



Introdução ao Repositório GIT

Semana Nacional de Ciência e Tecnologia

Um pouco sobre a história do GIT

- No início da década, os sistemas de controle de versão mais utilizados eram o SVN e Subversion.
- Ambos possuíam diversos problemas, como riscos de segurança, falhas em operações, dentre outros.
- A importância de Linus Torvalds sobre críticas a esses sistemas de controle foi algo primordial na época.
- Nessa mesma época, o Kernel do Linux era desenvolvido através de métodos chamados de Patches e Tarballs.

Um pouco sobre a história do GIT

- O uso de Patches e Tarballs era trabalhoso, porém Linus afirmava que era melhor do que usar SVN ou Subversion.
- Linus Torvalds conseguiu melhorar o versionamento do Kernel Linux graças a descoberta do BitKeeper.
- Após o BitKeeper se tornar pago, Linus desenvolveu o Git, com a promessa de versionamento eficiente e prático.
- Pouco tempo depois, Junio Hamano entrou no projeto, acrescentando comandos de baixo nível ao Git, tornando-se até os dias atuais uma das melhores ferramentas de versionamento do mundo.

GIT (Ferramenta) x GitHub (Plataforma)

- O Git é um sistema de controle de versão de arquivos bastante utilizado no mundo open-source.
- Ele gerencia as alterações feitas e guarda um histórico.
- Existem sites que provêm hospedagem gratuita de código fonte para repositório Git, um deles é o GitHub.
- O GitHub é uma plataforma onde você pode armazenar seus projetos.
- É como se fosse uma rede social, só que de códigos, onde seus desenvolvedores podem disponibilizá-los para outras pessoas verem.
- O GitHub só suporta o Git, então para você subir seus projetos deve utilizá-lo, mas a integração entre eles é bem fácil.

Pré-requisitos para Começar

Antes de efetuar a instalação do Git, é necessário dois conceitos básicos para sua utilização:

- Um pouco de Lógica de Programação
 - Esse requisito não necessariamente significa que você precisa somente compartilhar códigos em seu repositório, mas é importante ter ao menos a lógica caso você tenha que implementar projetos utilizando o Git.
- Noção ou o básico sobre Comandos Linux
 - Para a execução de algumas tarefas, utilizaremos o Git Bash, que tem como base, o Bash do Linux. Portanto, para navegação no repositório local sem precisar estar a todo o momento utilizando do Gestor de Arquivos do S.O, utilizaremos alguns comandos Linux nessa oficina.

Uma breve explicação de conceitos

- Sobre controle de versão, basicamente, é um sistema de monitoramento de modificações em arquivos de determinado projeto.
- A principal utilidade do versionamento é para saber quem ou o que fez no projeto, ou o que está fazendo, que é visível pelas modificações dos arquivos em um repositório, é ter a possibilidade de restaurar determinada versão e também organizar o projeto.

Uma breve explicação de conceitos

- Repositório

- Os repositórios são os ambientes criados para armazenar seus códigos.
- Você pode possuir um ou mais repositórios, públicos ou privados, locais ou remotos, e eles podem armazenar não somente os próprios códigos a serem modificados, mas também imagens, áudios, arquivos e outros elementos relacionados ao seu projeto.
- É através dos seus repositórios públicos que outros programadores poderão ter acesso aos seus códigos no GitHub, podendo, inclusive, cloná-los para adicionar.

Uma breve explicação de conceitos

- Snapshots

- A ideia principal do GIT é tirar uma "foto" do projeto a cada nova alteração, chamado de snapshot, que no caso é uma foto atual do projeto, assim mantendo um histórico do desenvolvimento.

- Branches

- Em geral, um branch de desenvolvimento é uma bifurcação do estado do código que cria um novo caminho para a evolução do mesmo.
- Ele pode estar em paralelo com outras Git Branches que você pode gerar. Como você pode ver, é possível incorporar novas funcionalidades para nosso código de um jeito ordenado e preciso.

Uma breve explicação de conceitos

- Fork

- Quando um profissional desenvolvedor precisa começar a trabalhar em um projeto, seu primeiro passo é copiar este repositório para a sua máquina.
- Quando queremos salvar esse repositório de forma local, utilizaremos o recurso de Clone do repositório, que seria basicamente um fork.
- O fork também é uma funcionalidade útil quando um membro da equipe precisa pegar um código público para manuseá-lo em um editor de código local ou interno.

Instalação do GIT em diferentes S.O's

- Windows

- Disponível através do site oficial do projeto (<https://gitforwindows.org>) na qual a instalação se resume em Next > Next > Finish. Atualmente sua versão mais recente é a 2.38.1, que está incluso os utilitários de Git GUI (Interface gráfica) e o famigerado Git Bash.

- Linux

- Para o S.O do pinguim, a instalação se baseia na distribuição (distro) que você escolheu para o uso próprio. Pode ser instalado através de uma Store própria da distro (Gnome Store, por exemplo), ou geralmente varia de distro para distro na instalação via Terminal Emulator (A mais comum em distribuições Linux).

Instalação do GIT em diferentes S.O's

- Linux

- Distros baseadas em Debian (Ex: Ubuntu, Linux Mint, ZorinOS, etc):

`sudo apt install git`

- Distros baseadas em Fedora (Ex: Montana, Fedora, etc):

`yum install git-core`

- Distros baseadas no Arch Linux (Ex: Manjaro, EndeavourOS, etc):

`sudo pacman -S git`

Instalação do GIT em diferentes S.O's

- MacOS

- Devido a algumas restrições impostas pelo sistema da Apple, é possível instalar o Git pelo Homebrew (Um gerenciador de softwares open-source do MacOS) com o seguinte comando:

```
brew install git
```

- Porém, existe uma maneira (não-oficial) de instalação em uma interface gráfica através do projeto Git-OSX-Installer, disponível gratuitamente no SourceForge (<https://sourceforge.net/projects/git-osx-installer/>).

Comandos Linux mais utilizados

Para um uso cotidiano do Git Bash, saber sobre esses comandos é essencial:

- `ls` -> Lista todos os arquivos do diretório;
- `cd` -> Acessa uma determinada pasta (diretório);
- `rm` -> Remove um arquivo/diretório;
- `mv` -> Move ou renomeia arquivos ou diretórios;
- `echo` -> É usado para mover alguns dados para um arquivo;
- `touch` -> Permite criar novos arquivos em branco
- `nano` -> Editor de texto screen-oriented

Comandos Iniciais – Configurações Básicas

Após a instalação do Git, é necessário configurar algumas coisas como **Nome e Email** que aparecerão ao fazer alterações em arquivos no repositório.

- Configurando Identidade

- Quando o Git vai guardar alterações feitas no repositório ele guarda o **Nome de Usuário e Email** de quem alterou.

```
git config --global user.name "Seu-Username"  
git config --global user.email "Seu-Email"
```

- Configurando Editor de Texto

- Algumas comandos do Git necessitam de inserção de mensagens, e por padrão o git abre um editor de texto para que o usuário insira a mensagem. É possível configurar qual editor de texto o Git irá abrir.

```
git config --global core.editor nano
```

Comandos GIT – Iniciando um Repositório

Existem dois modos de inicializar um repositório Git. O primeiro é inicializar um repositório vazio com Git Init e o segundo modo é clonando um repositório já existente com Git Clone.

- **Git Init**

- Dentro de uma pasta que pode ou não conter um projeto, inicialize o Git com:

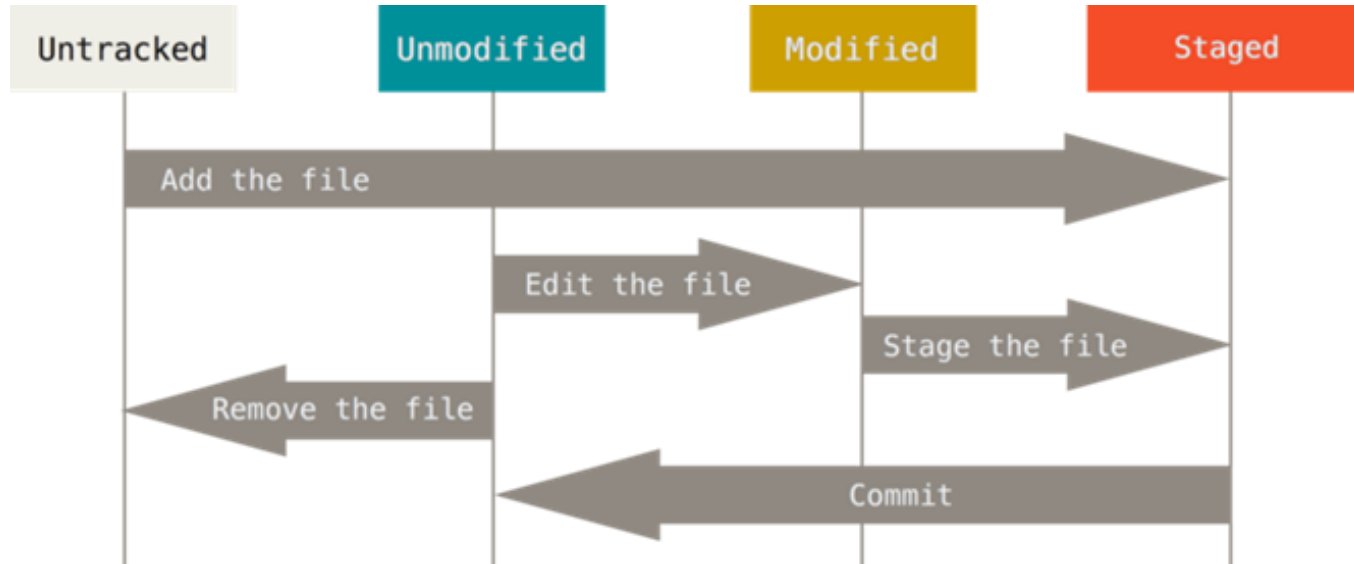
```
mkdir snct_git && cd snct_git  
git init
```

- **Git Clone**

- Para obter um repositório Git que já existe remotamente, utilize o mecanismo de clone no qual o usuário faz o download do projeto com todo o monitoramento do Git existente:

```
git clone <URL-repositório> -b <branch> <diretório>
```

Ciclos de um Arquivo/Modificações no GIT



Comandos GIT – Status de um Arquivo

Ao longo do desenvolvimento do projeto, é necessário saber qual o estado atual de um determinado arquivo. Por exemplo, se o arquivo está sendo monitorado, se o arquivo foi modificado ou se existe arquivos na área de transição.

O comando Git Status mostra se existe arquivos não monitorados, modificados ou em transição.

- Exemplo:

```
touch {a..d}; git add a b c d; git status  
git commit -m "adiciona arquivos a, b, c, d"  
echo "SNCT" >> a ; echo "Rodrigo" >> b ; echo "Erik" >> c ; git add a b c  
rm -rf d; git add d; git commit -m "remove arquivo d"; git status
```

Comandos GIT – Git Add

O comando a seguir possui duas principais funções, a de começar a monitorar arquivos e a de adicionar arquivos na área de transição.

- Monitorando novos arquivos

```
touch nomes; git status  
git add nomes; git status  
git commit -m "adicionado arquivo nomes"; git status
```

- Adicionando arquivos na área de transição

```
echo "Maria" >> nomes; git status  
git add nomes; git status  
git commit -m "insere 'Maria' em nomes"; git status
```

Comandos GIT – Opções Git Add

- -A ou --all
 - Adiciona todos os arquivos não monitorados ou modificados na área de transição, é equivalente a `git add` . Não é recomendado adicionar todos os arquivos de uma única vez, pois pode adicionar arquivos binários ou temporários ao repositório. Exemplo:
`git add -A`
`git add --all`
- -F ou --force
 - Força a adição de um arquivo não monitorado a área de transição. Exemplo:
`git add -F`
`git add --force`

Comandos GIT – Git Ignore

Ao longo do desenvolvimento de um projeto é normal surgir alguns arquivos temporários que não serão utilizados.

O comando git ignore, como o próprio nome sugere, ignora os arquivos temporários ou desnecessários que não serão adicionados no repositório remoto após o acréscimo do comando.

- Exemplo:

```
touch .gitignore && ls  
echo "*.tmp" >> .gitignore && git add .gitignore  
git commit -m "ignora arquivos .tmp"  
touch ignore_me.tmp && git add ignore_me.tmp && git status
```

Comandos GIT – Git Commit

Consolida as alterações nos arquivos que estão na área de transição.

- Commitando com editor

```
echo "João" >> nomes && git add nomes
git commit
```
- Commitando pela linha de comando

```
echo "Eduardo" >> nomes ; git add nomes
git commit -m "adicionando Eduardo a nomes"
```

Comandos GIT – Removendo/Movendo

- Removendo arquivos do repositório
 - O comando `git rm <arquivo>` remove um arquivo do repositório e do projeto. Exemplo:

```
git rm a  
git status
```
- Movendo arquivos do repositório
 - O comando `mv <arquivo> <destino>` pode causar algumas inconsistências no repositório, devido a isso, o comando `git mv <arquivo> <destino>` move o arquivo e garante que não haja nenhuma inconsistência. Exemplo:

```
git mv b a  
git commit -a -m "movi 'b' para o 'a'"  
git status
```

Comandos GIT – Histórico de Alterações

A grande motivação de se utilizar softwares de controle de versão é manter um histórico das modificações feitas no projeto. O Git oferece algumas dessas motivações, como por exemplo, poder olhar o histórico de alterações e restaurar determinadas modificações.

- **Git Log**

- O log do git consiste em modificações dos commits. Para representar um log simplificado, nós utilizamos o comando:

```
git log
```

OBS: Para sair do log, pressione a tecla Q

- **Git Log Simplificado**

- Uma forma interessante, ou peculiar, é utilizar a versão simplificada, que não possui tantas informações quanto a versão padrão do comando

```
git log --oneline
```

Comandos GIT – Desfazendo Alterações

A ideia do controle de versão é ter a possibilidade de saber quais alterações foram feitas e, se necessário, retornar a uma versão anterior do projeto. O git fornece algumas ferramentas para desfazer de maneira eficiente alterações passadas.

Para desfazer todas as alterações feitas se utiliza o comando `git checkout -- <arquivo>`.

- Exemplo:

```
echo "Pedro" >> nomes && git status  
git checkout -- nomes
```


Comandos GIT – Atualizando o último commit

Ao longo de desenvolvimento de um projeto versionado com o Git, é comum realizar um commit e esquecer de adicionar um arquivo ou errar a mensagem de commit. Para estes problemas existe o comando `git commit -amend` que, dependendo do contexto, adiciona arquivos a um commit ou altera a mensagem do commit.

- Modificando a mensagem do último commit

```
echo "ifs.snct" >> nomes ; git add nomes  
git commit -m "adiciona ."; git log --oneline  
git commit --amend -m "adiciona login ifs.snct"
```

- Adicionando um arquivo no último commit

```
touch login ; echo "ifs.snct" >> login  
git add login  
git commit --amend
```

Comandos GIT – Git Remote e Git Push

O Git permite a sua plena utilização sem a necessidade da plataforma, porém, para existir plenas integrações, é necessário que criemos um repositório remoto para armazenar as nossas modificações. Dessa forma, utilizaremos os seguintes comandos:

- **Git Remote Add**

- Consiste em ADICIONAR um parâmetro remoto para que exista a possibilidade de upload de modificações do repositório:

```
git remote add <parâmetro-remoto> <URL-repositório>
```

- **Git Remote Rm**

- Sua utilização consiste em REMOVER um parâmetro remoto já adicionado no qual esteja desatualizado ou incorreto:

```
git remote rm <parâmetro-remoto>
```

Comandos GIT – Git Remote e Git Push

Através do parâmetro remoto já adicionado, podemos fazer o PUSH dessas alterações, ou seja, o upload do projeto para o repositório remoto utilizando o seguinte comando:

- **Git Push**
 - Consiste em EMPURRAR, ou fazer UPLOAD do projeto de forma integralizada, através do Parâmetro remoto e da Branch que você selecionar:
`git push <parâmetro-remoto> <branch>`

Comandos GIT – Clonando Repositório Remoto

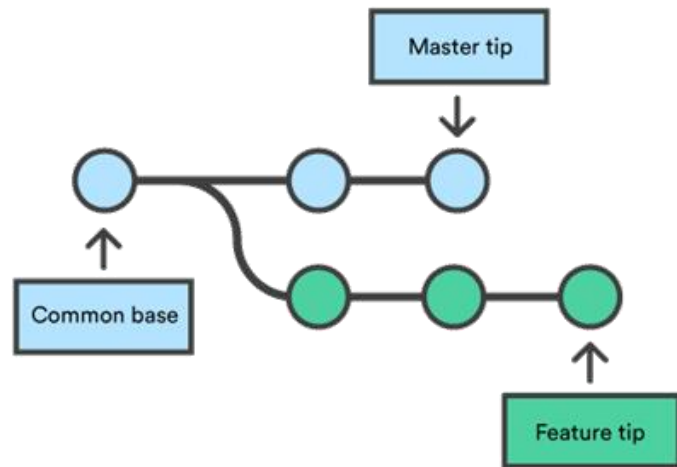
Quando um repositório remoto já se encontra no GitHub, é possível fazer o Download do mesmo, podendo ser feito através do próprio repositório disponível na plataforma, ou, nesse caso, criando um clone daquele repositório em sua máquina local. Para isso utilizaremos:

- **Git Clone**
 - Consiste em criar uma cópia do repositório remoto para criação de melhorias para serem adicionadas posteriormente:

```
git clone <URL-repositório> -b <branch> <diretório>
```

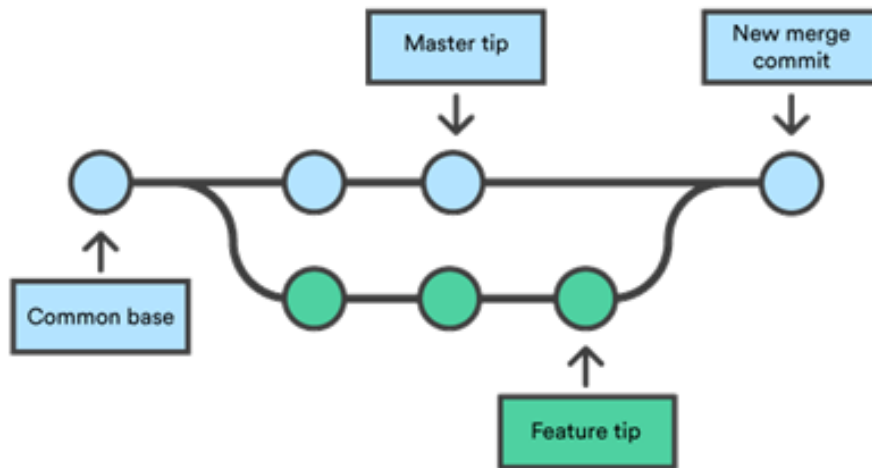
Ramificações

Ao longo de desenvolvimento de um projeto, em um repositório, a definição de ramificações tem o significado de visões diferentes do projeto. Que permitem que várias pessoas contribuam para um mesmo projeto, sem um interferir no trabalho e no progresso de outro membro da equipe.



Unindo duas Ramificações

Após trabalhar em uma ramificação secundária (development), é necessário mesclar com a ramificação principal (master ou main) do projeto.



Comandos GIT – Git Branch/Checkout e Git Merge

No acréscimo de novos recursos em uma ramificação principal (master ou main), é necessário o desenvolvimento em uma ramificação secundária (development) na qual será utilizada para mesclagem desses novos recursos, da branch secundária para a branch principal.

- Criando uma nova branch

```
git branch <nome_branch>  
git checkout -b <nome_branch>
```

- Mudando para uma branch criada

```
git checkout <nome_branch>
```

- Renomeando a branch atual

```
git branch -M <nome_requerido>
```

- Mesclando uma branch

- Para unir duas branches, mude para branch destinatária e utilize:

```
git merge <nome_branch>
```

Comandos GIT – Gerenciando Ramificações

- Listando Branchs

- Para listar todas as branchs locais e remotas utilize o comando
`git branch -a` ou `git branch --all`

- Deletando Branchs

- Existem duas formas de deletar uma branch. Podemos deletar com o comando `git branch -d <nome da branch>` que deleta a branch se, e somente se, o conteúdo da branch já estiver unido com outra branch.
- Caso contrário o Git não permitirá remover. O outro comando para remover uma ramificação é o `git branch -D <nome da branch>` que remove a ramificação independentemente do conteúdo ser unido a outra branch ou não.

Comandos GIT – Boas práticas com Ramificações

Existem algumas boas práticas ao utilizar branches como:

- **Duração da Branch**
 - O ideal é ter ramificações de curta duração em relação a master. Ter uma branch que está a muito tempo sem atualização em relação a branch principal, um eventual merge das duas será mais complexo e corre risco de haver bugs que podem prejudicar o progresso.
- **Branch principal estável**
 - É recomendado que a branch principal, master, esteja sempre funcional, ou seja, tudo que estiver na master deve estar funcionando. Muitos projetos adotam o padrão de ter a master com o progresso do projeto que já está funcionando e uma branch chamada development que contém o conteúdo em desenvolvimento e que não está 100%.

Comandos GIT – Boas práticas com Ramificações

Também é de extrema importância contar com a ausência de Branchs mortas no repositório:

- Branch Mortas

- São branchs que o conteúdo dela já foi unido com outra branch e não terá mais modificações novas, logo, a branch fica no repositório sem ser utilizada. É recomendado remover branchs que não terão mais utilidades no projeto.

Comandos GIT – Resolvendo Merge Conflict

Eventualmente, ao executar merge de duas branches poderá acontecer um conflito, que o Git não irá conseguir executar corretamente a mesclagem e você terá que fazer isso manualmente.

- Merge manual
 - Para cada arquivo que o Git não conseguir mesclar ele coloca as duas versões do arquivo separados por <<<< indicando alteração da branch atual, ==== separando as duas versões e >>>> indicando o término da versão da branch que foi tentado o merge.

Comandos GIT – Git Fetch e Git Cherry-pick

Em resolução de conflitos de Merge, eles podem ser corrigidos apenas utilizando os commits específicos da Branch development, que resulta em boas práticas, utilizando dois comandos:

- **Git Fetch**
 - Consiste em ADICIONAR uma referência para uma branch diferente da principal (main) para que possa utilizar commits dessa outra branch:
`git fetch <URL-repositório> <branch-development>`
- **Git Cherry-pick**
 - Sua utilização consiste em CLONAR os commits (modificações) já existentes na branch development para a branch principal (main):
`git cherry-pick <commit>`
`git cherry-pick <commit..commit>`

Comandos GIT – Git Revert e Git Reset

Para reverter um commit (alteração) já consolidado ou simplesmente apagar uma sequência de commits é necessário utilizar:

- **Git Revert**
 - O comando `git revert <commit>` reverte alterações de um commit, criando um novo commit, logo ele desfaz modificações com um novo commit e mantém o histórico de commits anteriores.
- **Git Reset**
 - O comando `git reset <commit>` ‘descarta’ todos os commits até o commit especificado. Portanto ele move o ponteiro do commit atual para o commit especificado, sem gerar um commit, assim perdendo a relação de commits entre o atual e o especificado.

Recapitulando...

Os principais comandos Git mais utilizados, e o significado deles:

- **Git Init**
 - Este comando dá origem a um repositório novo, local ou remoto, ou reinicializa um repositório já existente;
- **Git Clone**
 - Este comando clona o código de um repositório para sua manipulação em outro ambiente;
- **Git Add**
 - Este comando adiciona um arquivo alterado a uma área de transição, ou seja, o prepara para ser vinculado a um commit;
- **Git Commit**
 - Este comando move os arquivos da área de transição para um repositório local;

Recapitulando...

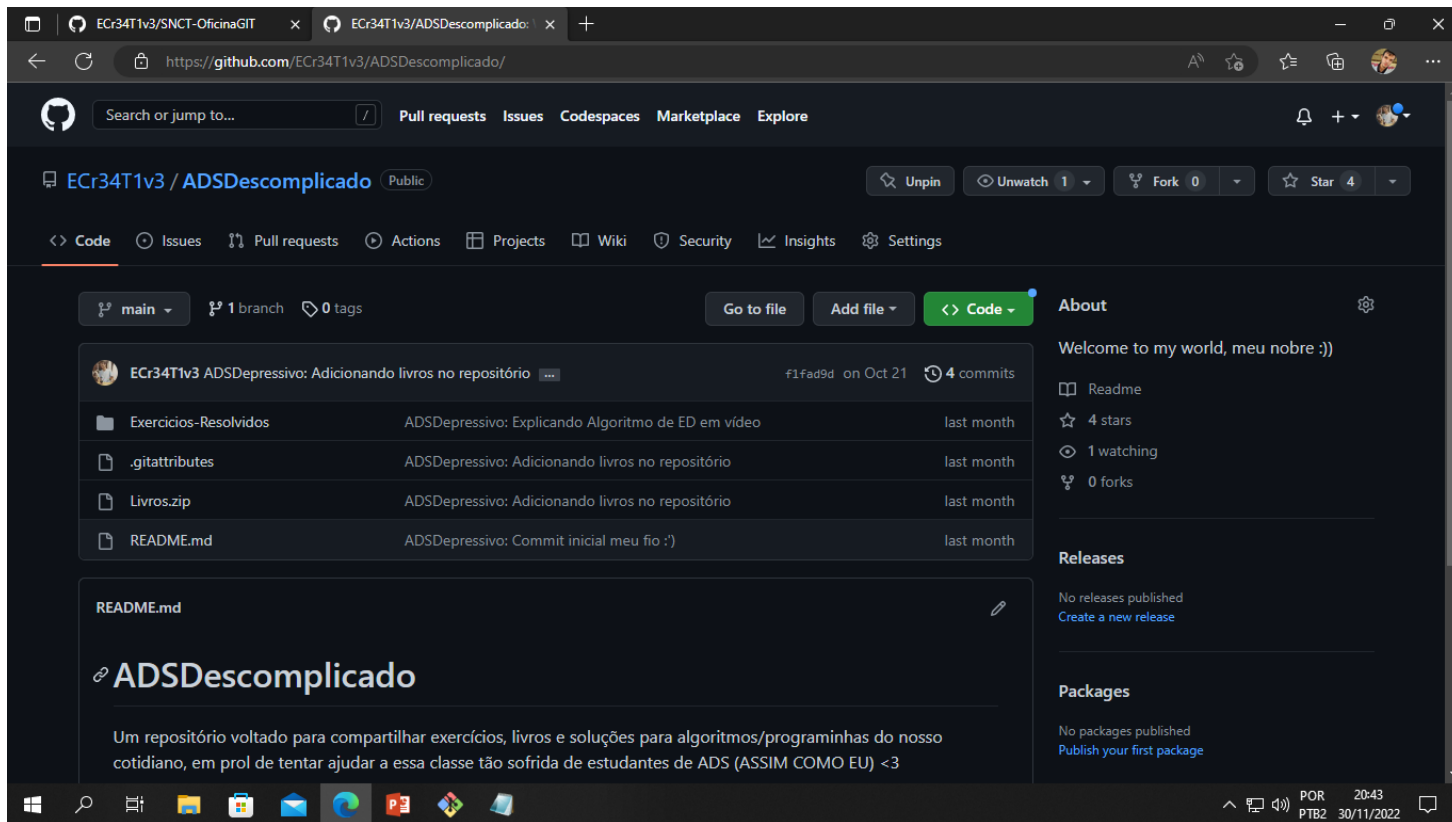
- Git Log
 - Este comando permite a visualização do histórico de commits de um arquivo ou usuário, ou o acesso de uma versão específica;
- Git Remote
 - Este comando permite a adição ou remoção de um parâmetro remoto junto de uma URL para upload de arquivos para um repositório remoto;
- Git Push
 - Este comando envia arquivos de um repositório local para um repositório remoto. No GitHub, por exemplo;
- Git Merge
 - Este comando serve para unir arquivos alterados ao arquivo original de um projeto. Em outras palavras, é ele quem une os branches as commits.



Fork e Pull-Request

Vídeo-explicativo

Projeto ADSDescomplicado



The screenshot shows the GitHub repository page for **ECr34T1v3 / ADSDescomplicado**. The repository is public and has 4 stars, 1 watcher, and 0 forks. The main branch is **main** with 1 branch and 0 tags. The repository contains the following files:

File	Commit Message	Commit Date
Exercicios-Resolvidos	ADSDepressivo: Explicando Algoritmo de ED em vídeo	last month
.gitattributes	ADSDepressivo: Adicionando livros no repositório	last month
Livros.zip	ADSDepressivo: Adicionando livros no repositório	last month
README.md	ADSDepressivo: Commit inicial meu fio :)	last month

The **README.md** file content is as follows:

ADSDescomplicado

Um repositório voltado para compartilhar exercícios, livros e soluções para algoritmos/programinhas do nosso cotidiano, em prol de tentar ajudar a essa classe tão sofrida de estudantes de ADS (ASSIM COMO EU) <3

The right sidebar shows the **About** section with a welcome message, **Readme**, **4 stars**, **1 watching**, and **0 forks**. The **Releases** section shows "No releases published" and a link to "Create a new release". The **Packages** section shows "No packages published" and a link to "Publish your first package".

Desafios Práticos

- O primeiro desafio consiste em utilizar o git clone no repositório que acabamos de criar (SNCT-Git) usando a branch principal (master ou main). Em seguida, crie uma nova branch, com o nome “development” (`git checkout -b development`) e faremos um `git push development`. Logo após, faça uma alteração simples no arquivo nomes, adicionando algum nome qualquer, adicione essa modificação com o `git add nomes`, e em seguida faça o `git commit -m “alterando nomes”`. Utilize o `git push development` (já com o commit novo consolidado), volte para a branch principal usando `git checkout master` (ou main) e utilize o `git fetch <url-do-repositório> development` (para referenciar a branch development) e o `git cherry-pick <commit>` para consolidar o commit da branch development para a branch principal. Após as mudanças, aplique com o `git push master` (ou main).

Desafios Práticos

- O último desafio prático consiste em utilizar o recurso de **fork**, clonando assim o repositório do ADSDescomplicado e aplique um `git clone` referenciando a cópia (fork) já disponível na sua conta do GitHub. Em seguida, adicione exercícios já feitos na pasta “Exercícios-Resolvidos” do repositório, utilizando `mkdir` (se necessário) para criar uma pasta com o nome da disciplina e movendo os exercícios para lá (já com o clone no seu computador). Adicione esses arquivos com o `git add` e em seguida, execute o `git commit` para consolidar as alterações. Utilize o `git push` para enviar ao seu repositório fork, e em seguida, mande um **PR (Pull-request)** com as modificações para o repositório original do ADSDescomplicado. <3

Referências

- CHACON, Scott. Pro Git. Everything you need to know about Git. 2nd. Edição. Ed.: Apress. 9 de novembro, 2014. Disponível gratuitamente em: <https://git-scm.com/book/en/v2>
- Fábio Akita (Entendendo Git): <https://www.youtube.com/watch?v=6Czd1Yetaac>
- Guia Git-SCM: <https://git-scm.com/docs/>
- Terminal ROOT (Git para Iniciantes): <https://terminalroot.com.br/git/>
- GitHub Folha de Dicas: https://training.github.com/downloads/pt_BR/github-git-cheat-sheet.pdf
- Kenzie (O que é Git?): <https://kenzie.com.br/blog/o-que-e-git/>
- Curso-R (Abandonando o termo 'master'): <https://blog.curso-r.com/posts/2020-07-27-github-main-branch/>
- TreinaWeb (Git e GitHub: Quais as diferenças?): <https://www.treinaweb.com.br/blog/git-e-github-quais-as-diferencas>

Agradecimentos Finais

- Ao IFS – Campus Aracaju
 - Prof^a Msc^o Jislane Menezes (Coordenadora – COINF)
 - Prof^a Msc^o Clênia Melo
 - Prof^a Msc^o Cristiane Oliveira
 - Prof^a Msc^o Renata Moraes
- A Comunidade Builders BR (Mantenedores Brasileiros)
 - Mickael Mendes (AOSPA Dev - <https://github.com/mickaelmendes50>)
 - Henrique Pereira (PE Core Member - <https://github.com/Hlcpereira>)
 - João Henrique (PE CeO Member - <https://github.com/jhenrique09>)
 - Renan Queiroz (LOS Dev - <https://github.com/RenanQueiroz>)

Agradecimentos Finais

- A Comunidade Indiana/Latinoamericana de Mantenedores (Redmi 4A/5A)
 - Carlos Arriaga (LOS Dev - <https://github.com/TheStrechh>)
 - Manuel Carmona (MK Dev - <https://github.com/manuelcarmona>)
 - Sunny Raj (LOS Dev - <https://github.com/SunnyRaj84348>)
 - Fabian Leutenegger (AOSPA Dev - <https://github.com/33bca>)
- A todos vocês presentes nessa oficina da SNCT. Meu muito obrigado <3