



# Workshop GIT - BNP

## Contexto GIT - Filipe

**Linus Torvalds - Motivações**

SCM - Source Code Management

CVS - Concurrent Version System

TFS - Team Foundation Server, Team Foundation Version Control

Como o GIT funciona?

Gource

```
gource --camera-mode track --seconds-per-day 0.25 --user-image-dir ../avatars/ --hide filenames
```

## Quais podem ser os usos para infra?

Documentação com Markdown

Ex:

<https://github.com/manifestotech/manifestotech>

Alterações de Arquitetura com ARM Template

<http://armviz.io/>

Controle e alteração de código-fonte de sistemas hospedados conosco (mapas, proac, etc.)

Esteiras de automação

## Setup GIT e GITHUB - Renato

### Instalação git (Download e configuração)

No site do git, é efetuado o download do instalador, o qual irá instalar o Git Bash(Linha de comando) e Git GUI (Interface Gráfica)

Invés de utilizar o Git GUI, utilizaremos o GITHUB Desktop

#### Git for Windows

Git for Windows focuses on offering a lightweight, native set of tools that bring the full feature set of the Git SCM to Windows while providing appropriate user interfaces for experienced Git users and novices alike.

<https://gitforwindows.org/>



### Configurando Informações iniciais do Git

Na primeira utilização, precisamos configurar as informações de usuário

```
$ git config --global user.name "Seu nome"
$ git config --global user.email seuprefixo@exemplo.br
```

a opção `--global` informa que essas configurações são definidas em todo o sistema.

## Comandos Básicos GIT

### GIT INIT

Para iniciar o versionamento local, entre na pasta a qual contem os arquivos e inicie com :

```
git init
```

## GIT ADD

Seleciona (Termo 'STAGE') os arquivos que serão salvos.

```
git add .
```

ADD . (PONTO)

Informa que irá adicionar todos os arquivos

## GIT STATUS

Verifica as alterações feitas.

Abaixo um exemplo de situação antes de utilizar ADD

```
git status

Arquivos não monitorados:
  (utilize "git add <arquivo>..." para incluir o que será submetido)
  README.md

nada adicionado ao envio mas arquivos não registrados estão presentes (use "git add" to registrar)
```

Depois de feito o ADD

Será informado que já foi feito o add (STAGE), e que pode ser feito o commit

Não iremos falar do Restore por agora

```
git status

No ramo main
Mudanças a serem submetidas:
  (use "git restore --staged <file>..." to unstage)
  new file:   README.md
```

## GIT COMMIT

Commit ou “*Commitar*”, ele salva as alterações (com descrição) localmente.

“Como boa prática , é recomendado inserir até 50 caracteres como base do comentário do commit. No entanto, essa regra não precisa ser seguida a ferro e fogo, ok?”

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

[main 9c5473b] Descrição do commit
1 file changed, 2 insertions(+)
create mode 100644 README.md
```

Na primeira Linha o código na frente do main, é o código do commit.

O qual pode ser consultado no GIT LOG Depois

Nas linhas abaixo mostra os dados de linhas adicionadas ou removidas do código (O que foi modificado por linha)

insertions(+) | deletion (-)

E na última linha quais arquivos foram alterados/adicionados/removidos

Lembrando que caso utilize o -m “texto commit” e em seguida utilize outro -m “texto commit 2”

O Primeiro será determinado como Título, e o segundo Descrição

## GIT LOG

Esse comando mostra o histórico de alterações feitas por commit.

```
git log

commit 9c5473b7c90325fe2056d5acdc7a7d0a5b03984a #CODIGO DO COMMIT
Author: Renato Di Giacomo <renato.giacomo@bnpsolucoes.com.br> #VEM DO GIT CONFIG
Date: Wed Mar 2 14:54:28 2022 -0300 #DATA QUE FOI FEITO O COMMIT

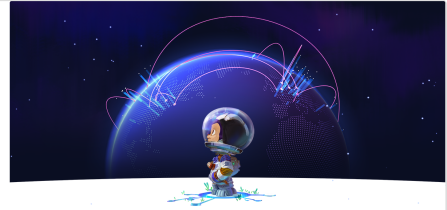
    Descrição do commit #DESCRIÇÃO DO COMMIT
```

# GITHUB E CONCEITO DE HOSPEDAGEM

GitHub: Where the world builds software

GitHub is where over 73 million developers shape the future of software, together. Contribute to the open source community, manage your Git repositories, review code like a pro, track bugs and features, power your

 <https://github.com/>



Sabemos que GIT é um sistema de controle de versionamento, o GITHUB é uma plataforma onde se armazena, controla, organiza projetos utilizando o sistema GIT e possibilita sair do local e ir para a web. Esse conceito é conhecido como Hospedagem de Repositório

GITHUB não é a única que existe, existem outras como o BITBUCKET, GITLAB e o próprio AzureDevOps da Microsoft.

Assim como qualquer plataforma, precisa criar a sua conta no servidor

Feito isso, poderá começar a utilizar o GITHUB

Agora vamos aprender alguns comandos para o envio e recebimentos do GITHUB

## GIT REMOTE

Ele faz conexão com a hospedagem de código fonte

```
git remote add origin https://github.com/<usuário>/<repositorio>.git
```

Origin = seu repositório local, ou o que foi feito de add e commit somente localmente

O link informa de forma HTTPS qual o repositório está sendo conectado

## GIT PUSH (EMPURRAR)

Ele envia as alterações feitas localmente para o GITHUB

Quando é enviado pela primeira vez:

```
git push origin main
```

Uma vez conectado e já feito a definição local para onde se utiliza somente:

```
git push
```

“Caso queira saber a situação, pode se utilizar GIT STATUS ou GIT LOG dependendo do que precisa”

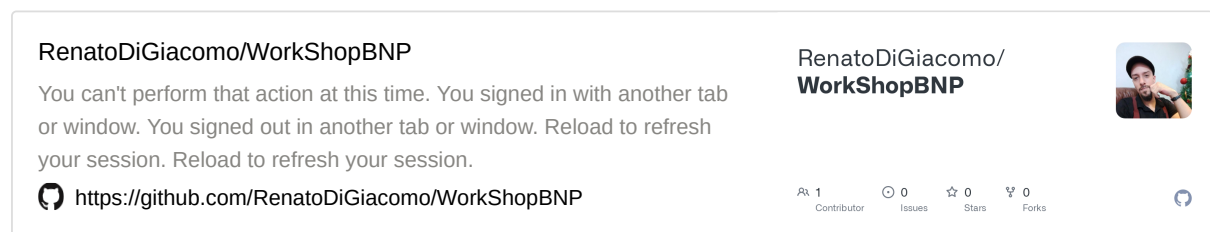
## GIT CLONE

Ele realiza um “DOWNLOAD” ou CLONE do repositório qual deseja.

```
git clone https://github.com/<usuario>/<repositorio>.git
```

Para aprender utilizem esse link :

<https://github.com/RenatoDiGiacomo/WorkShopBNP>



## GIT FETCH

Assim como o LOG e o STATUS mostram histórico, o FETCH atualiza ele com o histórico repositório o qual está selecionado (O qual foi feito o remote).

```
git fetch
```

Caso tenha algum erro será notificado. após o comando

## GIT PULL (PUXAR)

PULL (puxa) ou incorpora as mudanças do repositório remoto ao seu repositório local

```
git pull
```

## Subindo o primeiro Projeto

Git subindo o primeiro projeto

Git na prática -Parte 1 (Subindo projeto para o github).

Git é uma ferramenta de controle de versão, desenvolvida por nada menos que Linus Torvalds. Nessa primeira parte do post vou ensinar como criar um repositório e enviar um projeto para o GitHub. O que é um <https://medium.com/@rgdev/git-na-pr%C3%A1tica-parte-1-subindo-projet-o-para-o-github-133e294221ae>

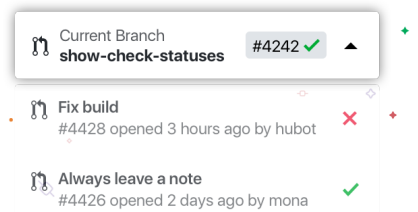


## GITHUB DESKTOP

GitHub Desktop

Checkout branches with pull requests and view CI statuses See all open pull requests for your repositories and check them out as if they were a local branch, even if they're from upstream branches or forks. See which

 <https://desktop.github.com/>

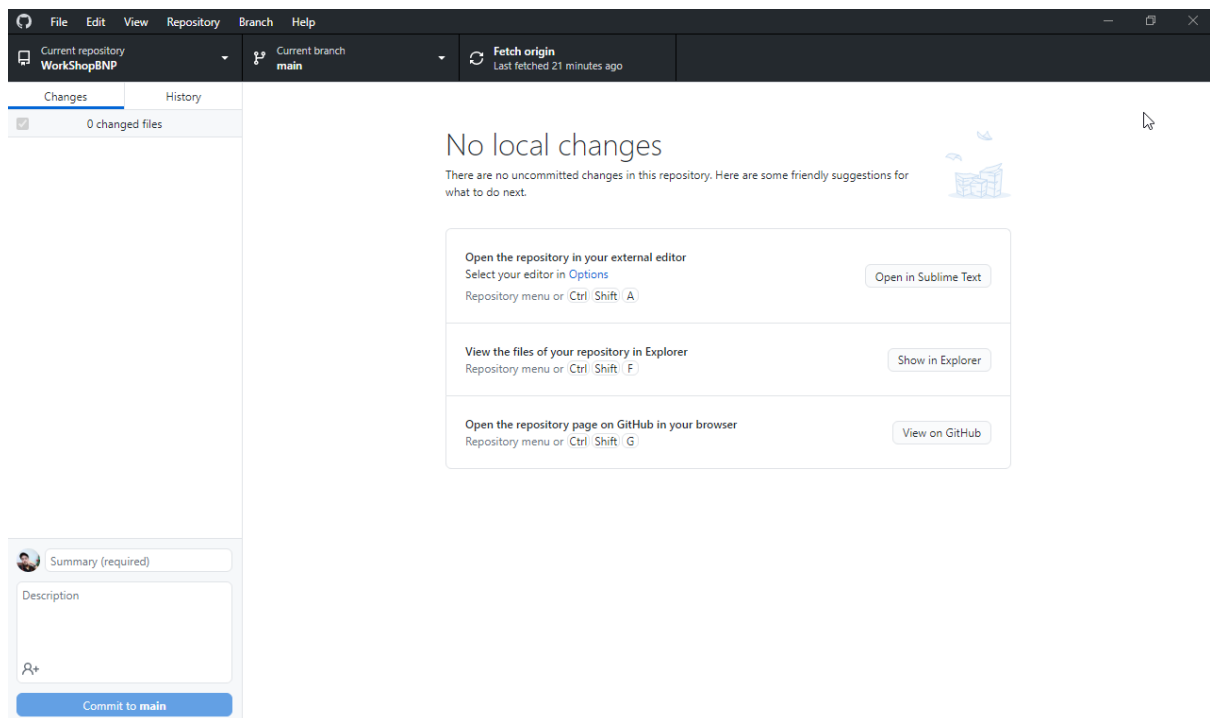


Até agora foram aprendidas os conceitos e comandos básicos para a administração de versionamento.

Porem é possível utiliza de forma visual essa administração

Como precisamos entender o conceito de versionamento e do sistema git, não podemos iniciar diretamente com a interface gráfica.

O GitHub Desktop, e uma ferramenta de alterativa de utilização e organização dos repositórios, de forma gráfica.



## Estratégias de branch - John

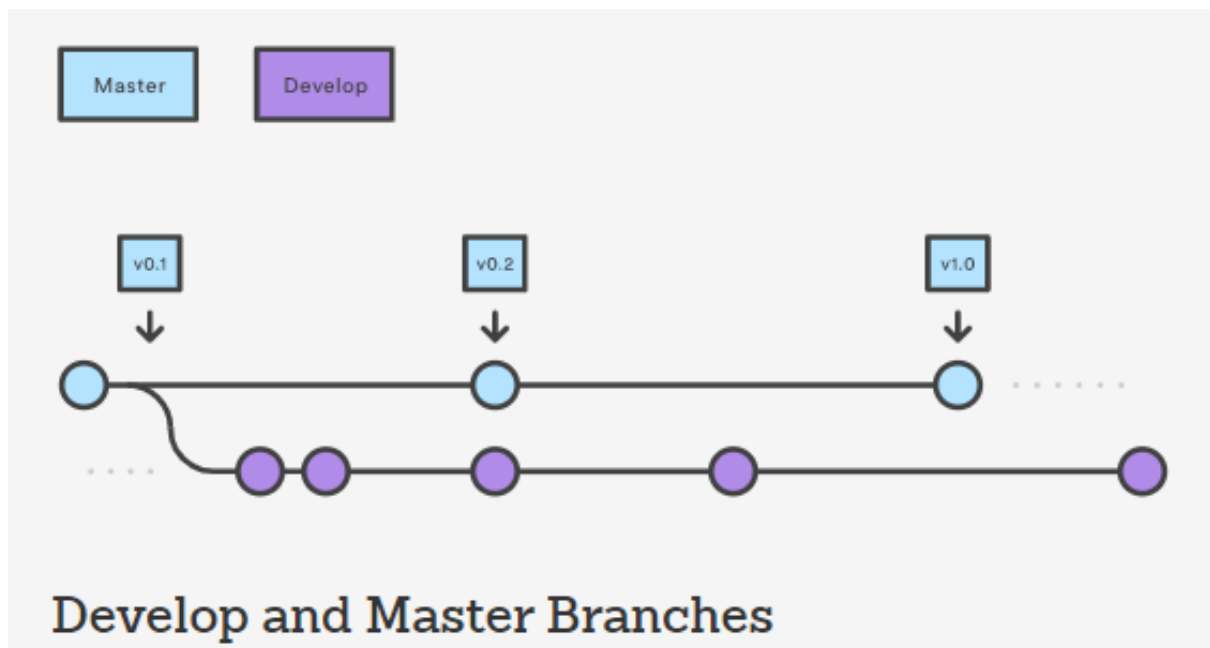
### Para que serve o branch?

Uma ramificação no git é uma linha do tempo das alterações feitas nos arquivos do projeto. É útil em situações nas quais você deseja adicionar um novo recurso ou corrigir um erro, gerando uma nova ramificação garantindo que o código instável não seja mesclado nos arquivos do projeto principal.

Por padrão, você sempre está trabalhando em um ramo no Git, e mesmo quando você não cria um *branch*, o Git cria automaticamente um *branch* chamada **master** (main) como **padrão**.

Na imagem abaixo podemos ver um exemplo de trabalho com vários ramos e *commits* aplicados. Veja que em alguns pontos da história os ramos são **unidos** para que as alterações de um ramo sejam aplicadas a outro.





## Branches principais

### ▼ Master (main)



Ramificação final e principal, esta é a branch onde ficam as versões finais de todas as alterações.

### ▼ Develop



É o ponto central de alterações, onde é centralizado todas as alterações e também serve como referência para criação de novas versões.

### Criando branch develop:

```
git branch develop
(apenas cria a branch)

# OR

git checkout -b develop
(cria a branch, e já entra nela)
```

---

## Branches de apoio

Depois das branches master e develop, o modelo mais conhecido de versionamento git usa uma variedade de outros branches de apoio pra facilitar as alterações em paralelo entre os membros da equipe, facilitar a identificação de recursos e para correção de rápida problemas. Ao contrário das branches principais esses possuem um tempo de vida limitado, uma vez que serão removidos posteriormente.

### ▼ Feature



Ramificação específica para novas alterações e implementações no projeto.

### Criando nova feature:

Importante salientar que uma feature só deve ser criada a partir da branch develop!

```
git checkout -b feature/nova-funcionalidade  
  
# Utilizando git flow  
git flow feature start nova-funcionalidade
```

### Finalizando funcionalidade:

```
git checkout develop  
git merge feature/nova-funcionalidade  
  
# Utilizando git flow  
git flow feature finish nova-funcionalidade
```

### ▼ Release



Ramificação para publicação de novas versões, nesta branch não pode haver novas features, apenas correções, geração de documentações e tarefas referentes a criação de novas versões.

### Criando nova release:

Igualmente a criação da feature, deve ser feita a partir da branch develop!

```
git checkout develop
git checkout -b release/0.1.0

#Utilizando git flow
git flow release start 0.1.0
```

### Finalizando branch de release:

```
git checkout master
git merge release/0.1.0

# Utilizando git flow
git flow release finish 0.1.0
```

## ▼ Hotfix



Este tipo de ramificação é específico para correções em produção.

Ao identificar um problema na versão atual de um projeto, devemos criar uma branch hotfix para corrigir o problema e já lançar uma nova versão direto para a **master** (main).

### Criando branch hotfix:

Esta branch deve ser criada a partir da branch **master** (main)!

```
git checkout master
git checkout -b hotfix/correcao

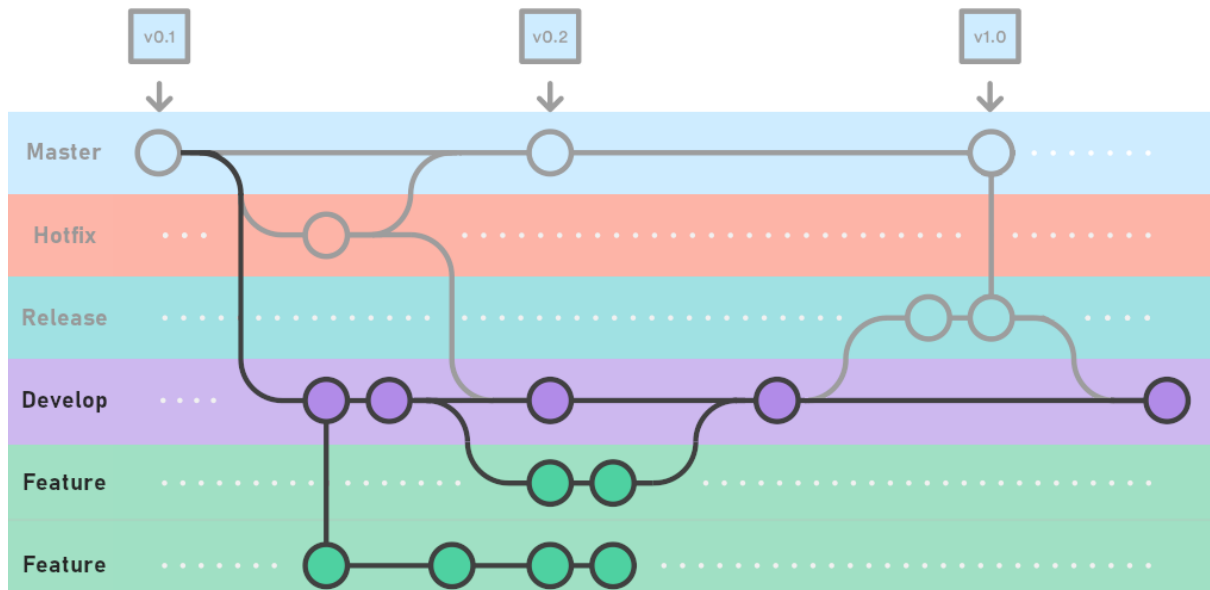
#Utilizando git flow
git flow hotfix start correcao
```

### Finalizando branch hotfix:

```
git checkout master
git merge hotfix/correcao
```

```
git checkout develop
git merge hotfix/correcao

# Utilizando git flow
git flow hotfix finish correcao
```



## Git flow

Nos exemplos acima, foi mostrado gerenciamento de branch utilizando a estratégia trabalho Git Flow. Atualmente a mais conhecida e usada para versionamentos git.

### Fluxo de trabalho de Gitflow | Atlassian Git Tutorial

O Gitflow é um fluxo de trabalho legado do Git que no começo era uma estratégia inovadora e revolucionária para gerenciar ramificações do Git. O Gitflow perdeu popularidade para fluxos de trabalho baseados em troncos, que hoje são considerados práticas recomendadas para o desenvolvimento moderno e contínuo de softwares e práticas de

<https://www.atlassian.com/br/git/tutorials/comparing-workflows/gitflow-workflow>

## Resources



## Saiba mais

[https://www.youtube.com/watch?v=6Czd1Yetaac&t=678s&ab\\_channel=FabioAkita](https://www.youtube.com/watch?v=6Czd1Yetaac&t=678s&ab_channel=FabioAkita)

### Git

Git is easy to learn and has a tiny footprint with lightning fast performance. It outclasses SCM tools like Subversion, CVS, Perforce, and ClearCase with features like cheap local branching, convenient


 <https://git-scm.com/>

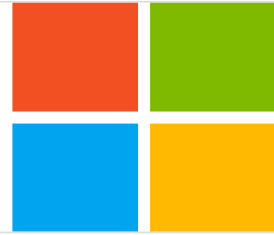


## Tech Talk: Linus Torvalds on git - YouTube

### Templates overview - Azure Resource Manager

With the move to the cloud, many teams have adopted agile development methods. These teams iterate quickly. They need to repeatedly deploy their solutions to the cloud, and know their infrastructure is in a reliable

 <https://docs.microsoft.com/en-us/azure/azure-resource-manager/templates/overview>



### O que é o TFVC

<http://www.linhadecodigo.com.br/artigo/1317/controlando-versoes-team-foundation-version-control-tfvc-check-out-e-workspace.aspx>

### O que é Concurrent Version System

<https://pt.wikipedia.org/wiki/CVS>

### Gitflow

[Fluxo de trabalho de Gitflow](#) | [Atlassian Git Tutorial](#)

### Outras estratégias de branch

[Diretrizes de ramificação do Git - Azure Repos](#) | [Microsoft Docs](#)

### Conceito e Comandos (MERGE):

<https://dev.to/eduardoopv/conceito-de-merge-git-e-github-4j0g>