

Sistemas Embarcados

Metodologia de projeto de sistemas embarcados / Git

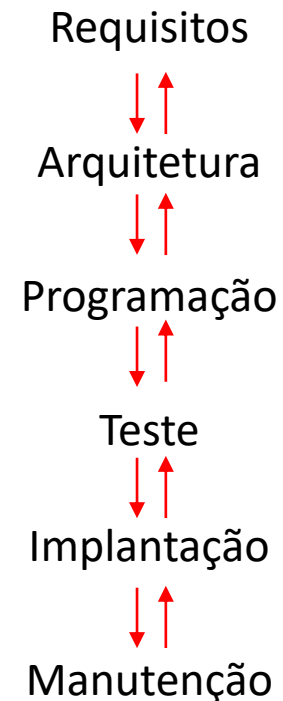
Tiago Piovesan Vendruscolo



Esta licença permite que outros remixem, adaptem e criem a partir do trabalho para fins não comerciais, desde que atribuam o devido crédito aos autores originais. [4.0 international](https://creativecommons.org/licenses/by-nc-nd/4.0/)

- Conforme visto nas últimas aulas, as etapas de desenvolvimento de um sistema são:
 - *Análise/requisitos, projeto/especificação, desenvolvimento, integração e prototipagem/testes.*
- Metodologias foram criadas para auxiliar no desenvolvimento dessas etapas.
- Alguns exemplos:
 - *Cascata, espiral, V, ágil, etc.*

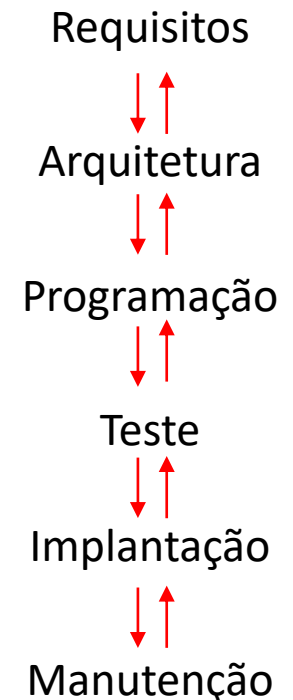
- Modelo Waterfall ou Cascata – década de 70
 - *Possui fases bem definidas e sequenciais.*
- Modelo inicial para desenvolvimento de software:
 - *Requisitos: determinação das características básicas.*
 - *Arquitetura: Decompor em módulos básicos.*
 - *Programação: Implementar e integrar.*
 - *Teste: Execução do projeto e busca por bugs.*
 - *Manutenção: Correção de erros, atualização.*



- Vantagens:
 - *Fases bem definidas ajuda a detectar erros cedo (prejuízo menor).*
 - *Busca promover a estabilidade dos requisitos, assim, o projeto só segue em frente quando os requisitos são aceitos.*
 - *Funciona bem com requisitos conhecidos e estáveis.*
 - *É uma boa opção quando a preocupação com a qualidade está acima das preocupações com custo ou tempo de desenvolvimento.*
 - *É adequado para equipes tecnicamente fracas ou inexperientes, pois fornece uma estrutura sólida ao projeto, direcionando todos os esforços.*

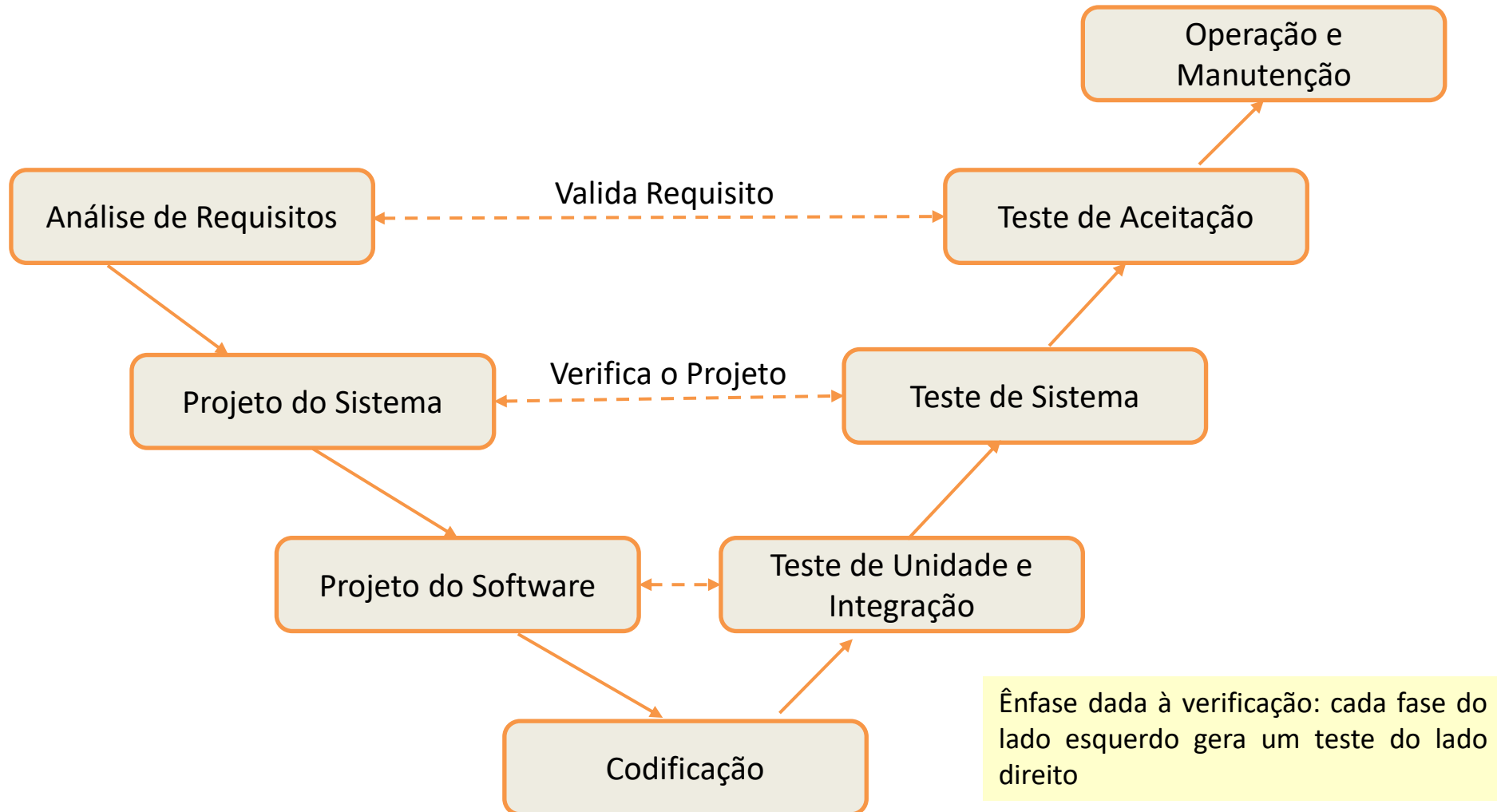
■ Desvantagens

- *Somente feedback local.*
- *Não produz resultados tangíveis até a fase de codificação (na perspectiva do cliente, o projeto está parado).*
- *Assume que o hardware é fornecido e/ou previamente decidido.*
- *Não há flexibilidade com requisitos.*
- *Vários aspectos do sistema só serão postos à prova na fase de testes.*



Modelo em V

- Desenvolvido na década de 90, basicamente o modelo em cascata é colocado na forma de V.



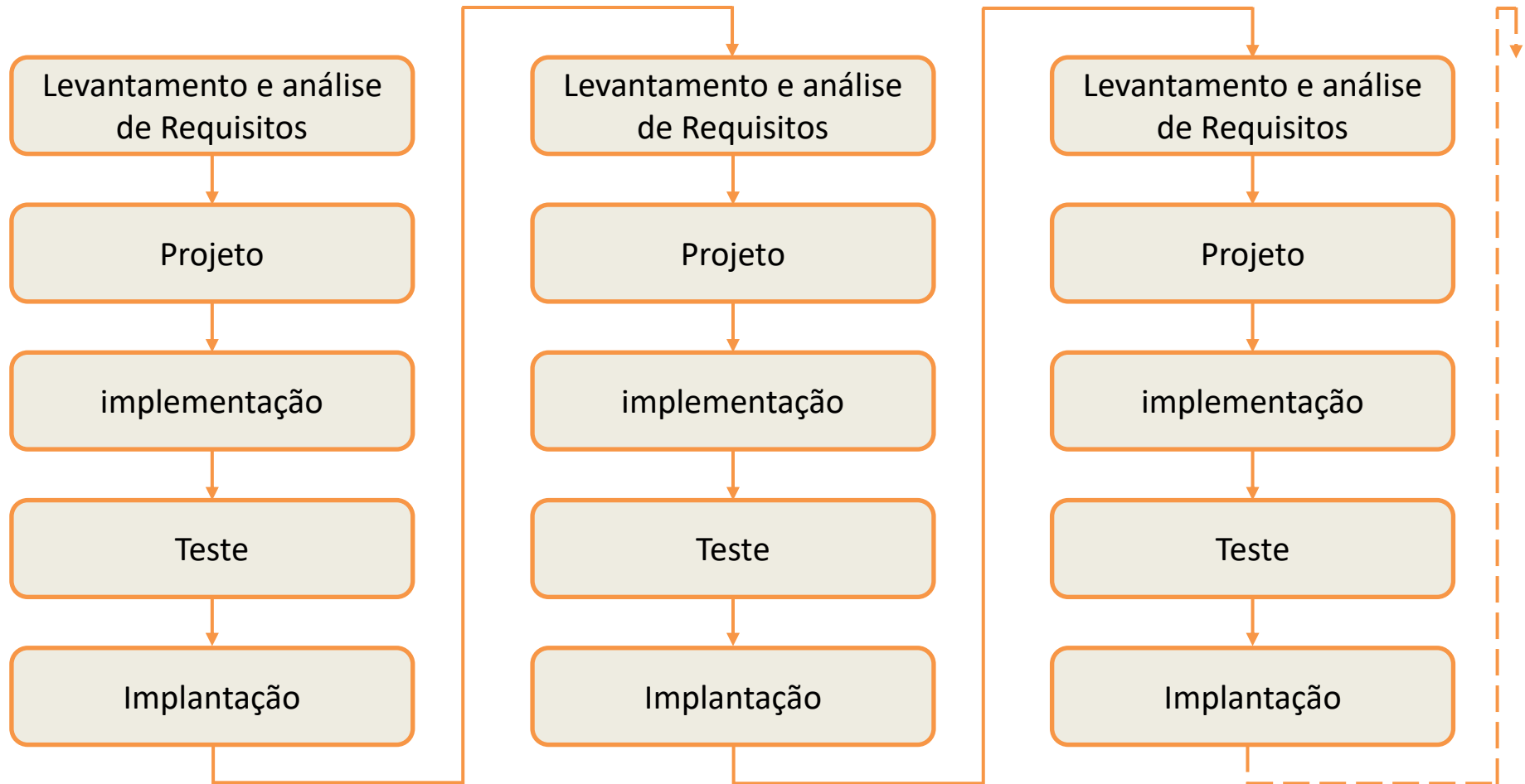
- Cada nível do lado esquerdo pode (e deve) ser testado com um nível superior do lado direito.
 - *Ex. Após a análise de requisitos, é gerado um teste de aceitação, para confirmar que os requisitos levantados atendem à expectativa do cliente.*
- No entanto, o cliente não consegue acompanhar todo o desenvolvimento. A primeira versão do software só é apresentada no final do ciclo, no entanto, devido ao planejamento e teste final, o risco é menor.
- Vantagem: O cliente tem um maior acompanhamento no desenvolvimento do projeto, o que reduz os riscos.

Modelo com ciclo de vida incremental

- Desenvolvido na década de 80, esse modelo faz um agrupamento de requisitos de acordo com sua funcionalidade, dessa forma, o sistema final é desenvolvido em módulos, cada módulo é equivalente ao modelo em cascata.
- O desenvolvedor, em conjunto com o cliente, define quais módulos terão prioridade de desenvolvimento.
- Após a conclusão de cada módulo “cascata”, o cliente recebe um software funcional.

Modelo com ciclo de vida incremental

- Desenvolvido na década de 80, esse modelo faz um agrupamento de requisitos de acordo com sua funcionalidade, dessa forma, o sistema final é desenvolvido em módulos.



Modelo com ciclo de vida incremental

- Vantagens

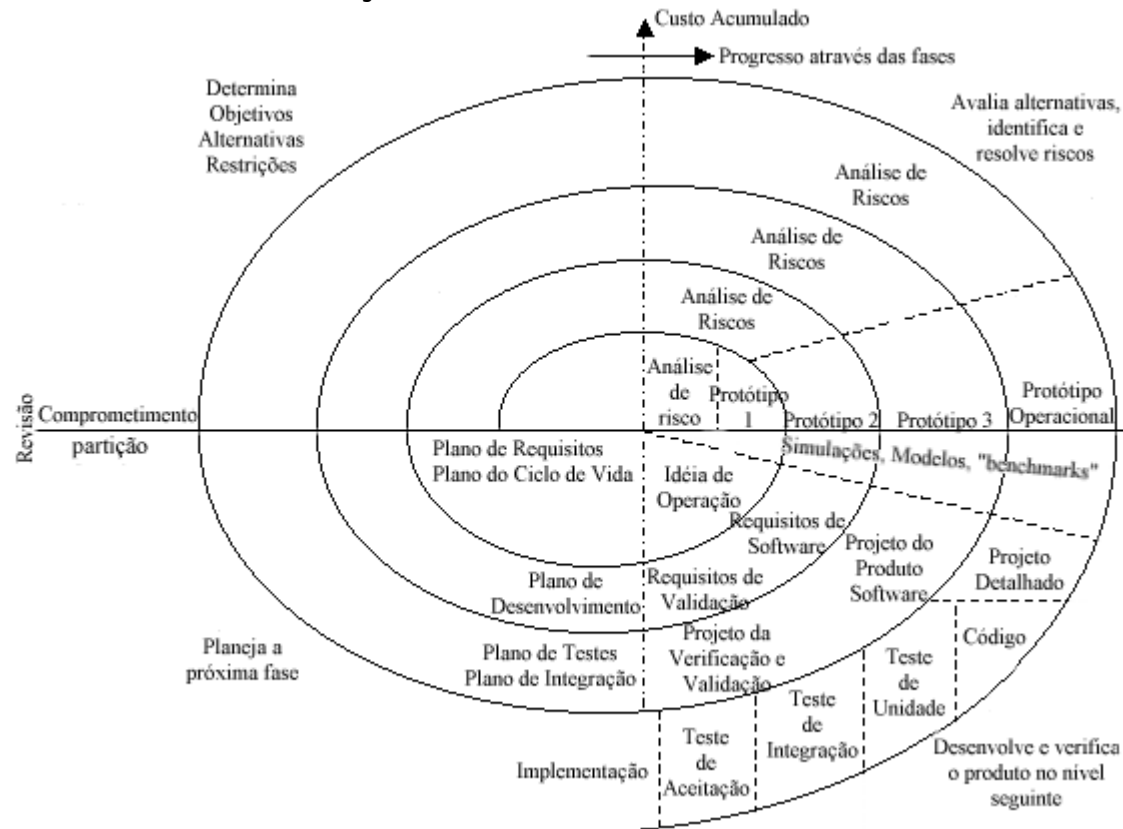
- *O cliente pode acompanhar de perto todo o processo, realimentando com feedback.*
- *O sistema pode ser desenvolvido por equipes pequenas, pois os módulos não são desenvolvidos em paralelo.*

- Desvantagens

- *A documentação é mais complexa, devido à constante atualização do projeto com os feedback do cliente.*

Modelo Espiral

- Modelo desenvolvido na década de 80. Baseado na realização de ciclos iterativos: A cada iteração da espiral, se constrói uma versão mais completa do produto.
- Possui ciclos de prototipação dedicada à análise de riscos de projeto, o que permite um envolvimento constante do cliente nas decisões.
- Orientado a redução de riscos.
- Aceita novas informações entre uma fase e outra.

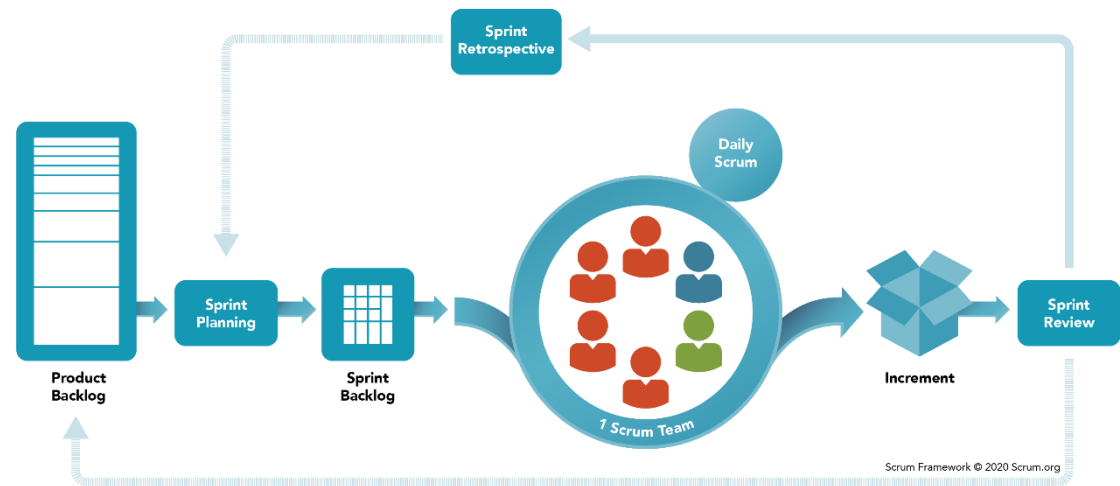


<https://www.devmedia.com.br/ciclos-de-vida-do-software/21099>

- Vantagens:
 - *Interessante para projetos de grande porte, devido à constante análise de risco.*
 - *As primeiras iterações são as mais baratas no que diz respeito a tempo e recursos.*
 - *A escolha dos riscos a serem explorados é feita em função das necessidades do projeto.*
 - *A medida que os custos aumentam, os riscos diminuem.*
- Desvantagens:
 - *Este modelo não fornece indicações suficientes sobre quantidade de trabalho esperada em cada ciclo.*
 - *O tempo de desenvolvimento se torna imprevisível.*
 - *Esse modelo possui movimentação entre fases complexa, o que torna complexo o gerenciamento do projeto.*

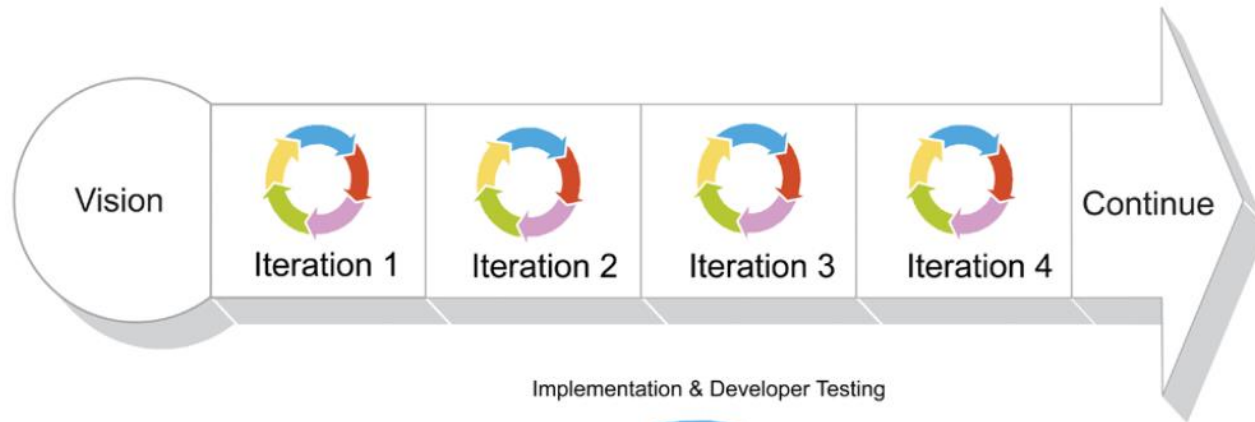
Modelo Ágil - Scrum

- Desenvolvido em 1993. Ele é baseado no modelo cascata, no entanto, para o modelo cascata ser bem sucedido, é necessário que as fases iniciais estejam muito bem definidas e alinhadas com o cliente, caso contrário, o custo para refazer é alto.
 - *Na metodologia Scrum, cada fase possui várias iterações, o que permite constantes atualizações e aperfeiçoamentos.*
- Particionamento:
 - *Backlog do produto: Lista total de ações necessárias para o desenvolvimento do produto.*
 - *Sprint backlog: Lista de ações de um Sprint.*
 - Sprint: Pequenos ciclos (tarefas) de entrega. Mais fáceis de controlar, gerenciar e entregar.

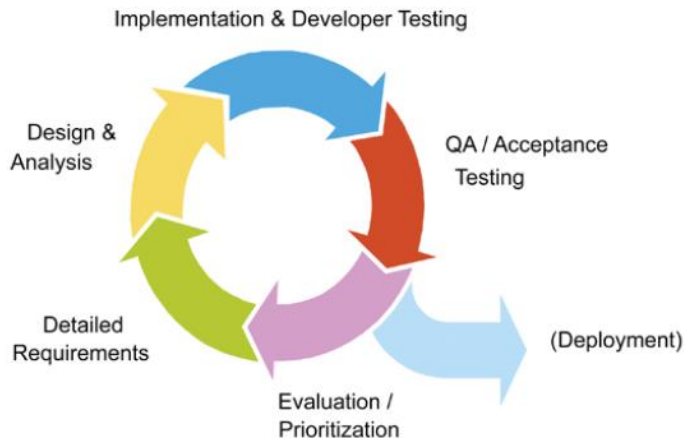


Fonte: <https://www.scrum.org/resources/what-is-scrum>

Modelo Ágil - Scrum



Iteration Detail



<https://scrumreferencecard.com/scrum-reference-card/>

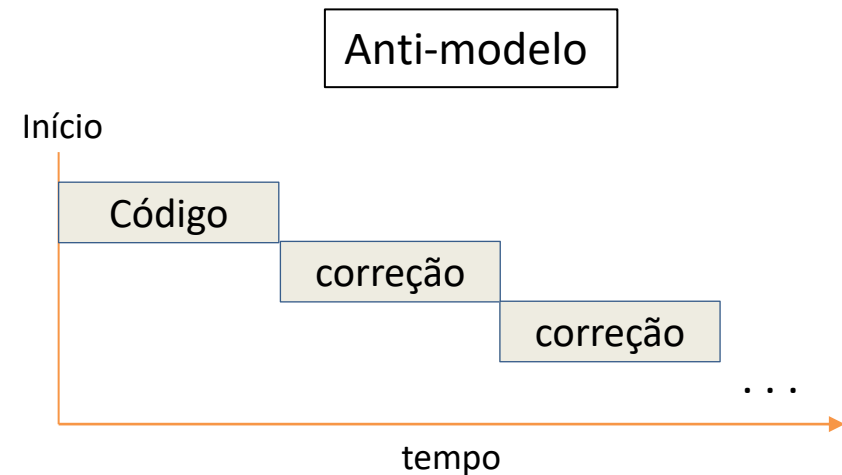
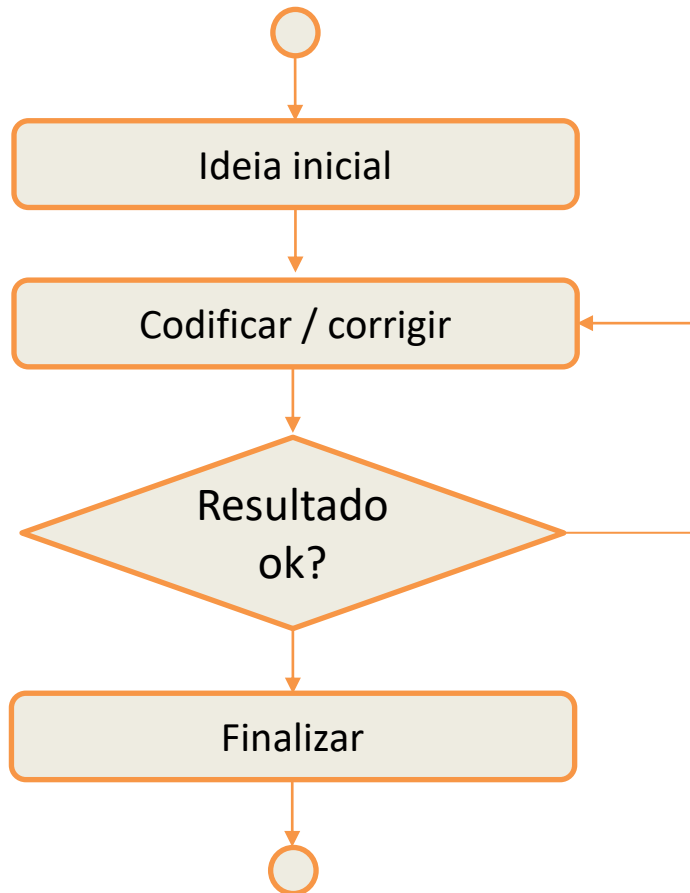
- Cerimônias Scrum
 - *Planejamento do Sprint*
 - *Daily Scrum*
 - *Revisão de Sprint*
 - *Retrospectiva de Sprint*

Guia rápido:
<https://www.youtube.com/watch?v=InbOnXMAA7k>

- Vantagens do Scrum
 - *Atualização (melhoria) constante*
 - *Feedback contínuo – transparência*
 - *Maior agilidade*
 - *Gerenciamento de riscos*
- Desvantagem do Scrum
 - *Prazos curtos/incoerentes: O projeto pode não ser finalizado a tempo ou não atingir a qualidade esperada, frustrando o cliente.*
 - Reuniões muito longas.
 - *Uma tarefa pode passar por inúmeras atualizações, de forma que não é mais possível “encaixar” no projeto, sendo necessário alterar o conceito original.*

“Modelo” mais utilizado...Codificar e consertar (code and fix)

- Construir um entendimento preliminar sobre o sistema que deve ser desenvolvido.
- Implementar uma primeira versão desse sistema.
- Codificar, corrigir e expandir até satisfazer o projeto desejado.



“Modelo” mais utilizado...Codificar e consertar (code and fix)

- Não é um modelo de processo, na verdade, é considerado um anti-modelo.
 - *Não há previsibilidade em relação às atividades;*
 - *Não há previsibilidade em relação aos resultados obtidos.*

Qual o grande problema em relação ao hardware (sistemas embarcados)?

- Pode chegar um ponto do projeto onde o hardware selecionado:
 - *Não comporte mais o código (exceda a capacidade de memória).*
 - *Desempenho de processamento insuficiente (não atende mais os requisitos de tempo).*
 - *Periféricos insuficientes.*
- Como corrigir? Reiniciando o projeto...

“Modelo” mais utilizado...Codificar e consertar (code and fix)

- Vantagens:
 - *O progresso é facilmente visível a medida que o programa vai ficando “pronto”.*
- Desvantagens:
 - Não há tempo para documentação, planejamento ou projeto.
 - Não há necessidade de treinamento ou conhecimento especiais. Qualquer programador pode desenvolver software com esse modelo de desenvolvimento.
 - É muito difícil avaliar a qualidade e os riscos do projeto.
 - Se no meio do projeto a equipe descobrir que as decisões arquiteturais estavam erradas, não há solução, a não ser começar de novo.

- Documentação automática
 - *Doxygen – Gera a documentação automática do código. Possui saída em html, rtf, latex, etc.*

Macros

```
#define LED1 PA0
#define LED2 PA1
#define LED3 PA2
#define LED4 PA3
#define Botao PA7
```

Functions

```
void setup ()
void ESTADO_1 (void)
void ESTADO_2 (void)
void ESTADO_3 (void)
void ESTADO_4 (void)
void loop ()
```

◆ ESTADO_1()

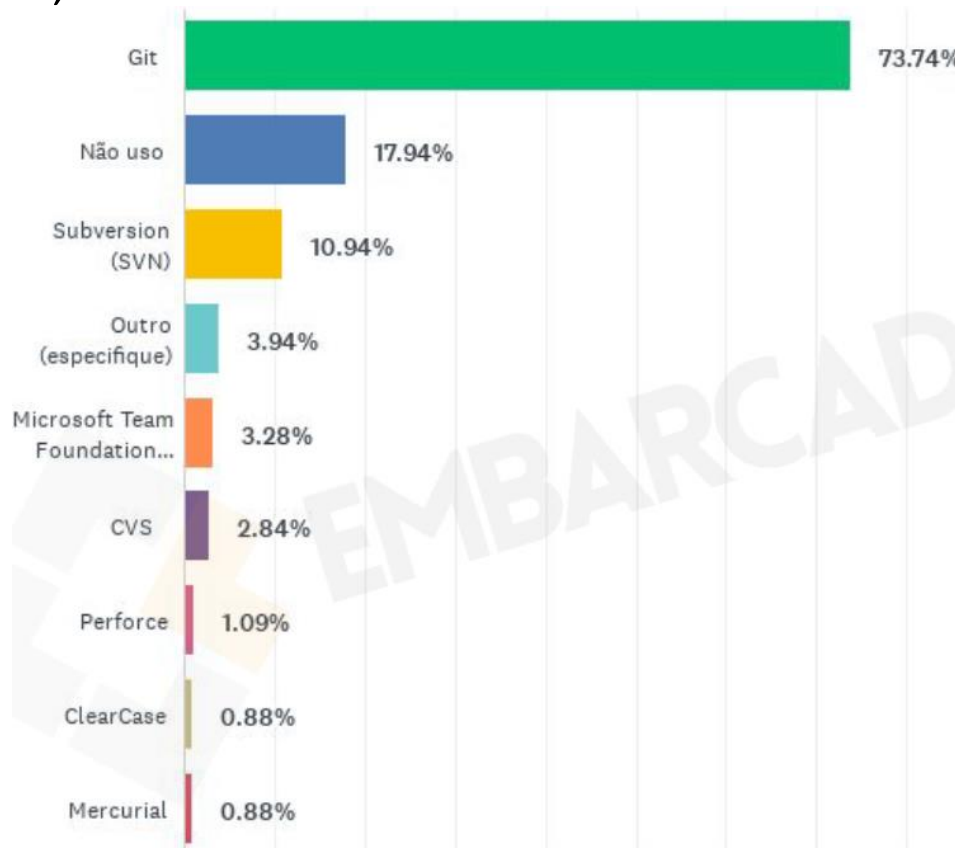
void ESTADO_1 (void)

```
24 {
25     digitalWrite(LED1, HIGH);
26     digitalWrite(LED2, LOW);
27     digitalWrite(LED3, LOW);
28     digitalWrite(LED4, LOW);
29     if(digitalRead(Botao)){
30         PonteiroDeFuncao = ESTADO_2;}
31     else
32         {PonteiroDeFuncao = ESTADO_4;}
33 }
```

- Versionamento de código
 - *Utilizado para gerenciar diferentes versões no desenvolvimento de um documento qualquer.*
- Vantagens
 - *Controle do histórico: Bastante útil quando é necessário recuperar versões mais antigas e estáveis, pois é possível marcar as versões em que o sistema estava estável, Também utilizado para acompanhar o desenvolvimento do sistema.*
 - *Trabalho em equipe: Permite que várias pessoas trabalhem com o mesmo sistema sem gerar conflito nas edições. Também evita que uma alteração feita por um usuário não seja incorporado na versão oficial.*

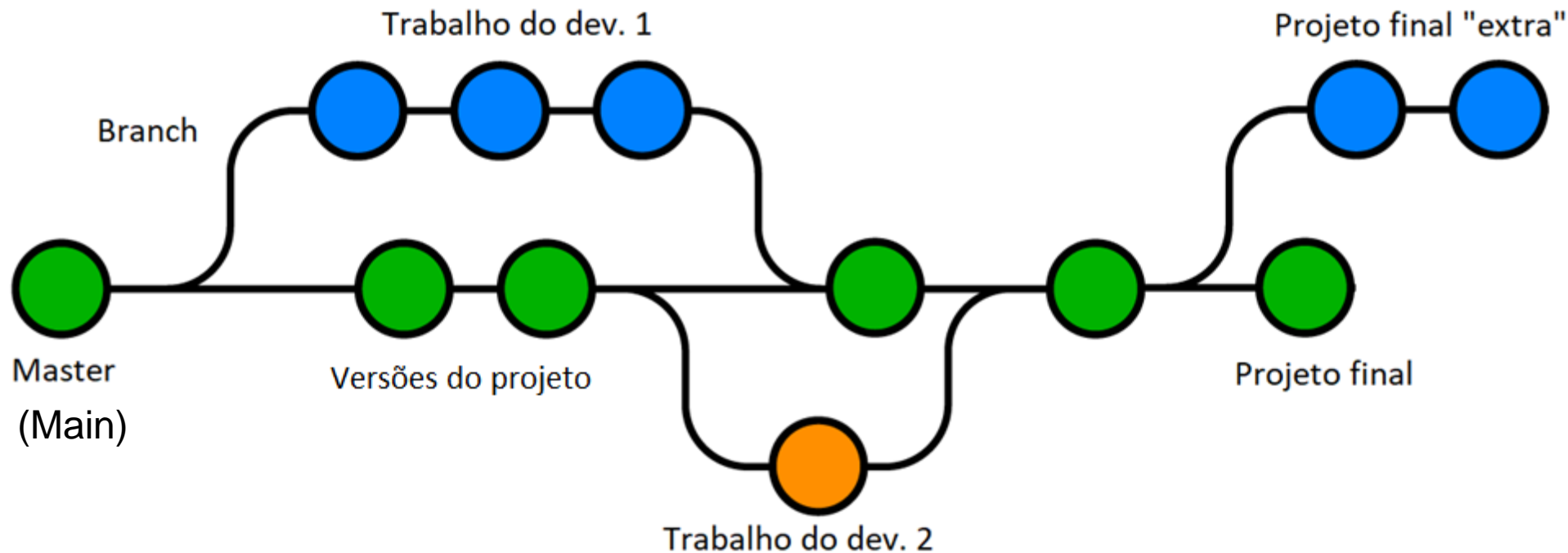
Documentação e controle de versão

- Vantagens
 - *Divisão de projeto: Possibilita a divisão do projeto em várias linhas de desenvolvimento, de forma que possam ser trabalhadas em paralelo.*
- Exemplos: Git, svn, etc.



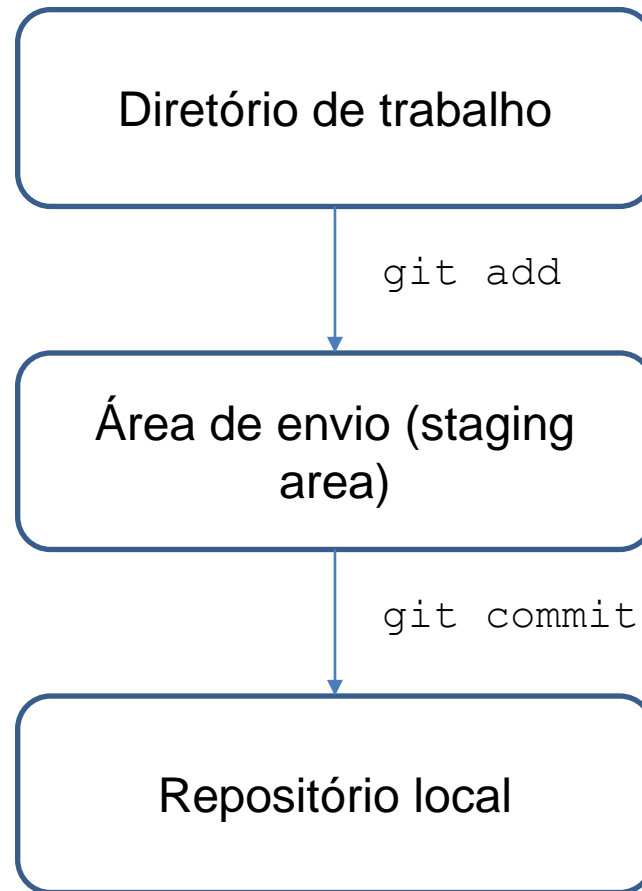
<https://embarcados.com.br/relatorio-da-pesquisa-sobre-o-mercado-brasileiro-de-sistemas-embarcados-e-iot-2023/>

- Git: Trabalho local
- Github: Armazenamento online

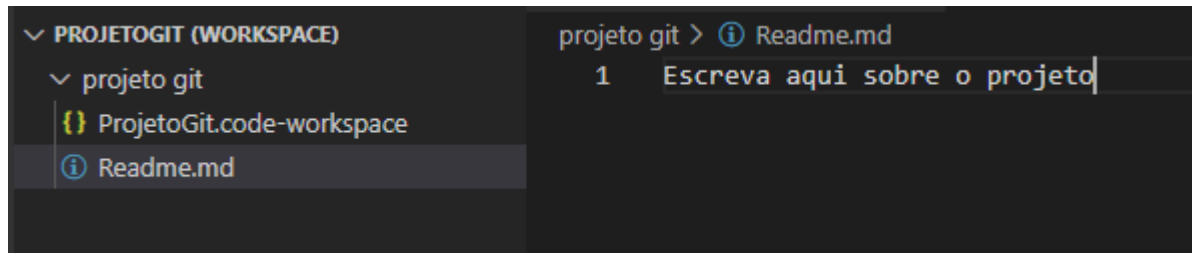


■ Comandos

- *Init: Iniciar a branch master na pasta de trabalho*
- *Add: Adicionar arquivo para a área de envio*
- *status: Mostra a situação do branch aberto: Arquivos enviados e na área de envio.*
- *commit - ir adicionando versões (pastas) – “congela o arquivo”*
- *merge - unir versões*
- *push - colocar o commit do git no github*
- *pull - puxar o que está no repositório do github (quando vc quer puxar um arquivo desenvolvido por outra pessoa)*

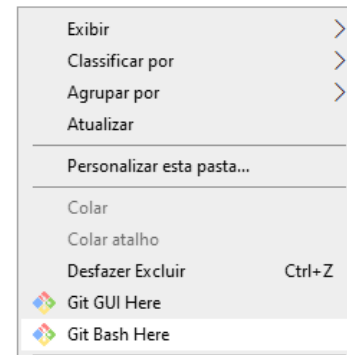


- Link para download
 - <https://git-scm.com/downloads>
- Crie uma pasta para o projeto e abra o Visual Studio Code
 - *Crie um arquivo Readme.md (extensão markdown – linguagem de marcação)*



- Após, abra o Git Bash dentro da pasta do projeto

Nome	Data de modificação	Tipo	Tamanho
ProjetoGit.code-workspace	10/02/2022 17:05	Arquivo Fonte Co...	1 KB
Readme.md	10/02/2022 17:06	Arquivo Fonte Ma...	1 KB



- Digite `git init` para iniciar a branch master (será criada uma pasta `.git` na pasta do projeto)

```
tvend@TiagoV MINGW64 ~/Desktop/projeto git
$ git init
Initialized empty Git repository in C:/Users/tvend/Desktop/projeto git/.git/

tvend@TiagoV MINGW64 ~/Desktop/projeto git (master)
$ ..
```

- Adicione o arquivo `Readme.md` para enviar para o branch digitando “`git add Readme.md`” (ainda não está no branch de fato, está na área de envio)
- Digite `git status` para verificar se o arquivo foi adicionado

```
tvend@TiagoV MINGW64 ~/Desktop/projeto git (master)
$ git add Readme.md

tvend@TiagoV MINGW64 ~/Desktop/projeto git (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   Readme.md
```

- Repare que ele está falando que o arquivo ainda está na área de espera.

- Antes de fazer o primeiro commit, você precisa se identificar (para que as versões sejam identificadas também pelo autor), para isso, use os comandos abaixo

```
git config --global user.email "you@example.com"  
git config --global user.name "Your Name"
```

- Agora faça o envio do arquivo para o branch usando `git commit -m "versao 1"`

```
tvend@TiagoV MINGW64 ~/Desktop/projeto git (master)  
$ git commit -m "versao 1"  
[master (root-commit) f3d2295] versao 1  
1 file changed, 1 insertion(+)  
create mode 100644 Readme.md
```

O parâmetro -m no primeiro comando solicita ao Git a execução da operação de renomeação

- Se verificar o status, verá que não tem nenhum arquivo na área de envio

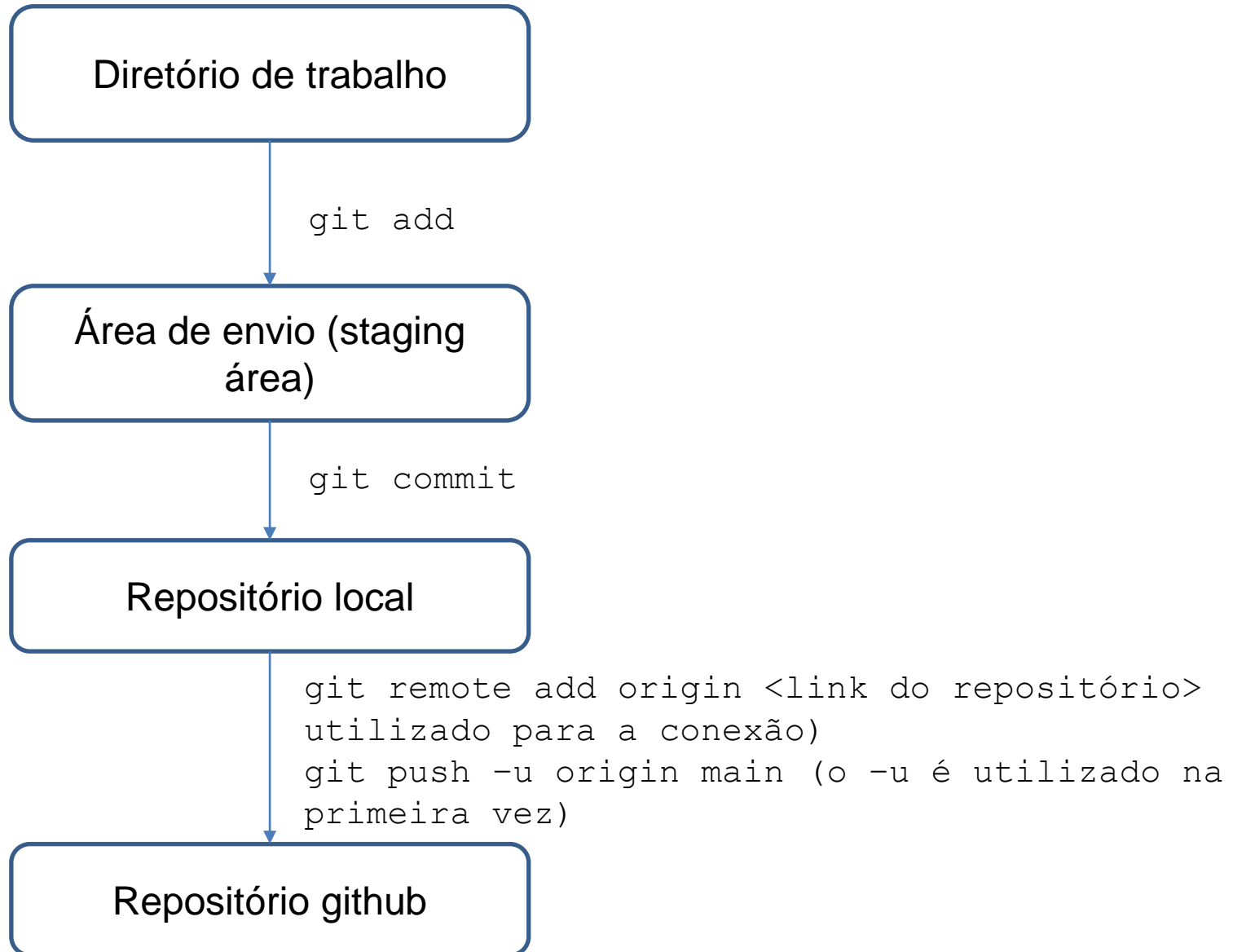
```
tvend@TiagoV MINGW64 ~/Desktop/projeto git (master)  
$ git status  
On branch master  
nothing to commit, working tree clean
```

- Dependendo o lugar, ao invés de “master”, o nome da branch principal, por padrão é “main”. Para renomear, utilize o comando:

```
git branch -M "main"
```

```
tvend@TiagoV MINGW64 ~/Desktop/projeto git (master)
$ git branch -M "main"

tvend@TiagoV MINGW64 ~/Desktop/projeto git (main)
$ |
```



- Crie uma conta no github, e logo após crie um repositório

Quick setup — if you've done this kind of thing before



Set up in Desktop

or

HTTPS

SSH

`https://github.com/TiagoPV/Tutorial.git`

Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

...or create a new repository on the command line

```
echo "# Tutorial" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/TiagoPV/Tutorial.git
git push -u origin main
```

“origin” é o apelido no repositório

Digite essa linha no bash para fazer a conexão com o github

Utilize esse comando para enviar (vai aparecer uma janela de login)

...or push an existing repository from the command line

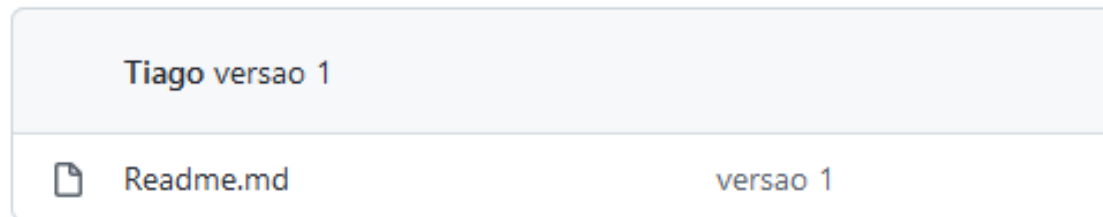
```
git remote add origin https://github.com/TiagoPV/Tutorial.git
git branch -M main
git push -u origin main
```

Se aparecer o erro "fatal: protocol 'https' not is supported", use "git remote set-url origin +link"

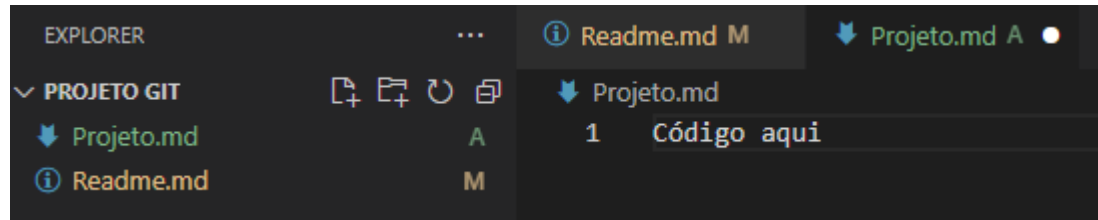
Já fizemos essa parte

```
tvend@TiagoV MINGW64 ~/Desktop/projeto git (main)
$ git push -u origin main
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 239 bytes | 239.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/TiagoPV/Tutorial.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.
```

- Atualize o seu repositório e verá que o arquivo foi inserido



- Crie um novo arquivo .md no VScode



- Digite “git add .” para adicionar todos os arquivos da pasta na área de transferência

```
tvend@TiagoV MINGW64 ~/Desktop/projeto git (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   Projeto.md
        modified:   Readme.md
```


- Digite `git commit -m "Projeto"` para criar um commit com os dois arquivos
- Após, digite `git push origin main` (o remote é utilizado apenas na primeira vez) para fazer o envio

```
tvend@TiagoV MINGW64 ~/Desktop/projeto git (main)
$ git commit -m "Projeto"
[main 83c6c0e] Projeto
2 files changed, 1 insertion(+), 1 deletion(-)
create mode 100644 Projeto.md

tvend@TiagoV MINGW64 ~/Desktop/projeto git (main)
$ git push origin main
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 12 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (4/4), 316 bytes | 316.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/TiagoPV/Tutorial.git
f3d2295..83c6c0e main -> main
```

Nome da versão atual

TiagoPV Projeto 83c6c0e 3 minutes ago 2 commits

File	Commit	Time
Projeto.md	Projeto	3 minutes ago
Readme.md	Projeto	3 minutes ago

Aqui você poderá ver as versões

main

Commits on Feb 10, 2022

Projeto

TiagoPV committed 8 minutes ago

versao 1

Tiago committed 1 hour ago

Arquivo Readme com as alterações

2 Readme.md

Line	Change	Text
...	@@ -1 +1 @@	
1	-	Escreva aqui sobre o projeto
1	+	Escreva aqui sobre o que é o projeto

- Criando e mudando o diretório para outra branch
 - Crie um novo arquivo para enviar para a branch (branch1.md)
 - Digite `git checkout -b "nova_branch"`
 - Checkout altera o diretório, "-b" cria o branch, caso não exista
 - Para deletar um branch - `git branch -d <nome da branch>`

```
tvend@TiagoV MINGW64 ~/Desktop/projeto git (main)
$ git checkout -b "nova_branch"
Switched to a new branch 'nova_branch'

tvend@TiagoV MINGW64 ~/Desktop/projeto git (nova_branch)
```

- Faça o envio com os mesmos códigos já vistos
 - `git add .`
 - `git commit -m "nova_branch"`
 - `git push origin nova_branch`

Diretório atual



- Criando e mudando o diretório para outra branch

Agora temos 2 Branches

main 2 branches 0 tags

Go to file Add file Code

TiagoPV Projeto 83c6c0e 25 minutes ago 2 commits

Projeto.md	Projeto	25 minutes ago
Readme.md	Projeto	25 minutes ago

Alterando o branch

Mostra que essa branch está mais atualizada que a main

This branch is 1 commit ahead of main. Contribute

TiagoPV nova_branch 237de6f 6 minutes ago 3 commits

Projeto.md	Projeto	28 minutes ago
Readme.md	Projeto	28 minutes ago
branch1.md	nova_branch	6 minutes ago


- Fazendo merge – A branch “nova_branch” foi unida com o main. E após, enviada para o github


```
tvend@TiagoV MINGW64 ~/Desktop/projeto git (nova_branch)
$ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.


tvend@TiagoV MINGW64 ~/Desktop/projeto git (main)
$ git merge nova_branch
Updating 83c6c0e..237de6f
Fast-forward
 branch1.md | 0
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 branch1.md

tvend@TiagoV MINGW64 ~/Desktop/projeto git (main)
$ git push origin main
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/TiagoPV/Tutorial.git
 83c6c0e..237de6f  main -> main
```

- O arquivo branch1 foi acrescentado no main

 main ▾



 2 branches




 0 tags

Go to file

Add file ▾

Code ▾

 **TiagoPV** nova_branch 237de6f 17 minutes ago  3 commits

 Projeto.md	Projeto	39 minutes ago
 Readme.md	Projeto	39 minutes ago
 branch1.md	nova_branch	17 minutes ago

- Você também pode alterar um arquivo diretamente no github, e depois atualizar o repositório local utilizando o comando `git pull`. Faça o teste!

- Clonando o repositório de um colega (não pode estar privado)
 - *Crie uma nova pasta > Abra o git bash dentro da pasta*
 - *Digite git clone <endereço>*
- Caso o colega altere o código, você pode atualizar os teus arquivos
 - *Digite cd <nome do projeto do colega>*
 - *git pull*
 - Com isso, os arquivos serão atualizados! (ou avisará que já estão atualizados)

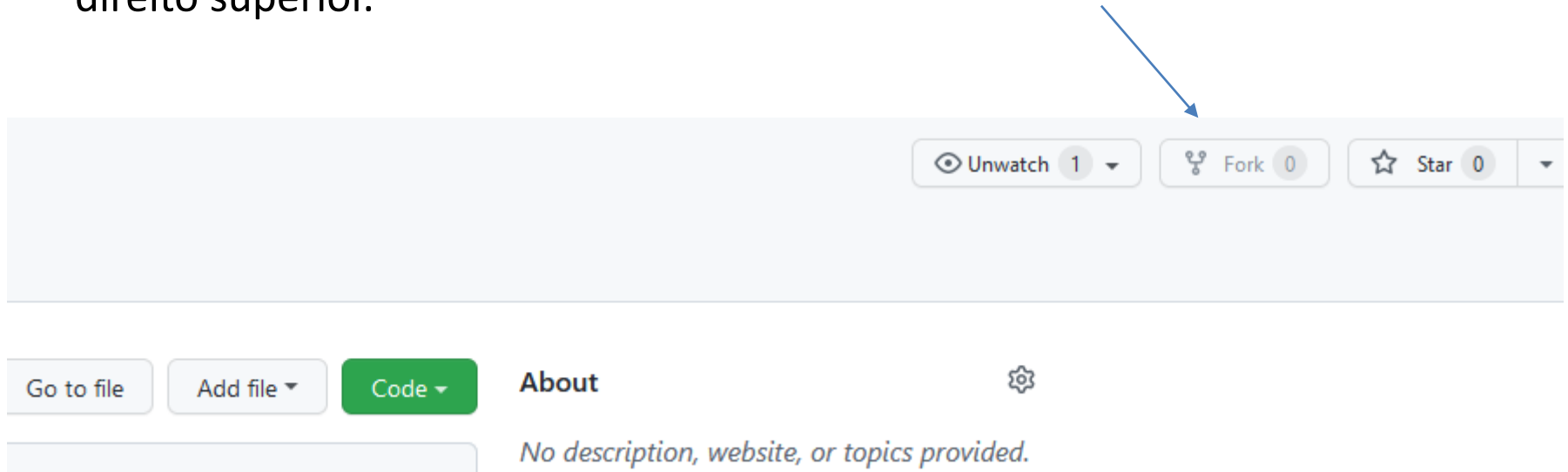
```
tvend@TiagoV MINGW64 ~/Desktop/CloneGitTiago
$ git clone https://github.com/TiagoPV/Tutorial.git
Cloning into 'Tutorial'...
remote: Enumerating objects: 9, done.
remote: Counting objects: 100% (9/9), done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 9 (delta 1), reused 8 (delta 0), pack-reused 0
Receiving objects: 100% (9/9), done.
Resolving deltas: 100% (1/1), done.

tvend@TiagoV MINGW64 ~/Desktop/CloneGitTiago
$ git pull
fatal: not a git repository (or any of the parent directories): .git

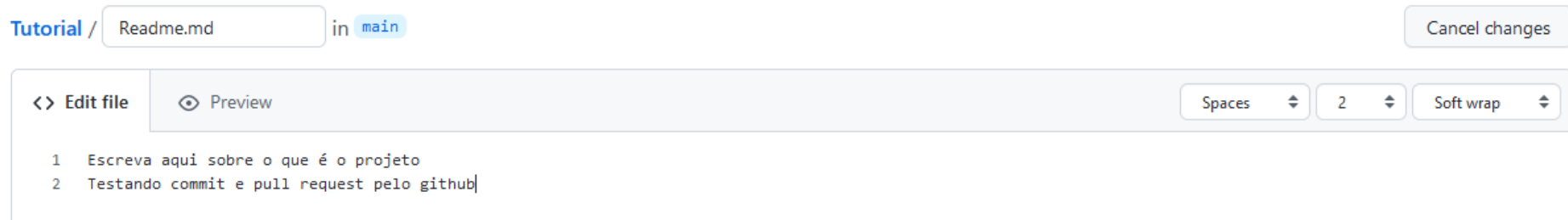
tvend@TiagoV MINGW64 ~/Desktop/CloneGitTiago
$ cd Tutorial

tvend@TiagoV MINGW64 ~/Desktop/CloneGitTiago/Tutorial (main)
$ git pull
Already up to date.
```

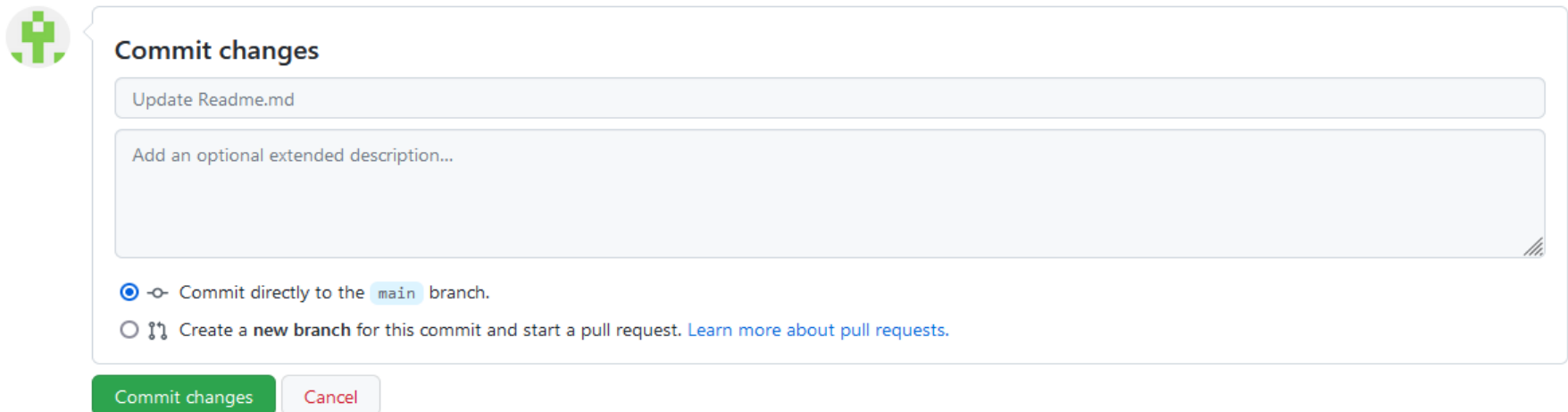
- Com o clone, você terá os arquivos no seu PC, mas não no seu repositório, caso você queira clonar para o seu repositório, é só clicar no botão “fork”, no canto direito superior.



- Você também pode editar o arquivo do repositório de outra pessoa que você fez o fork, e, caso ache interessante, envie um pull request pro autor do projeto atualizar o projeto dele (se ele achar que é interessante). Primeiro faça uma modificação no seu fork.



- Faça o commit para salvar a modificação (coloque alguma observação se necessário). O projeto original do autor em que foi feito o fork permanece inalterado.




This branch is 1 commit ahead of main.


 Contribute ▾

Nesse caso aparecerá o nome do autor

- Clicando em contribute e indo em “open pull request” você abrirá um pedido para atualizar o projeto.
- Irá aparecer se é possível fazer o merge no projeto.
- Acrescente comentários sobre a modificação e clique em create pull request.
- O autor do projeto irá receber o pedido na página inicial do projeto em questão


 **TiagoPV / Tutorial** Private


 Code


 Issues

 Pull requests

 Actions

 Projects

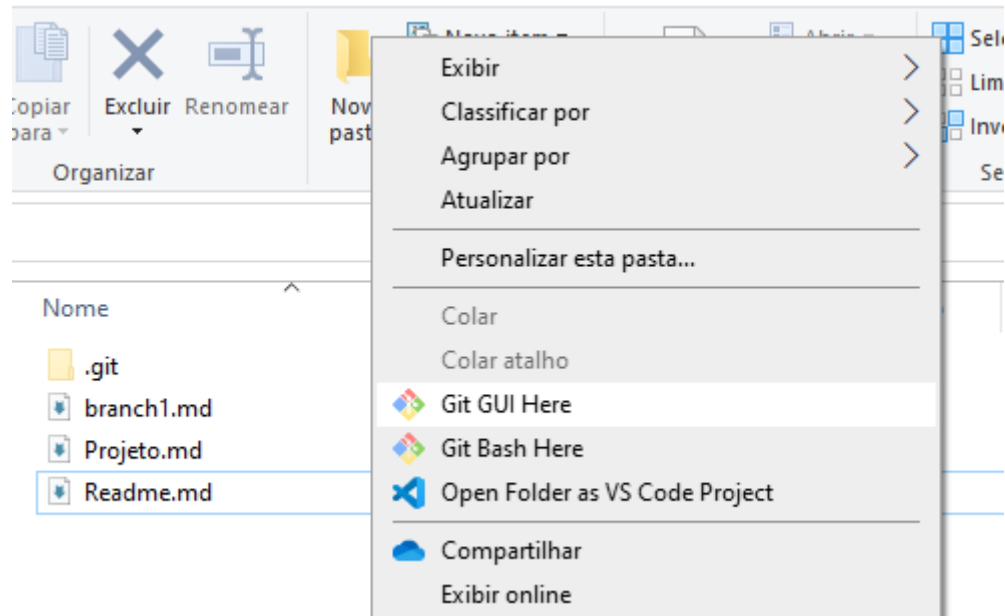
 Security

 Insights

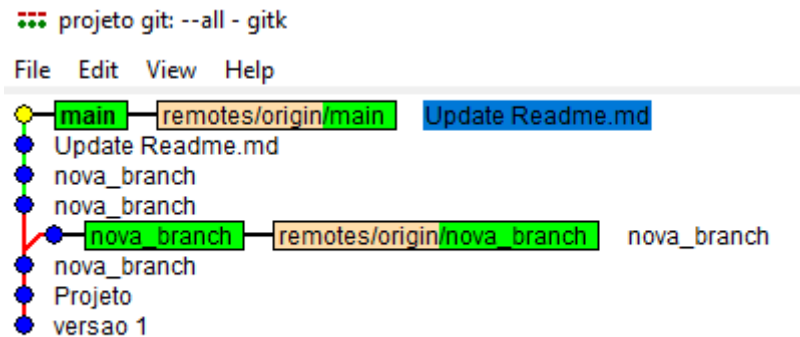
 Settings

- Para aceitar, clique em “merge pull request”

- Para verificar seus projetos, é possível utilizar o Git GUI, clicando com o botão direito na pasta do projeto:



- Ou digitando git gui no git bash. Indo em Repository > Visualize main's history ou Visualize all branch history, é possível verificar a situação do projeto de forma visual.



- Existem vários outros GUI clientes em <https://git-scm.com/downloads/guis>

- Sensores e atuadores

- <https://www.devmedia.com.br/ciclos-de-vida-do-software/21099>
- <https://www.scrum.org/resources/what-is-scrum>
- <https://scrumreferencecard.com/scrum-reference-card/>