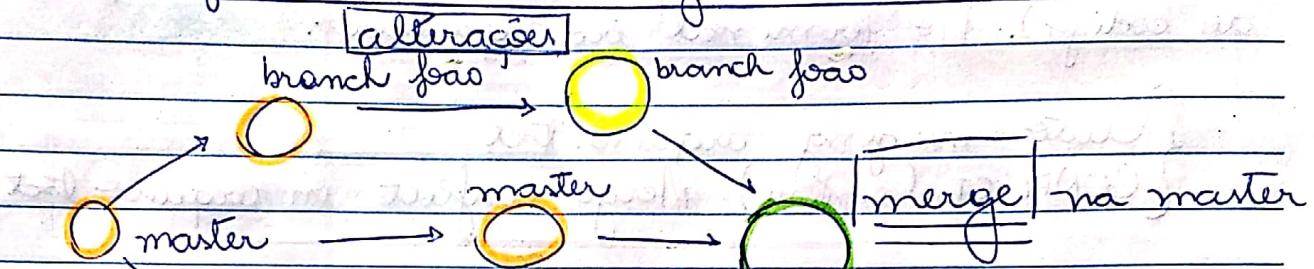
Para entrar na nova branch:

% git checkout Nome-da-branch

- Para mudar de branch

`% git checkout Nome-da-branch`

- Unificar as branches (juntar os trabalhos)



Quando queremos unificar o trabalho por exemplo da branch foao com a branch master.

`% git checkout master`

nome da branch

`% git merge foao`

principal p/ onde queremos juntar as alterações

mas atenção! Git verifica se é compatível a função das branches, contudo, caso a branch foao tenha um conflito com o conteúdo da branch master devemos corrigir manualmente

Obs: Se você fizer o comando: `% git log --online` ele irá mostrar os commits apenas da branch que você está. Caso você quira ver os commits de todas as branches do projeto você pode usar o comando:

`% git log --online --graph --all`

mostra em
1 linha cada
commit
(+ resumido)

mostra
de todas as
branches
facilita visualizar
as ramificações

- Fazendo o merge:

Pode mos juntar os trabalhos direto pelo terminal através do git ou também via Github, por exemplo.

Há também casos onde os projetos alterados geram conflitos durante o merge (por exemplo as pessoas alteraram de forma diferente uma mesma parte do código). No terminal irá aparecer:

{ Auto-merging arquivo.txt

CONFLICT (content): Merge conflict in arquivo.txt

- Abriendo via terminal ou via editor de texto (no computador) ou até mesmo via Github (merge feito via Github) o arquivo irá mostrar onde está em conflito:

Dá para editar manualmente:

<<<<<

Esta linha foi modificada no nome mante

Esta linha foi modificada no nome foco

(m)

>>>>>

No terminal o Git te dá opções para selecionar o conflito:

Accept Current Change/Accept Incoming Change/Accept Both

Seleciona manter as alterações feitas em uma das branches

Seleciona manter as alterações da outra branch

Mantém as duas alterações (as duas linhas por exemplo)

- Depois de fazer as alterações do arquivo continuar com os comandos para salvar e fazer o commit das alterações para concluir o merge.

data

S T Q Q S S D

% git add arquivo.txt

% git commit -m "Alteração do arquivo para merge"
Merge concluído!

Caso ocorra o conflito e eu quiser cancelar o merge, utilizar o comando:

% git merge --abort

• Atualizando o merge no repositório remoto:

% git push origin master

* Excluindo um repositório

• Caso você queira excluir um repositório no Github siga esses passos:

- 1 Entrar no repositório que você deseja excluir
- 2 Clicar na opção "Settings" (Configuração)
- 3 No final da página aparece uma área de cor vermelha "Danger Zone". Nesta área existem configurações especiais e delicadas como:
 - alterar perfil do repositório p/ público/privado
 - Transferir a posse do repositório p/ outro usuário
 - Arquivar repositório
 - Deletar repositório

Obs: Por questão de segurança (Para ter certeza que você quer deletar ele pede para você digitar o texto).

Confirmar deletar o arquivo

- Caso você quisesse deletar o arquivo / repositório no Git (é importante para rastrear arquivos que foram excluídos):

Para excluir um arquivo do repositório que foi criado e comitado:

```
% git rm arquivo.txt
```

Quando usamos o comando `%git status` ele irá informar que houve a alteração que deve ser comitado:

```
% git commit -m "arquivo.txt foi deletado"
```

Obs: E se deletarmos um arquivo mas quisermos recuperar ele? Precisaremos pegar o número do commit da exclusão (hash)

```
% git log --diff-filter=D --summary
```

Commit bf b414 ad79d ...

Author: _____

Date: _____

arquivo.txt foi deletado

pegar o nome dele (as 4 primeiras letras já é suficiente)

```
% git checkout bf b4~1 arquivo.txt
```

hash

nome do arquivo deletado
(importante colocar nome)

Ao colocar o comando `%ls` você já encontra novamente o arquivo no seu diretório

• Desfazer alterações no git:

→ Caso fizemos alterações em um arquivo mas ainda não adicionamos ele e nem comitarmos use o comando:

`% git checkout -- arquivo.txt`

→ Caso já fizemos a adição desse arquivo:

`% git reset HEAD arquivo.txt`

→ Caso você tenha comitado o arquivo:

1º Busque o hash pelo comando: `% git log`

↳ identificação do commit

2º `% git revert 97b5ef9f1502 ...`

↳ hash

• Salvando as alterações de um arquivo

Caso você tenha alterado algum arquivo mas não queria comitar ele ainda pode mos apenas salvar as alterações. Assim você consegue rever aquelas alterações e quando quiser + cito comitá-las.

Após fazer as alterações em vez de adicionar o arquivo e comitar utilize o comando:

`% git stash`

• Para ver a lista de arquivos salvos pelo stash use o comando:

`% git stash list`

Para aplicar as modificações que ficaram salvas use o comando:

% git stash pop ①

→ Esse número ① se refere ao item da lista (% git stash list) pois às vezes temos ① um arquivo salvo

Navegando no tempo dos commits

Vamos imaginar que estamos trabalhando em um código mas queremos "voltar no tempo" e fazer com que seu código volte ao estado anterior na linha de commits.

% git checkout e6f684b3

→ hash do commit
 (os 6-7 primeiros caracteres já são suficiente para identificar)

Quando voltamos em algum commit assim nós saímos da linha do "tempo" daquele projeto. Podemos testar alguma alteração mas caso desejamos manter aquelas novas alterações devemos criar uma branch antes de fazer as alterações. Fazendo assim:

1º Volta no commit desejado

% git checkout hash

2º Cria uma nova branch

% git checkout -b Nova-branch

→ nome da nova branch

3º Faz as alterações, add e commita

4º Caso tenha interesse faça o merge com a master

• Verificando as alterações entre commits

As vezes queremos ver as alterações realizadas nos commits para comparar as mudanças no projeto.

Isto pode ser realizado pelo comando `% git diff`.

Ele nos permite ver as diferenças nos códigos entre commits, branches e até no código que estamos digitando mas ainda não adicionamos nem comitamos.

`% git diff`

este comando sozinho mostra as alterações do arquivo que você está trabalhando

de hash

outro hash do commit

`% git diff 5fe7b3..fb49fhe`

Lá

As alterações aparecem da seguinte forma:

+ linha adicionada

- linha removida

- linha modificada

+ linha modificada

O sinal de subtração indica que esta linha não está mais presente no código

, mostra uma nova linha