Residência em Software

Versionamento de Código

Professores:

Álvaro Coelho, Edgar Alexander, Esbel Valero e Hélder Almeida

INSTITUIÇÃO EXECUTORA

cepedi UESC

COORDENADORA

MCTI FUTURO Softex

APOIO

MINISTÉRIO DA CIÊNCIA, TECNOLOGIA E INOVAÇÃO

GOVERNO FEDERAL BRASIL UNIÃO E RECONSTRUÇÃO

Residência em Software

Objetivo

O objetivo deste módulo é ajudá-lo a navegar pelas complexidades do Git e do GitHub, com exemplos passo a passo e atividades práticas. Desde a compreensão dos fundamentos até o gerenciamento de ramificações e mesclagem de código, este recurso capacitará você a colaborar de forma eficaz e simplificar seu fluxo de trabalho de codificação.

Residência em Software

Introdução

No mundo tecnológico de hoje, dominar o versionamento de código com Git e GitHub tornou-se essencial para os desenvolvedores.

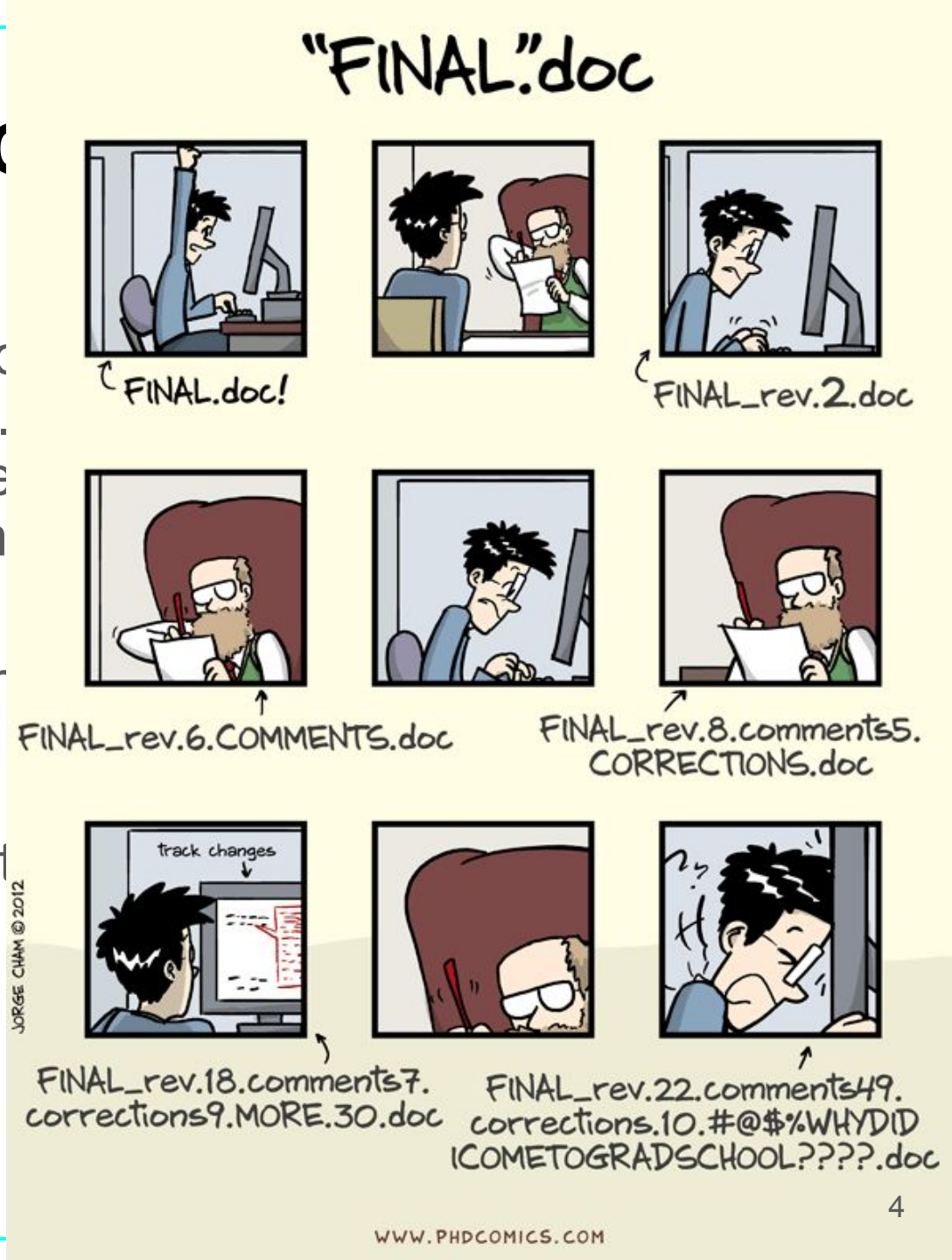
Versionamento de código é o processo de rastrear e controlar as alterações feitas em um software ao longo do tempo. Isso permite que desenvolvedores acompanhem as diferentes versões do código-fonte, possibilitando a recuperação de versões anteriores, o trabalho colaborativo entre equipes e a implementação de novas funcionalidades de forma organizada. O versionamento garante a integridade do código, facilita a resolução de conflitos, e permite aos desenvolvedores revisar e auditar mudanças, melhorando a eficiência do desenvolvimento de software e a qualidade do produto final. O Git é um exemplo popular de sistema de controle de versão distribuído utilizado amplamente na indústria.

Residência em Software

O que são Sistemas de C (SCV)?

Softwares que têm a finalidade de gerenciar c desenvolvimento de um documento qualquer. comumente utilizados no desenvolvimento de diferentes versões — histórico e desenvolvim também da documentação.

Esse tipo de sistema é muito presente em em tecnologia e desenvolvimento de software. É desenvolvimento de software livre. É útil, em projetos pessoais pequenos e simples como t comerciais.



Residência em Software

Vantagens de usar SCVs

Controle do histórico: facilidade em desfazer e possibilidade de analisar o histórico do desenvolvimento, como também facilidade no resgate de versões mais antigas e estáveis. A maioria das implementações permitem analisar as alterações com detalhes, desde a primeira versão até a última.

Trabalho em equipe: um sistema de controle de versão permite que diversas pessoas trabalhem sobre o mesmo conjunto de documentos ao mesmo tempo e minimiza o desgaste provocado por problemas com conflitos de edições. É possível que a implementação também tenha um controle sofisticado de acesso para cada usuário ou grupo de usuários.

Marcação e resgate de versões estáveis: a maioria dos sistemas permite marcar onde é que o documento estava com uma versão estável, podendo ser facilmente resgatado no futuro.

Ramificação de projeto: a maioria das implementações possibilita a divisão do projeto em várias linhas de desenvolvimento, que podem ser trabalhadas paralelamente, sem que uma interfira na outra.

Residência em Software

Vantagens de usar SCVs

Segurança: Cada software de controle de versão usa mecanismos para evitar qualquer tipo de invasão de agentes infecciosos nos arquivos. Além do mais, somente usuários com permissão poderão mexer no código.

Rastreabilidade: Com a necessidade de sabermos o local, o estado e a qualidade de um arquivo; o controle de versão traz todos esses requisitos de forma que o usuário possa se embasar do arquivo que deseja utilizar.

Organização: Alguns softwares disponibilizam uma interface visual onde podem ser vistos todos os arquivos controlados, desde a origem até o projeto por completo.

Confiança: O uso de repositórios remotos (na nuvem) ajuda a não perder arquivos por eventos inesperados. Além disso, e possível fazer novos projetos sem danificar o desenvolvimento do atual.

Residência em Software

Evolução dos Sistemas de Controle de Versão

- 1970s – SCCS (Source Code Control System)
- 1980s – RCS (Revision Control System)
- 1986 – CVS (Concurrent Versions System
- 2000 – SVN (Apache Subversion)
- 2005 – Git
- 2008 – Mercurial
- 2018 – GitHub e GitLab
- 2020 – Bitbucket

Residência em Software

Evolução dos Sistemas de Controle de Versão

A evolução dos sistemas de controle de versão é um testemunho da crescente importância do versionamento de código para o desenvolvimento de software e colaboração entre equipes de desenvolvedores. O Git, em particular, se tornou o sistema de controle de versão mais amplamente adotado devido à sua velocidade, eficiência e abordagem distribuída, impulsionando uma nova era na gestão de alterações em projetos de software.

Residência em Software

Introdução ao Git e ao GitHub

- Git é um sistema de controle de
  rastreia alterações em arquivos

- GitHub é uma plataforma basea
  repositórios Git e facilita a colab

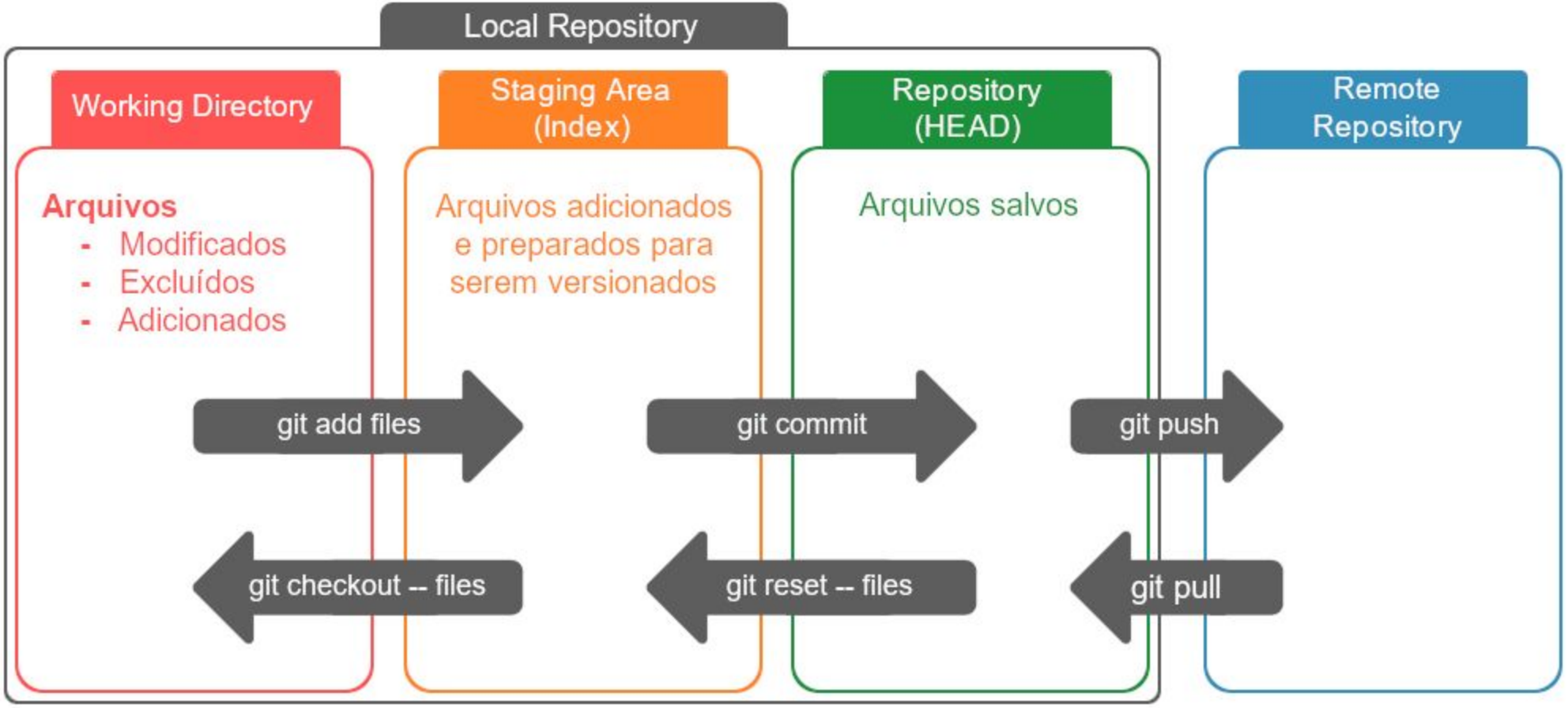
Residência em Software

Principais Características do Git

- **Velocidade:** o Git é rápido no gerenciamento de grandes projetos.
- **Descentralização:** cada colaborador possui uma cópia completa do repositório.
- **Eficiência em ramificação e mesclagem:** facilitando o trabalho em equipe.

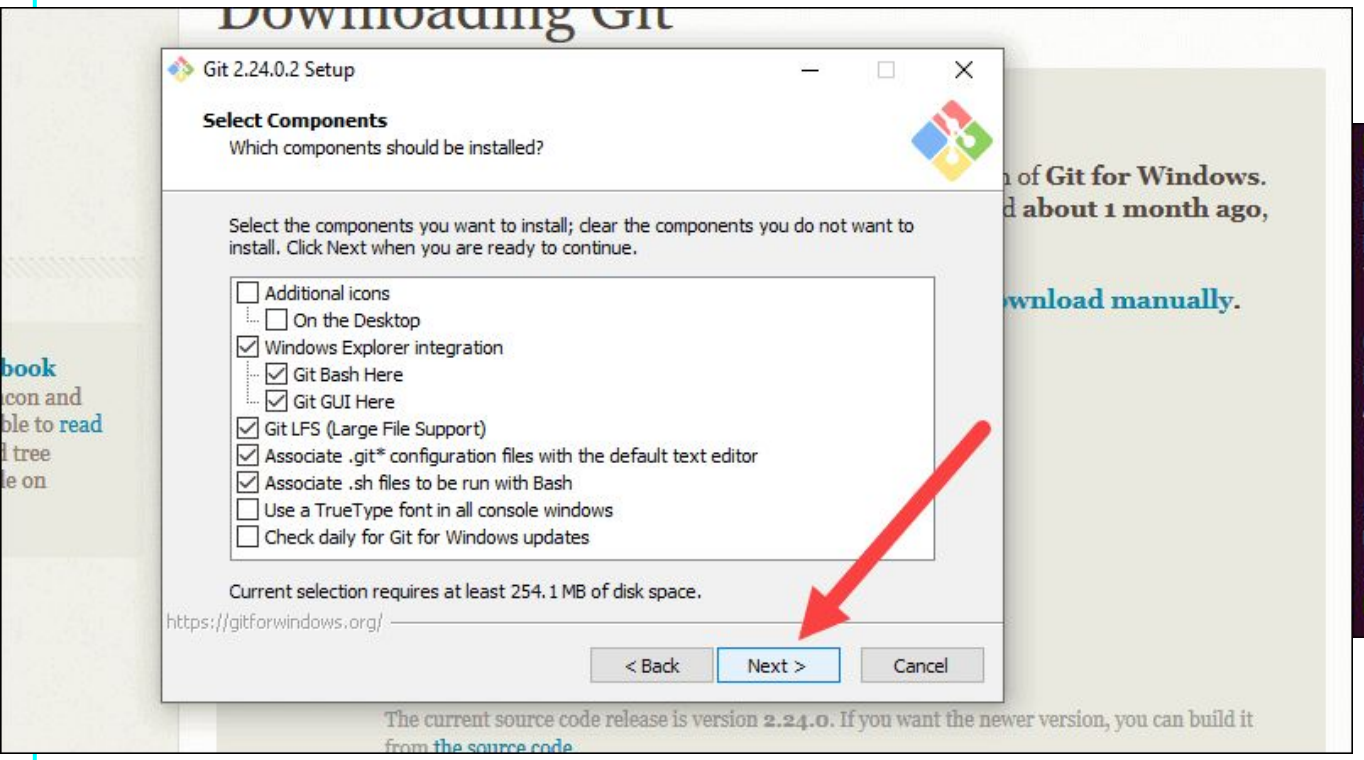
Estrutura de um Repositório do Git

Repositório local



Prática - Instalando e configurando o Git

Windows



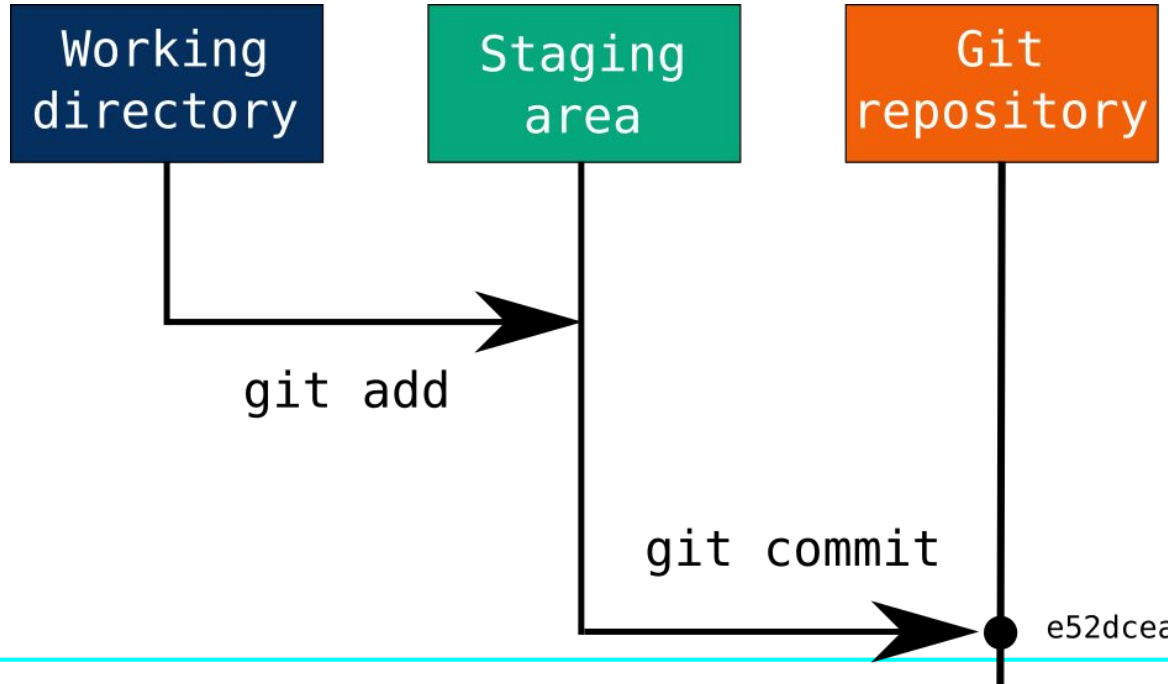
Linux

```
bosko@pnap:~$ sudo apt install git
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  git
0 upgraded, 1 newly installed, 0 to remove and 4 not upgraded.
Need to get 0 B/4,529 kB of archives.
After this operation, 36.6 MB of additional disk space will be used.
Selecting previously unselected package git.
(Reading database ... 164181 files and directories currently installed.)
Preparing to unpack .../git_1%3a2.25.1-1ubuntu3.6_amd64.deb ...
Unpacking git (1:2.25.1-1ubuntu3.6) ...
Setting up git (1:2.25.1-1ubuntu3.6) ...
bosko@pnap:~$
```

```
git config --global user.name “fulano de tal”

git config --global user.email “nick@mail.com”
```

Comandos Básicos do Git - Parte 1

- **git init:** Iniciando um novo repositório Git.
- **git add:** Adicionando alterações para serem registradas no próximo commit.
- **git commit**: Registrando alterações no repositório com uma mensagem descritiva.



Residência em Software

Stash: Salvar na área de staging

Stash é um conjunto de alterações que ficam salvas em uma pilha.

O comando **`git stash`** permite salvar as mudanças feitas e voltar para um estado limpo, ou seja, sem mudanças, do seu diretório de trabalho.

- `git stash -m “mensagem”`
- `git stash list`
- `git stash apply stash{#}`
- `git stash pop`
- `git stash drop stash{#}`

Residência em Software

Markdown - parte 1

- Linguagem de marcação, como HTML;
- O principal motivo de se criar uma outra linguagem de marcação é que, códigos HTML se tornam difíceis de ler (baixa *readability*), pois as marcas textuais como <*body*> fazem com que as palavras do texto original não possam ser compreendidas facilmente.
- No lugar de marcas textuais, as formatações se resumem a símbolos simples digitados pelo teclado que marcam partes do texto e geram resultados visuais. Atualmente, é possível usar markdown até mesmo em editores como o Microsoft Office, o LibreOffice Writer, que são compatíveis com arquivos do tipo .md ou .markdown.
- O GitHub também usa a linguagem para especificar os arquivos README, criados nas raízes dos repositórios, além de ser usada na gestão de issues e de pull requests na plataforma.
- **https://stackedit.io/app**

Residência em Software

Prática - Iniciando um Repositório

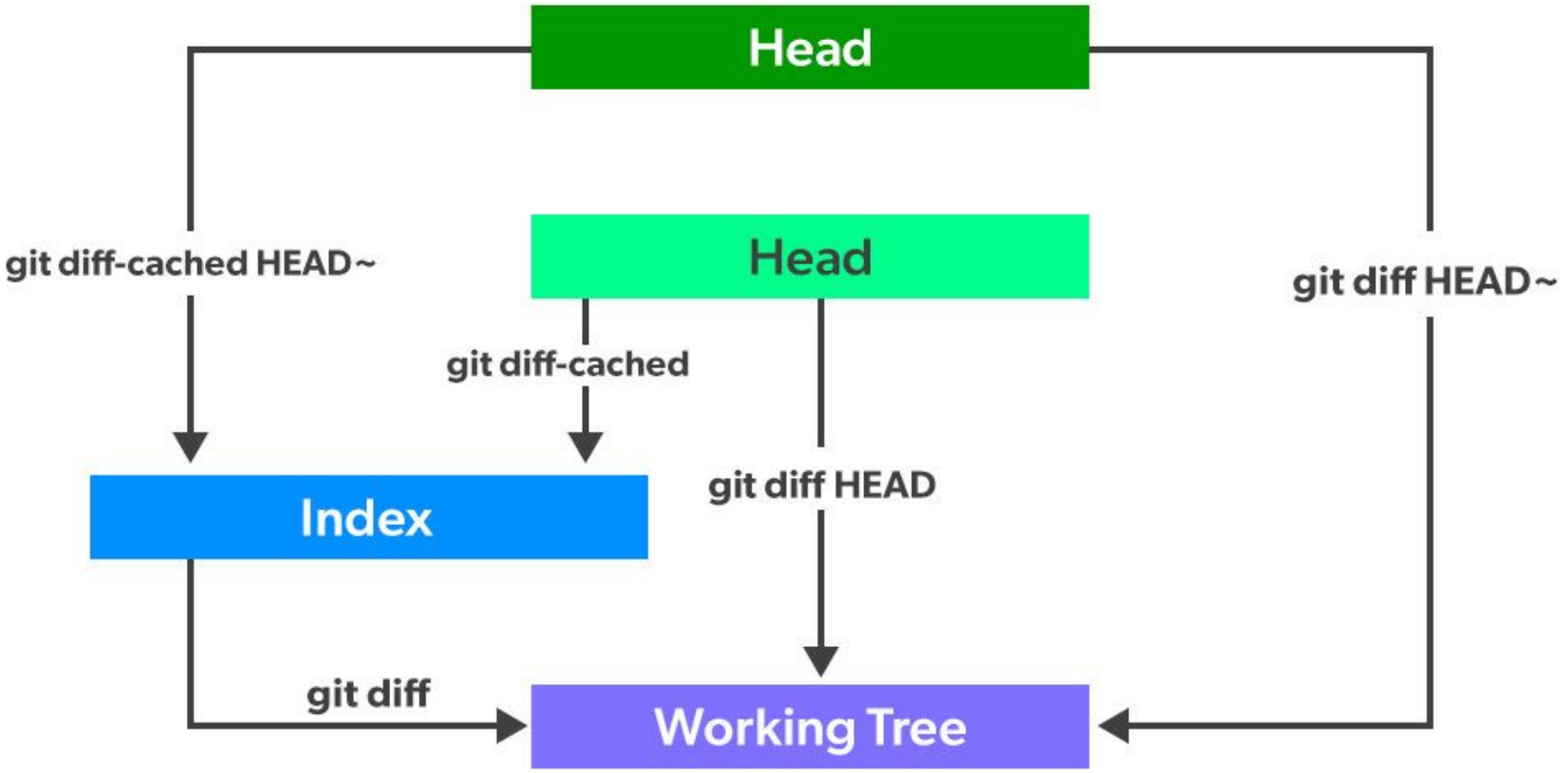
- Criar um novo repositório Git em um diretório vazio.
- Adicionar um arquivo Readme.md ao repositório.

```
# Olá mundo do Git

Olá esse é um arquivo em formato *Markdown*.
Esse é meu primeiro projeto Git dentro da **Residência
TIC18**.


## Objetivo

Conhecer os comandos básicos e algumas recomendações
para o uso do Git e do Github.
```

Residência em Software

Comandos Básicos do Git - Parte 2

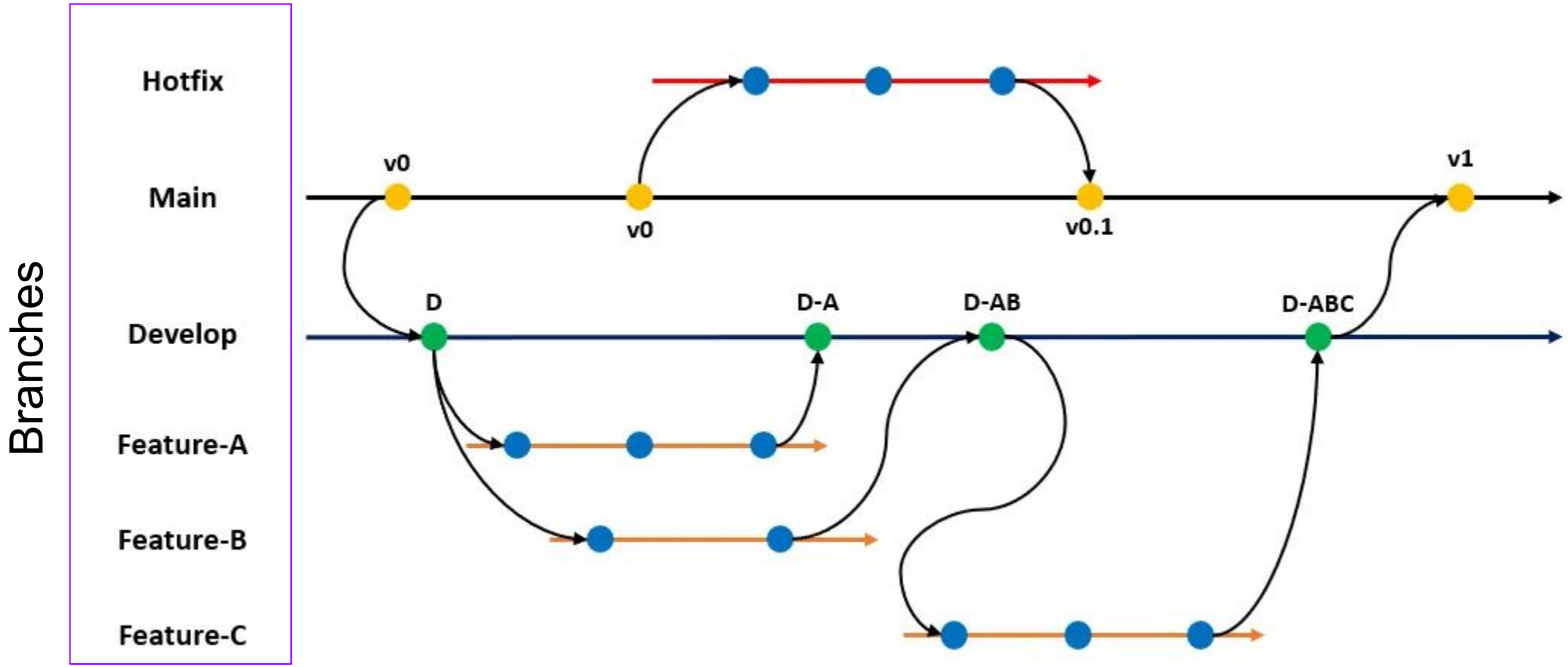
- **git status**: Verificando
- **git log**: Visualizando o
- **git diff**: Comparando al



Branching no Git

Branches ⇔ Ramificações

Head ⇔ branch atual



Residência em Software

Branching no Git

git branch: Criando, listando e deletando branches.

`git branch`	List branches (the asterisk denotes the current branch)
`git branch -a`	List all branches (local and remote)
`git branch [branch name]`	Create a new branch
`git branch -d [branch name]`	Delete a branch
`git branch -m [old branch name] [new branch name]`	Rename a local branch

git checkout: Alternando entre branches existentes.

`git checkout -b [branch name]`	Create a new branch and switch to it
`git checkout -b [branch name] origin/[branch name]`	Clone a remote branch and switch to it
`git checkout [branch name]`	Switch to a branch
`git checkout -`	Switch to the branch last checked out
`git checkout -- [file-name.txt]`	Discard changes to a file

Prática - Criando branches

- Criar um novo repositório Git em um diretório vazio.
- Adicionar um arquivo Readme.md ao repositório.

```
# Olá mundo do Git

Olá esse é um arquivo em formato *Markdown*.
Esse é meu primeiro projeto Git dentro da **Residência
TIC18**.


## Objetivo

Conhecer os comandos básicos e algumas recomendações
para o uso do Git e do Github.
```

Residência em Software

Gitignore

O gitignore é um arquivo de texto oculto que geralmente fica na raiz do repositório, utilizado para descrever ao Git tanto as pastas quanto os arquivos que a pessoa desenvolvedora deseja que sejam ignorados.

Ele pode conter desde um arquivo sensível, ou seja, que contenha AppKey, appToken ou qualquer outro tipo de senha, quanto arquivos comuns que não devem ser commitados.

Portanto, os arquivos que se encontram descritos dentro do gitgnore não serão rastreados e nem versionados, fazendo então com que não tenha chances de serem exibidos, por exemplo, dentro de um repositório remoto.

Residência em Software

.gitignore - sintaxe

O gitignore se trata de um arquivo bastante simples, sem muitos detalhes para a sua utilização, porém algumas regras são necessárias para que ele realmente possa funcionar:

- * equivale a “todos” os arquivos ou diretórios serão ignorados. Exemplo: *.svn. Nesse caso, estamos solicitando ao git ignorar todo os arquivos que possui a extensão .svn;
- / é utilizado quando desejamos ignorar um arquivo ou diretório que possui um caminho relativo. Exemplo: public/* . Nesse caso estamos solicitando que o Git ignore todos os arquivos que encontra dentro da pasta public.
- # apenas adiciona comentários dentro do arquivo .gitignore. Exemplo: Se você deseja ignorar a pasta node_modules, você pode colocar um comentário logo acima para informar isso, dessa forma:

Residência em Software

Prática com .gitignore - exemplo e teste

.gitignore

```
*.o
*.tmp
/private
/keys
```

```
→  residenciaTic18 git:(master) ✗ ls
keys  local.tmp  main.cs  main.o  private  src
→  residenciaTic18 git:(master) ✗ git check-ignore ./*
./keys
./local.tmp
./main.o
./private
→  residenciaTic18 git:(master) ✗
```

Residência em Software

Introdução aos Repositórios Remotos

Criar um repositório localmente não torna possível que outras pessoas contribuam em nosso projeto. Para que isso seja possível, precisamos hospedar nosso repositório em um servidor Git remoto.

Existem diversas plataformas de hospedagem de repositórios, como o Bitbucket, GitLab e a mais famosa que é o GitHub.

O fluxo de trabalho é: nós criamos o repositório em nossa máquina e enviamos para a hospedagem ou criamos o repositório na hospedagem e baixamos em nossa máquina.

Quando enviamos ou baixamos utilizando o Git, estamos na realidade enviando o histórico de alterações dos arquivos. Ao utilizar os comandos de enviar e receber, vamos perceber que o que está sendo baixado são os commits armazenados pelo Git.

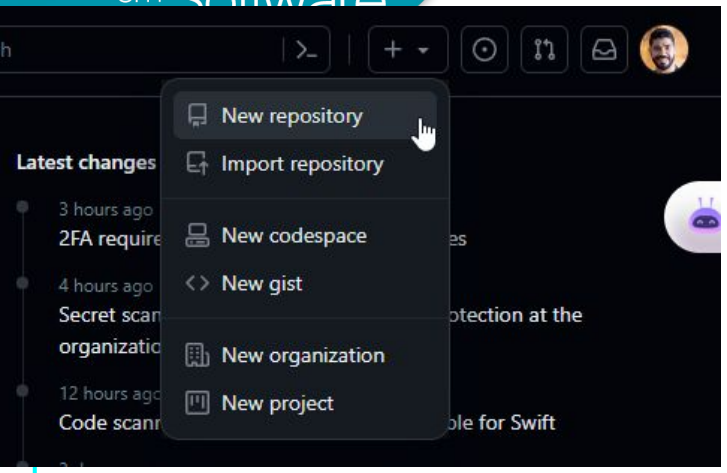
Residência em Software

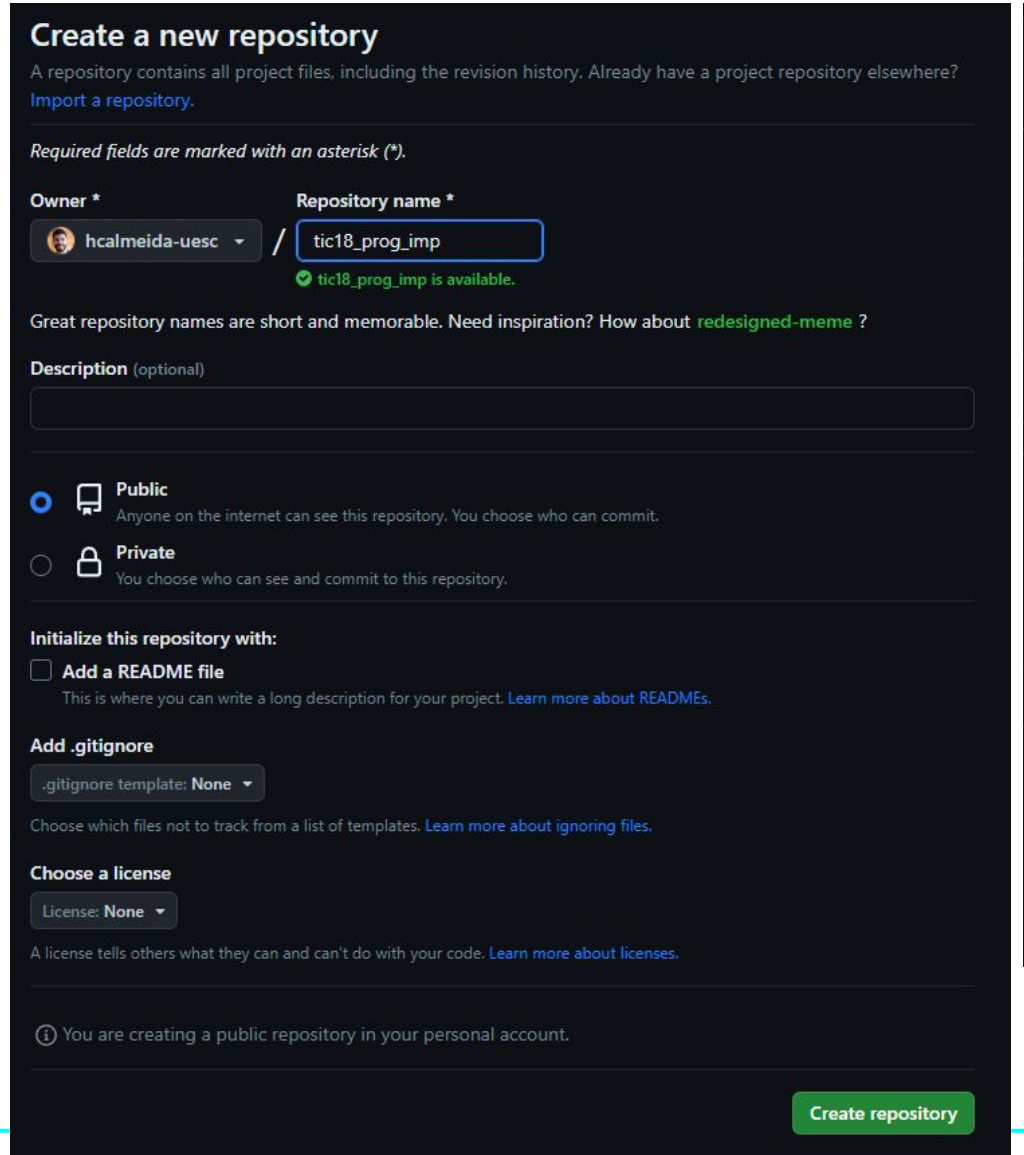
Clonando um Repositório Remoto

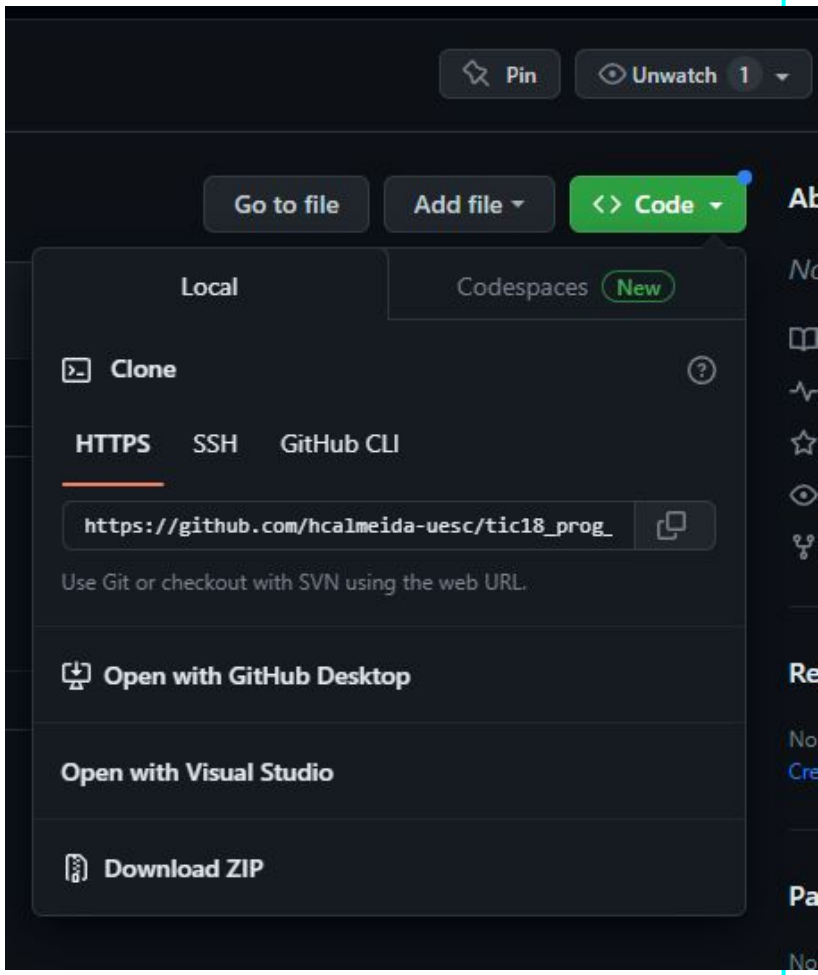
- git clone: Clonando um repositório remoto para o ambiente local.
  - `git clone url_do_repositorio`

- git remote: Gerenciando conexões com repositórios remotos.
  - `git remote -v`
  - `git remote add origin`

Criando um Repositório no GitHub







Residência em Software

Prática - repositórios remotos

1. Criar um repositório no GitHub.
2. Clonar o repositório para sua máquina.
3. Navegar para outro diretório e clonar esse outro repositório: https://github.com/hcalmeida-uesc/tic18_prog_imp.git
4. Utilizar os comandos para visualizar os links remotos nos dois repositórios

Residência em Software

Enviando Alterações para um Repositório Remoto

Para enviar nossas alterações locais para o nosso repositório remoto, utilizamos o comando

```
git push
```

Se estivermos usando o protocolo SSH cadastrado no GitHub, recomendado para máquinas particulares, não será solicitada credenciais.

Ao utilizar o protocolo HTTPS para se conectar ao repositório remoto, será necessário efetuar autenticação no GitHub em cada envio de alterações.

PS. No primeiro envio ao repositório remoto, pode ser necessário indicar qual branch enviar por padrão, para isso usar o comando:

```
git push --set-upstream origin [branch_name]
```

Residência em Software

Atualizando um Repositório Local com as Alterações do Remoto

Para atualizar o repositório local com as alterações do repositório remoto utilizamos o comando

```
git pull
```

Normalmente isso é necessário quando trabalhamos em equipe e/ou quando usamos mais de um equipamento para editar o mesmo código. Ou seja, sempre que há alteração de algo no repositório remoto e precisamos baixar essas alterações.

Residência em Software

Prática - repositórios remotos

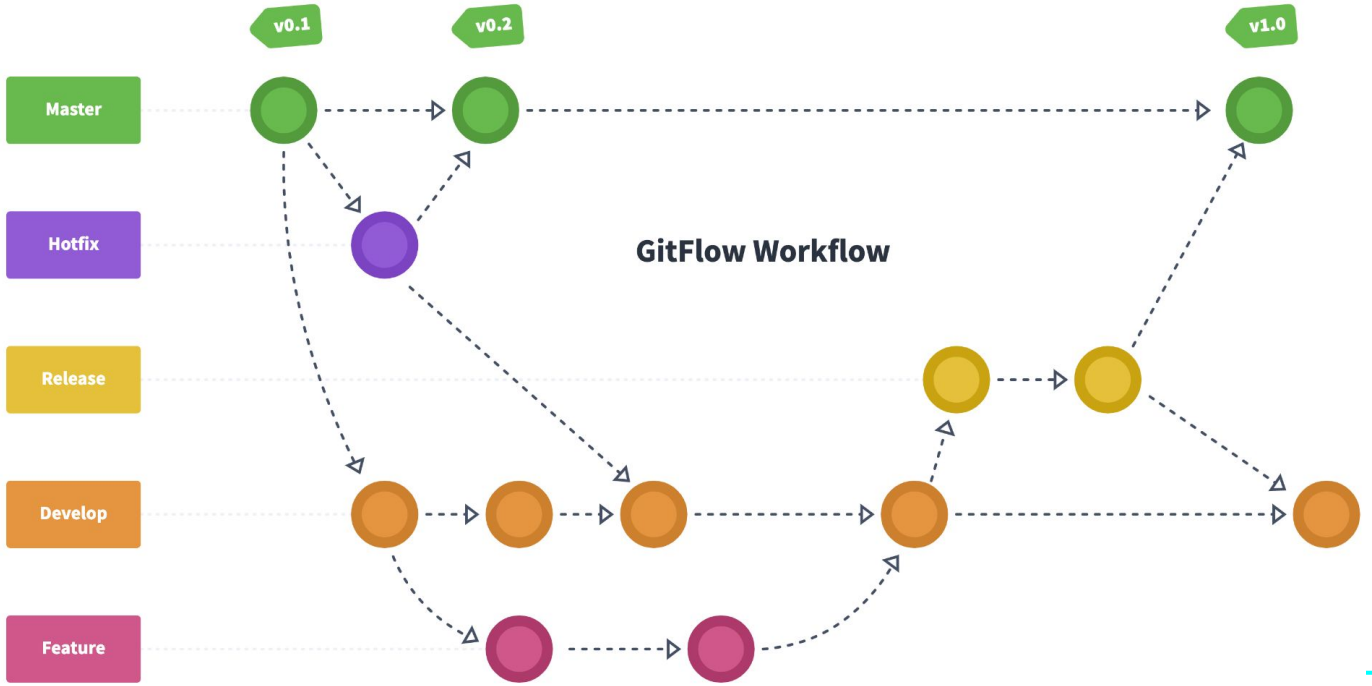
1. Crie um outro diretório e clone o mesmo repositório que você criou anteriormente;
2. Faça alterações em apenas um dos diretórios.
3. Envie as mudanças para o GitHub.
4. Vá ao outro diretório, verifique o status do repositório e, em seguida, sincronize os dados locais com o servidor.

Residência em Software

Merging no Git

Quando trabalhamos com algum tipo de fluxo de trabalho no Git, como o Git flow, por exemplo, acabamos por criar várias ramificações da branch principal e com isso precisaremos fazer a unificação quando houver a finalização das novas branches. Os comandos **`git merge`** e **`git rebase`** auxiliam nessa tarefa de juntar as ramificações.

Git flow é um modelo de fluxo de trabalho que busca simplificar e organizar o versionamento de ramificações de um projeto de desenvolvimento no Git.



Residência em Software

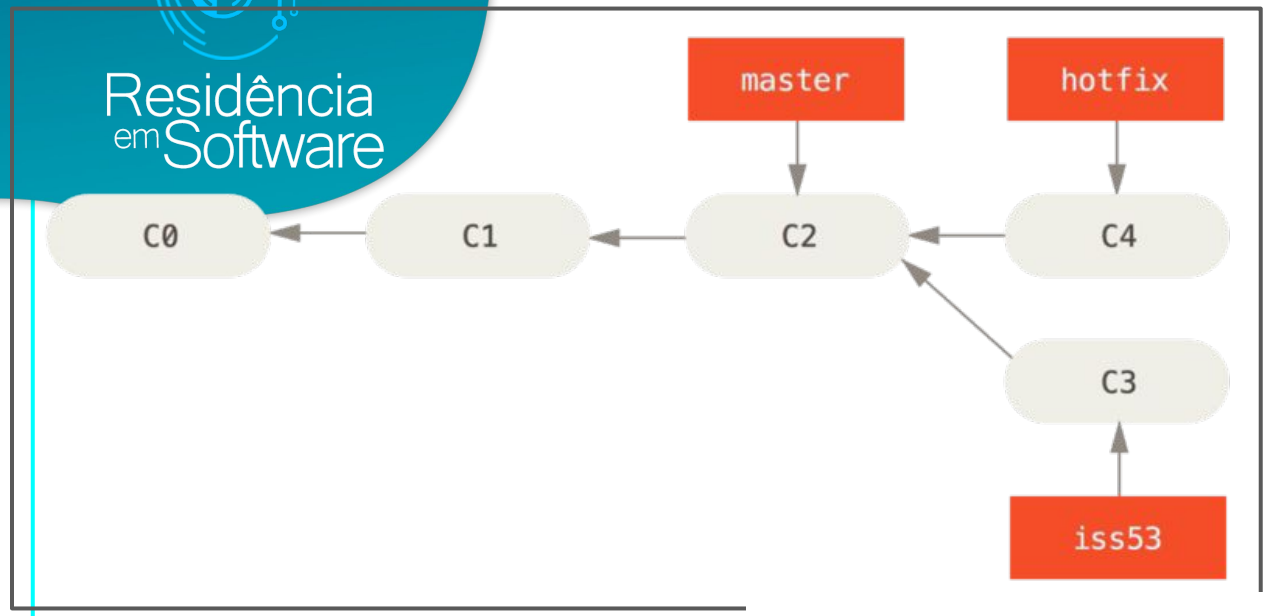
Git Merge

O Git Merge é uma abordagem mais simples e direta para integrar branches. Quando você executa o comando "`git merge [branch_alvo]`", o Git cria um novo commit de merge que combina as alterações dos branches envolvidos. Esse commit de merge possui duas ou mais linhas de pais, representando a origem das alterações.

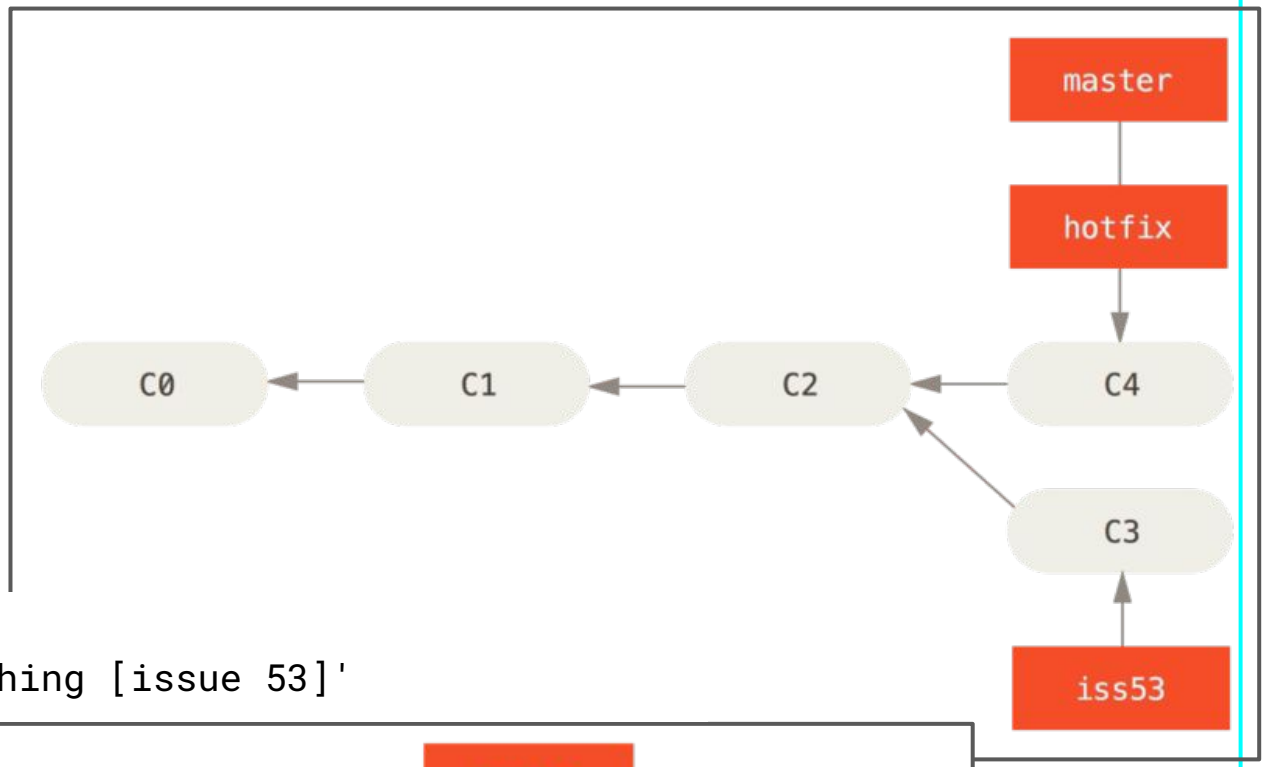
O Git Merge preserva a história de cada branch individualmente, mantendo todos os commits originais. Isso resulta em um histórico de commits mais completo e explícito.

Os commits do branch que está sendo mesclado são preservados em sua ordem original, o que pode levar a um histórico de commits com muitos commits de merge.

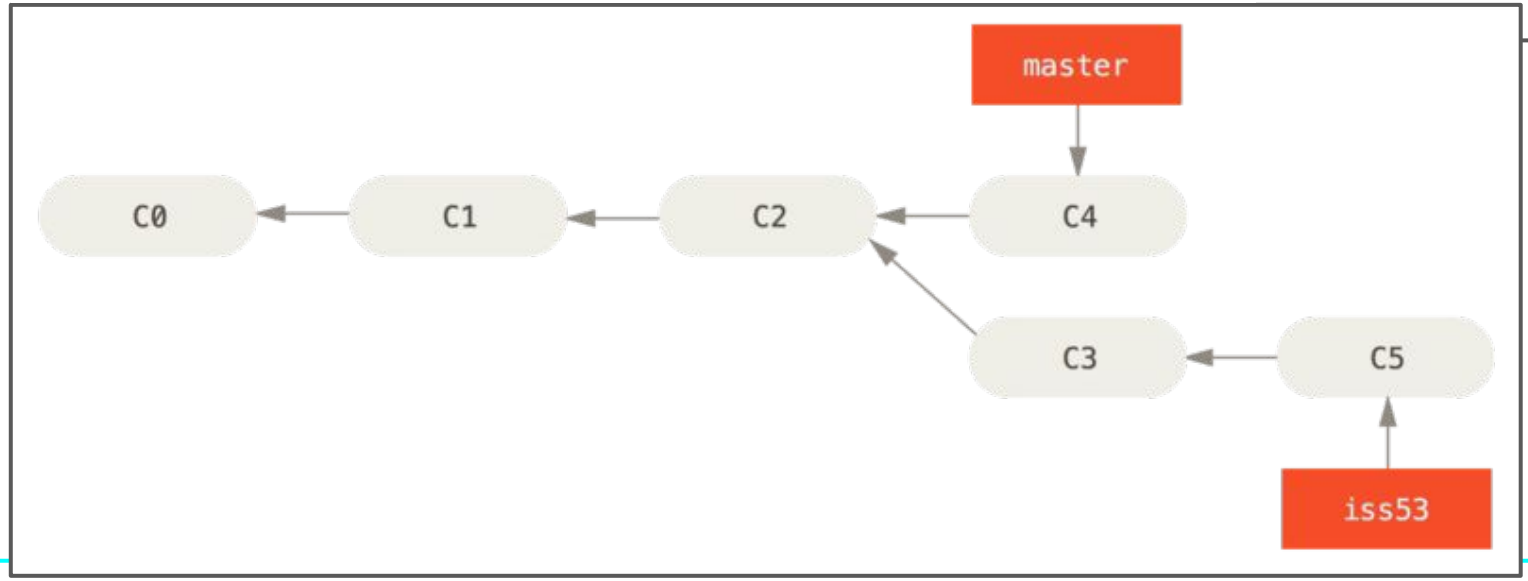
```
git checkout -b hotfix
git commit -a -m 'Fix something'
```




```
git checkout master
git merge hotfix
```




```
git checkout iss53
git commit -a -m 'Finishing [issue 53]'
```



Residência em Software

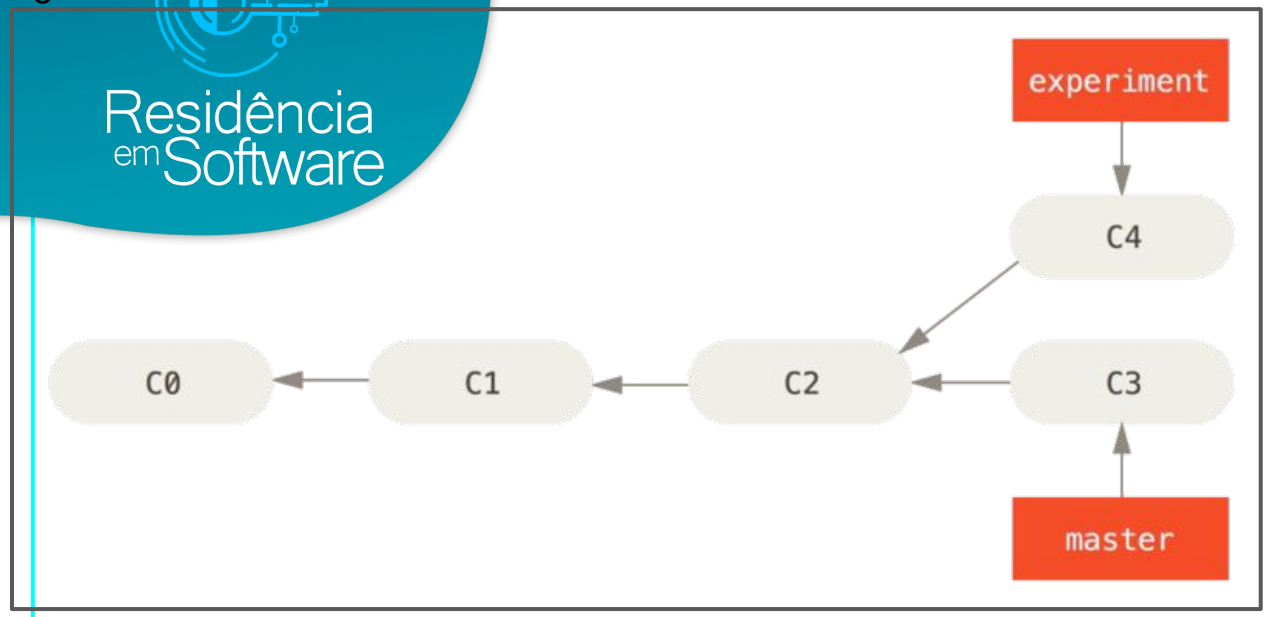
Git Rebase

O Git Rebase é uma abordagem que altera a base de um branch, reescrevendo o histórico de commits. Quando você executa o comando "`git rebase [branch_alvo]`", o Git transfere as alterações do branch atual para o ponto de ramificação do branch alvo, aplicando os commits do branch atual na sequência do branch alvo.

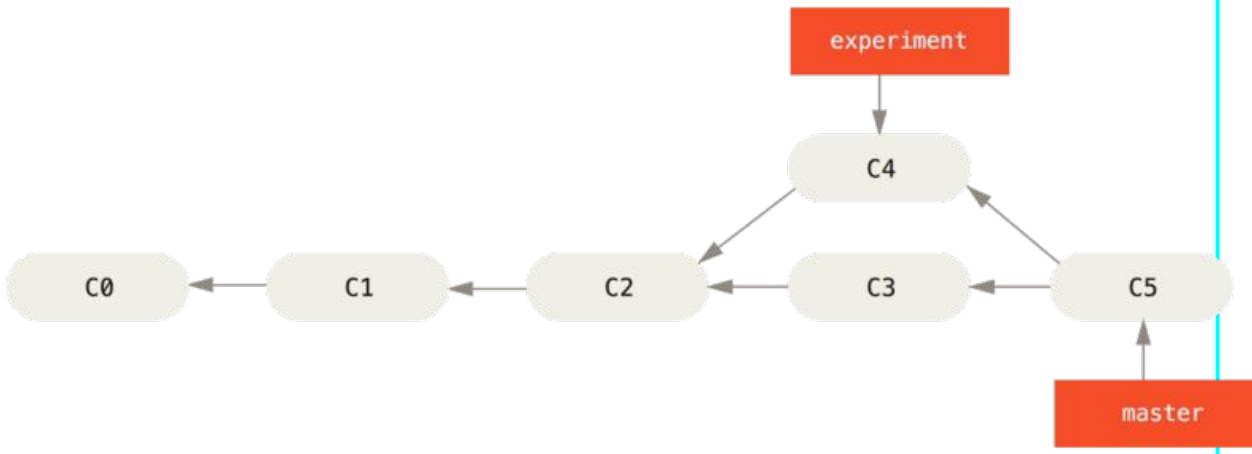
O Git Rebase oferece a vantagem de criar um histórico de commits mais linear e limpo, pois os commits do branch atual são adicionados diretamente à sequência de commits do branch alvo, sem a criação de commits de merge adicionais.

No entanto, como o histórico de commits é reescrito, isso pode gerar problemas de conflitos maiores em relação ao Git Merge, especialmente quando o branch atual possui muitos commits ou foi compartilhado com outras pessoas.

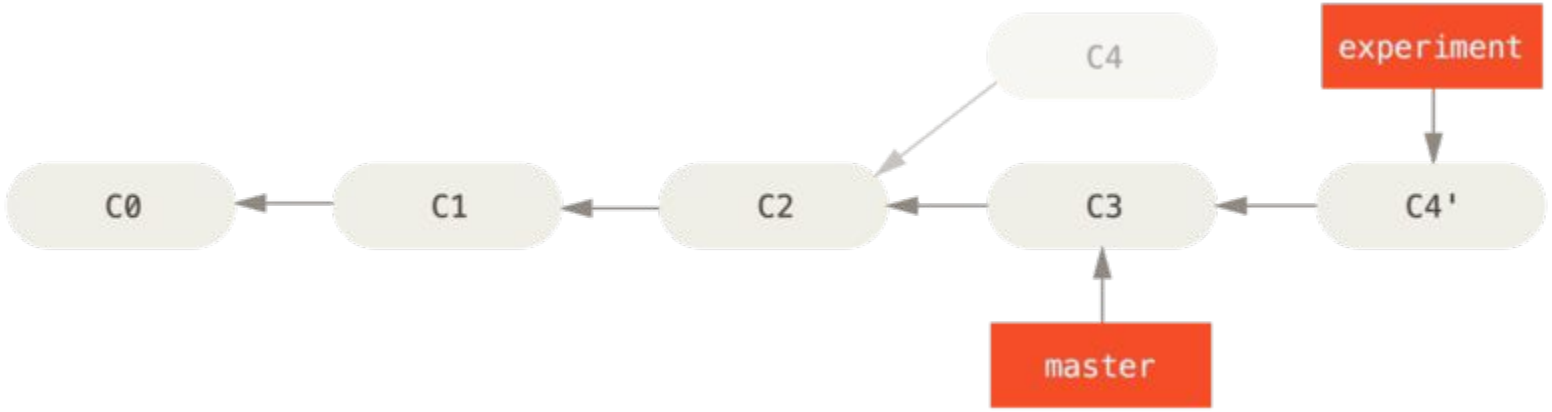
```
git checkout -b experiment
git commit -a -m 'New feature'
```




```
git checkout master
git merge experiment
```




```
git checkout experiment
git rebase master
```



Residência em Software

Prática - mesclando mudanças

1. Em um dos diretórios criados anteriormente crie um arquivo chamado “merge1.txt”;
2. Adicione seu nome ao arquivo e salve-o;
3. Envie as mudanças para o GitHub.
4. Vá ao outro diretório e antes de sincronizar com o servidor crie uma nova branch “hotfix”;
5. Nessa branch, adicione um arquivo chamado “merge2.txt” e adicione o nome de um filme qualquer;
6. Faça commit das alterações e sincronize
7. Faça a mesclagem das branches com o merge e veja o histórico de commits.
8. Repita o processo para e faça a mesclagem com o rebase e veja o histórico de commits.

Residência em Software

Fork e Pull Request

Contribuir com projetos usando fork e pull request é uma maneira comum e colaborativa de participar e melhorar projetos mantidos por outras pessoas ou organizações. Para fazer isso siga os passos:

- **Faça um Fork do Repositório:** Acesse o projeto que deseja contribuir no GitHub. No canto superior direito, clique no botão "Fork" para criar uma cópia do repositório para sua própria conta do GitHub.
- **Clone o Repositório `Forked`:** No seu perfil do GitHub, acesse o repositório que você fez o fork. Clique no botão "Code" e copie a URL do repositório. No terminal, use o comando `git clone [URL_repositório]` para clonar o repositório para o seu computador.
- **Crie um Branch para a Contribuição:** No repositório local, crie um novo branch com um nome descritivo para a sua contribuição. Use o comando `git checkout -b [nome_branch]` para criar e alternar para o novo branch.
- **Faça Suas Alterações:** Faça as alterações que deseja contribuir para o projeto, como correções de bugs, implementação de novas funcionalidades ou atualização de documentação.

Residência em Software

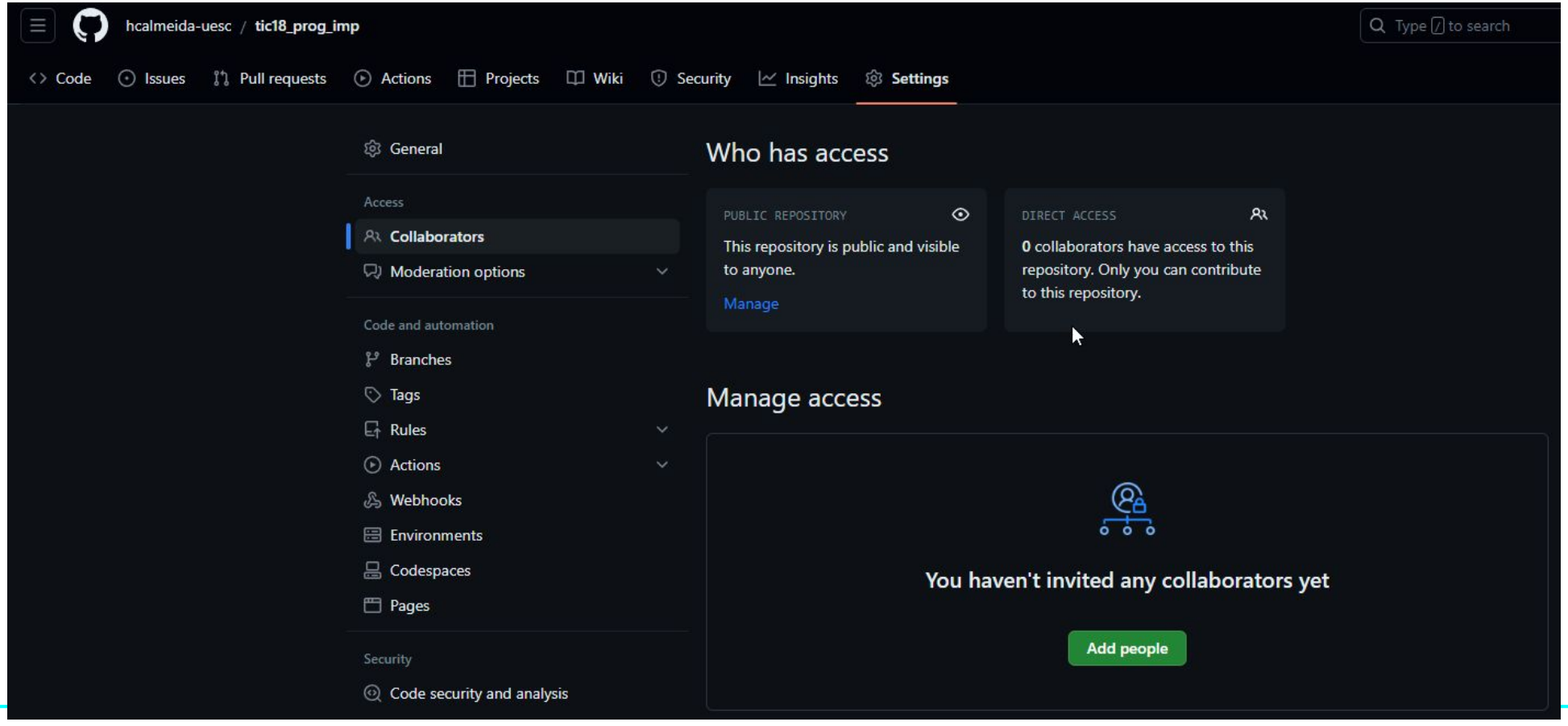
Fork e Pull Request

- **`Commit` Suas Alterações:** Use o comando git add . para adicionar todas as suas alterações ao próximo commit, e em seguida, use git commit -m "[mensagem de commit]" para criar um novo commit com suas alterações.
- **Envie as Alterações para o Seu Repositório `Forked`**: Use o comando git push origin [nome-do-branch] para enviar suas alterações para o seu repositório forked no GitHub.
- **Crie um Pull Request**: No seu repositório forked no GitHub, clique no botão "Compare & pull request" ao lado do nome do seu branch. Preencha uma mensagem descritiva para o pull request, explicando suas alterações. Clique em "Create pull request" para enviar o pull request para o repositório original.
- **Aguarde a Análise e a Aceitação:** Agora, os mantenedores do projeto original irão analisar o seu pull request. Eles podem solicitar alterações adicionais ou aprovar suas mudanças e fundi-las ao projeto principal.
- **Mantenha-se Ativo na Discussão:** Esteja aberto para receber feedback e discutir suas alterações com os mantenedores do projeto. Este processo pode envolver discussões técnicas e possíveis melhorias em suas contribuições.

Residência em Software

Gerenciamento de Colaboradores no GitHub

Nos repositórios do GitHub para trabalhar em equipe é necessário adicionar os colaboradores na seção Settings -> Collaborators:



Residência em Software

Boas Práticas de Versionamento de Código

- **Utilize Branches Adequadamente:** Faça bom uso de branches no Git para isolar diferentes funcionalidades ou tarefas. Crie um branch para cada nova feature ou correção de bug. Isso facilita o desenvolvimento paralelo, evita conflitos desnecessários e permite que você mantenha a branch principal (geralmente chamada de "master" ou "main") estável.
- **Mensagens de Commit Descritivas:** Escreva mensagens de commit claras e descritivas. Isso ajuda a entender rapidamente o que foi alterado em cada commit. Uma boa prática é começar a mensagem com um verbo no imperativo, como "Adicionar", "Corrigir" ou "Atualizar", seguido de uma breve descrição da alteração.
- **Gitignore para Arquivos Desnecessários:** Utilize o arquivo .gitignore para evitar o versionamento de arquivos temporários, arquivos gerados pelo sistema ou dados sensíveis (como senhas ou chaves de API). Isso mantém o repositório limpo e evita a poluição do histórico com arquivos desnecessários.

Residência em Software

Boas Práticas de Versionamento de Código

- **Tags para Versões Estáveis:** Use tags para marcar versões estáveis do seu projeto. Isso permite que você identifique facilmente versões importantes e crie pontos de referência para futuras atualizações. Tags são especialmente úteis quando você precisa fazer deploy ou enviar uma versão específica do seu software.

- **Revisão e Comunicação:** Encoraje revisões de código e a comunicação eficiente entre os membros da equipe. Revisões de código ajudam a identificar problemas e melhorar a qualidade do código. Além disso, uma comunicação aberta sobre mudanças, novas features e decisões de versionamento garante que todos os membros da equipe estejam alinhados e cientes do estado do projeto.

  Ao seguir essas dicas, você estará estabelecendo uma base sólida para um versionamento eficiente e organizado, garantindo que o desenvolvimento do seu projeto seja mais suave, colaborativo e bem documentado.

Residência em Software

Considerações Finais

Esperamos que esta aula tenha fornecido uma visão abrangente sobre os conceitos e práticas essenciais para um gerenciamento eficiente do código-fonte em projetos de desenvolvimento de software.

Ao se aprofundar no versionamento de código com o Git, você estará preparado para contribuir com projetos de código aberto, trabalhar de forma mais organizada em equipes e ter um histórico claro e confiável de suas alterações ao longo do tempo.

Lembre-se de continuar praticando e explorando outras funcionalidades do Git para aprimorar suas habilidades e tornar-se um desenvolvedor mais eficiente.

Muito obrigado e bons commits! 🚀

Residência em Software

Dúvidas