

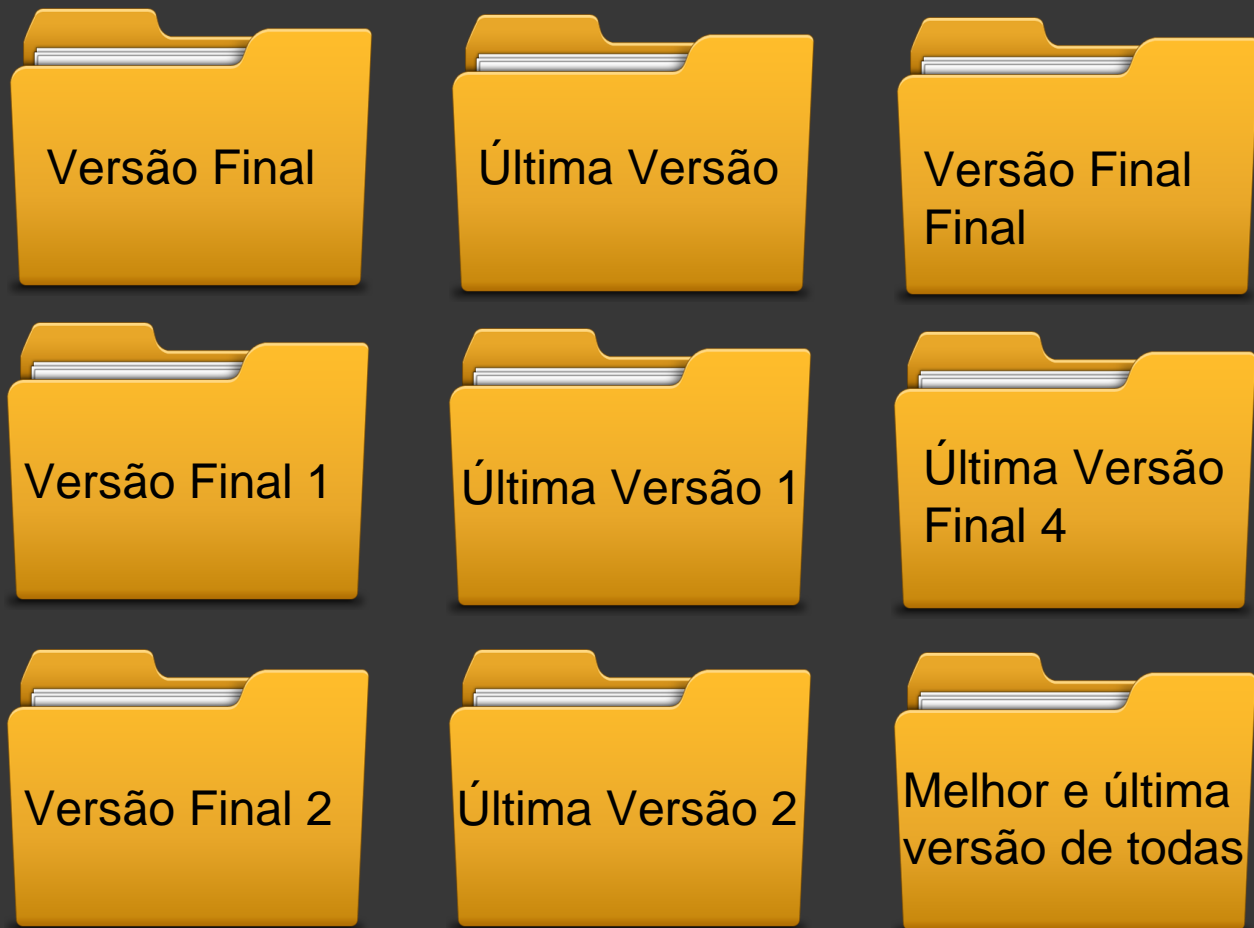


# GIT

**Prof. Me. Maicon dos Santos**  
**Senac - TDS**

Como você  
organiza suas  
mudanças?





Esse é um modelo que muitas pessoas utilizando para armazenar e copiar arquivos movimentando os mesmos para outros diretórios. Esta abordagem é muito comum porque é muito simples, **mas incrivelmente propensa a erros.**

# CONTROLE DE VERSÃO

**Um pouco  
de  
história...**

## Uma Breve História do GIT

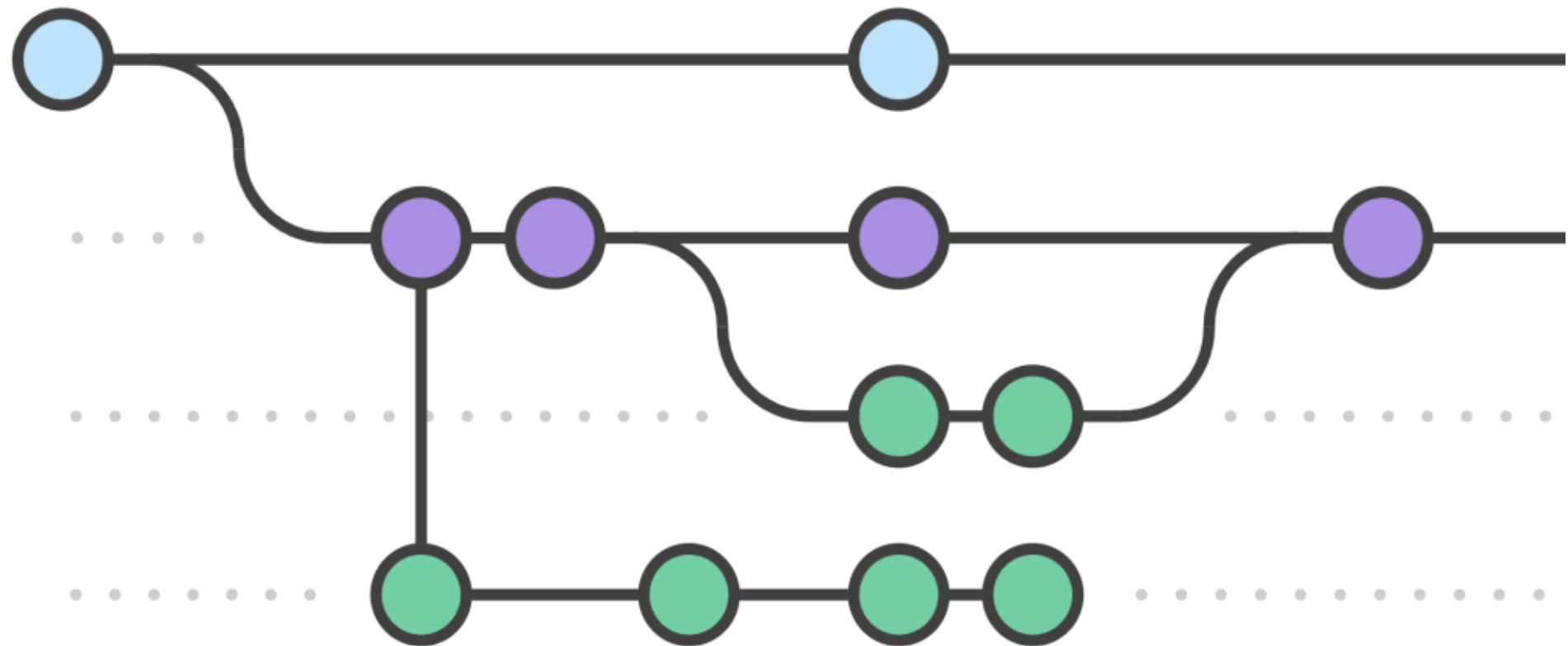
O desenvolvimento do Git surgiu após vários desenvolvedores do kernel (núcleo) do Linux decidirem desistir de acessar ao sistema do BitKeeper.

Desenvolvido/Criado em 2005 pelo Linus Torvalds (o homem conhecido por criar o núcleo, ou kernel, do SO Linux).

Desde sua criação, a ferramenta evoluiu e amadureceu para **ser fácil de usar** e ainda reter essas qualidades iniciais.

Incrivelmente **rápido**, e **muito eficiente** para projetos grandes, essa ferramenta tem um incrível sistema de ramos para **desenvolvimento não linear**.





Os VCS às vezes são conhecidos como ferramentas SCM (Gerenciamento de código-fonte) ou RCS (Sistema de controle de revisão).

**Uma das ferramentas VCS mais populares em uso hoje é chamada de Git.**

## Companies & Projects Using Git

Google

facebook

Microsoft

twitter

LinkedIn

NETFLIX



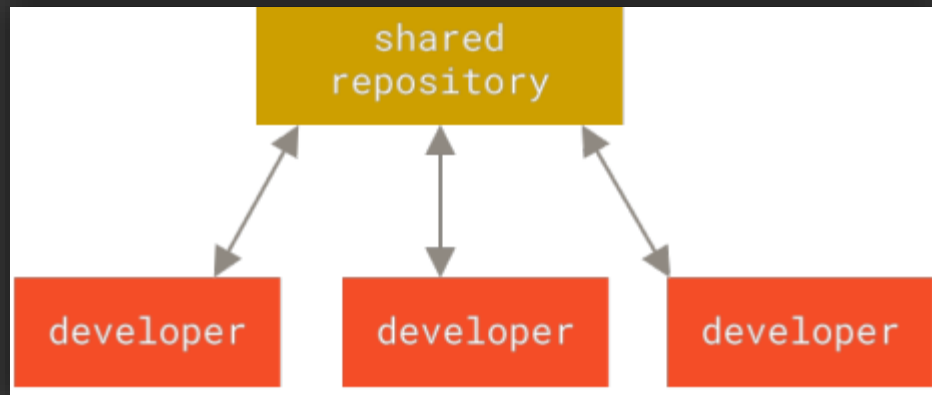
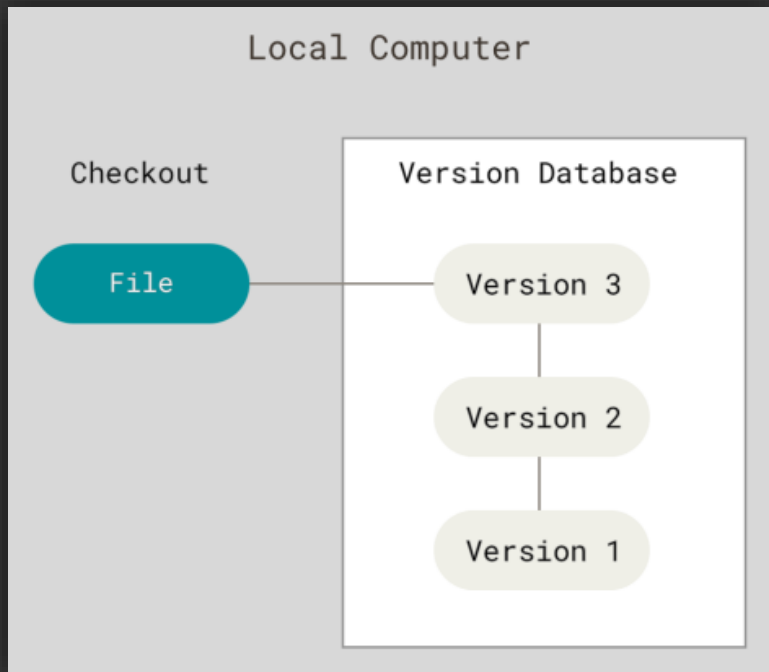
PostgreSQL



# Controle de Versão

**Controle de versão** é um sistema que registra alterações em um arquivo ou conjunto de arquivos ao longo do tempo para que você através das versões que são geradas.

# Controle de Versão Centralizado



No entanto, esta configuração também tem algumas desvantagens graves.

O mais óbvio é o ponto único de falha que o servidor centralizado representa.

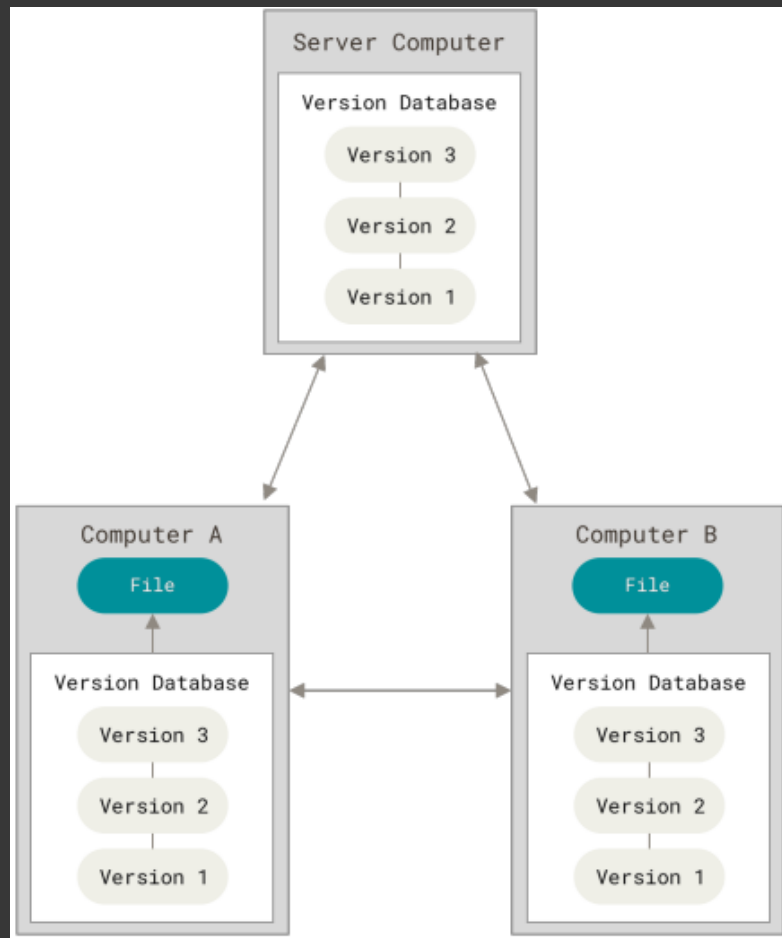
Se o servidor der problema por uma hora,  
**ninguém pode colaborar ou salvar as alterações de versão.**



Se o disco rígido do banco de dados central for corrompido, e backups apropriados não foram mantidos, **você perde absolutamente tudo, toda a história do projeto.**



# Controle de Versão Distribuído



Um **controle de versão distribuído** permite que você configure vários tipos de fluxos de trabalho, colaborar com diferentes repositórios, que podem inclusive ser restaurados em caso de perda.

## Centralized

Central Repo



Working Copy



Working Copy

## Distributed

Full Repo



Full Repo



Full Repo

# Instalação do GIT



## ❑ Instalação do Git

- ❑ Para utilizarmos o Git, é fundamental torná-lo disponível em seu computador.
- ❑ <https://git-scm.com/download>
- ❑ Escolha o SO e mãos à obra!

# Downloads



macOS



Windows



Linux/Unix

Older releases are available and the [Git source repository](#) is on [GitHub](#).



Temos a possibilidade de utilizar GIT em qualquer SO

# Download for Windows

[Click here to download](#) the latest (**2.40.1**) **64-bit** version of **Git for Windows**. This is the most recent [maintained build](#). It was released **about 1 month ago**, on 2023-04-25.

## Other Git for Windows downloads

### Standalone Installer

[32-bit Git for Windows Setup.](#)

[64-bit Git for Windows Setup.](#)

### Portable ("thumbdrive edition")

[32-bit Git for Windows Portable.](#)

[64-bit Git for Windows Portable.](#)

### Using winget tool

Install [winget tool](#) if you don't already have it, then type this command in command prompt or Powershell.

```
winget install --id Git.Git -e --source winget
```

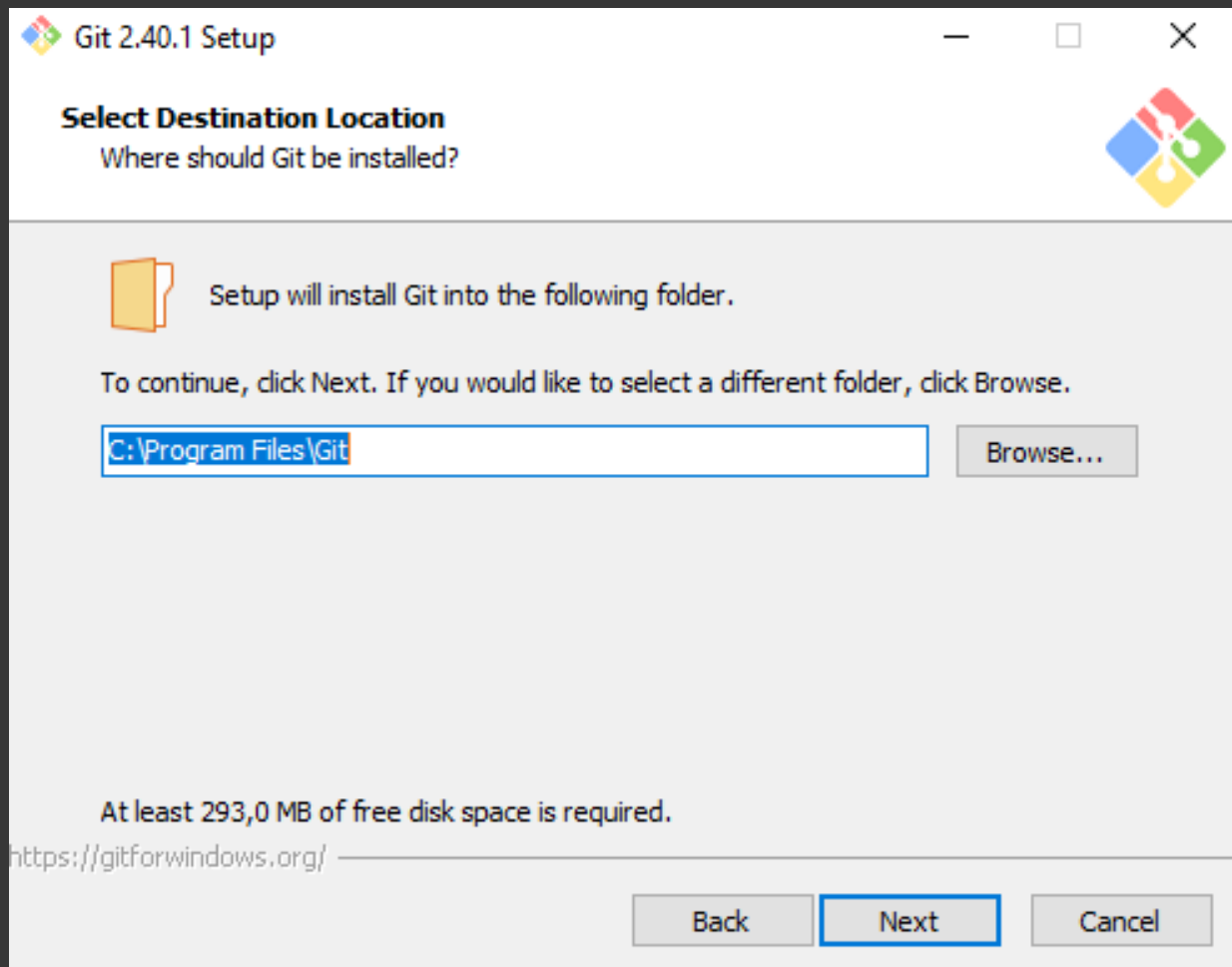
The current source code release is version **2.40.1**. If you want the newer version, you can build it from [the source code](#).

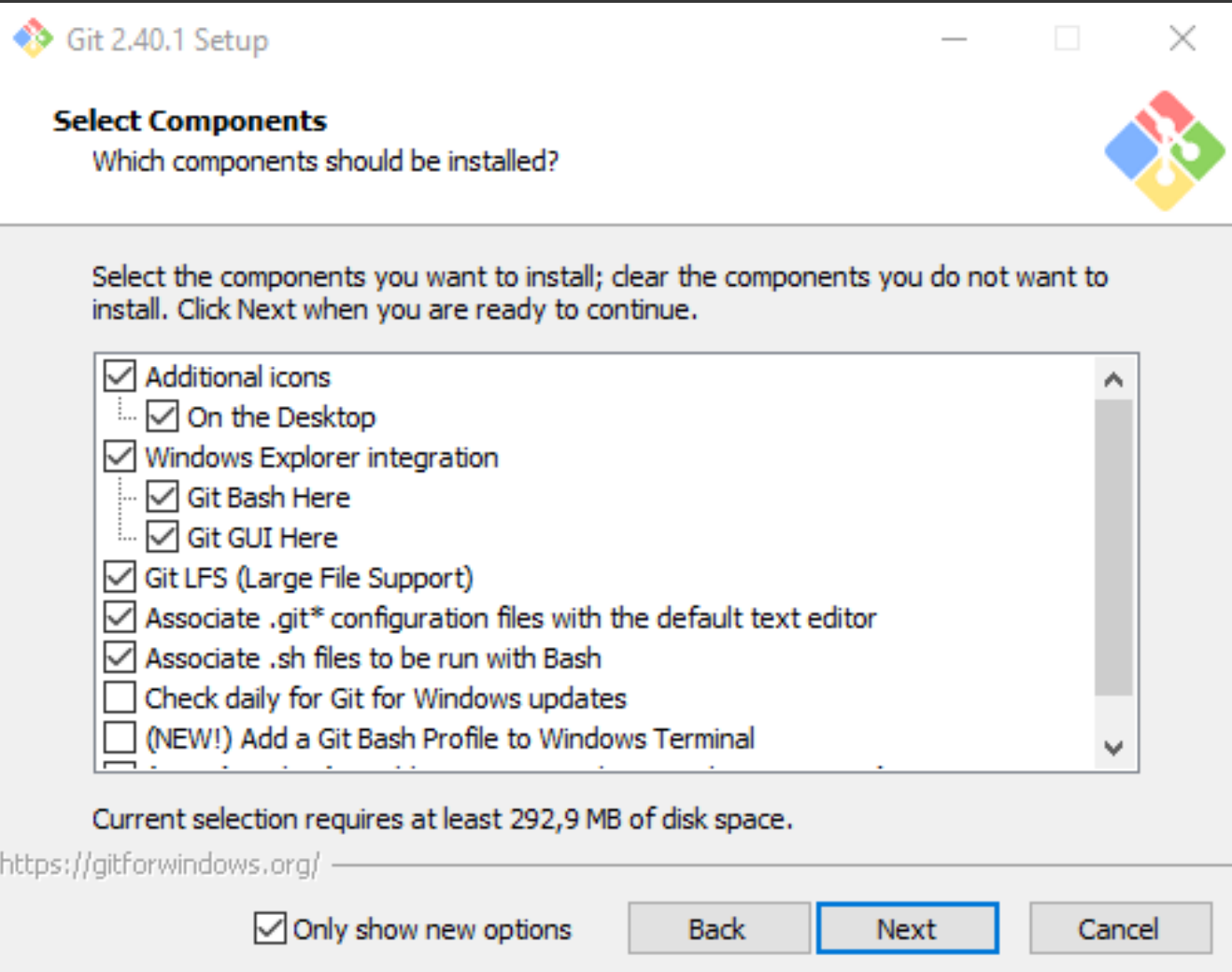
Escolher a forma de instalação do GIT para Windows.

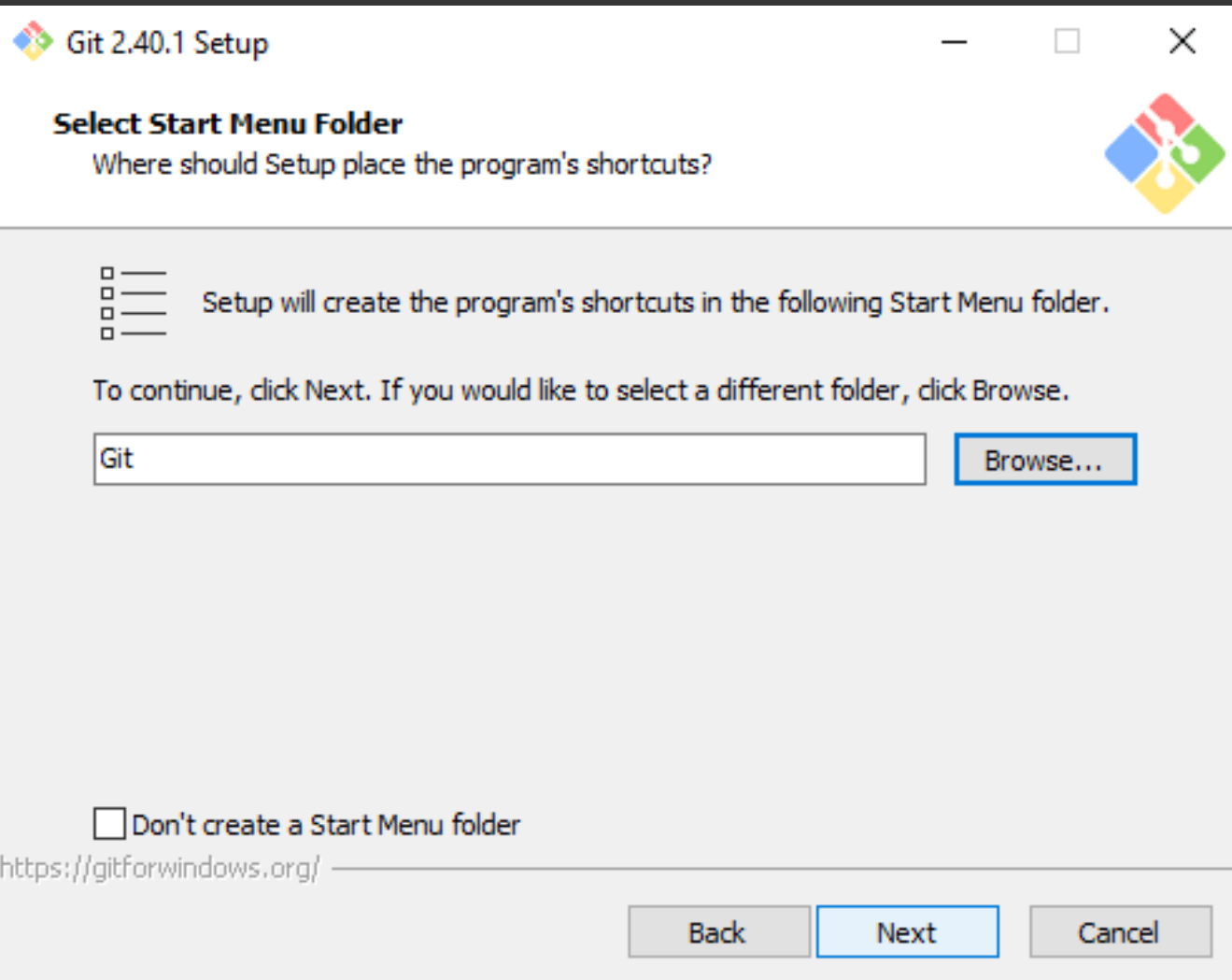


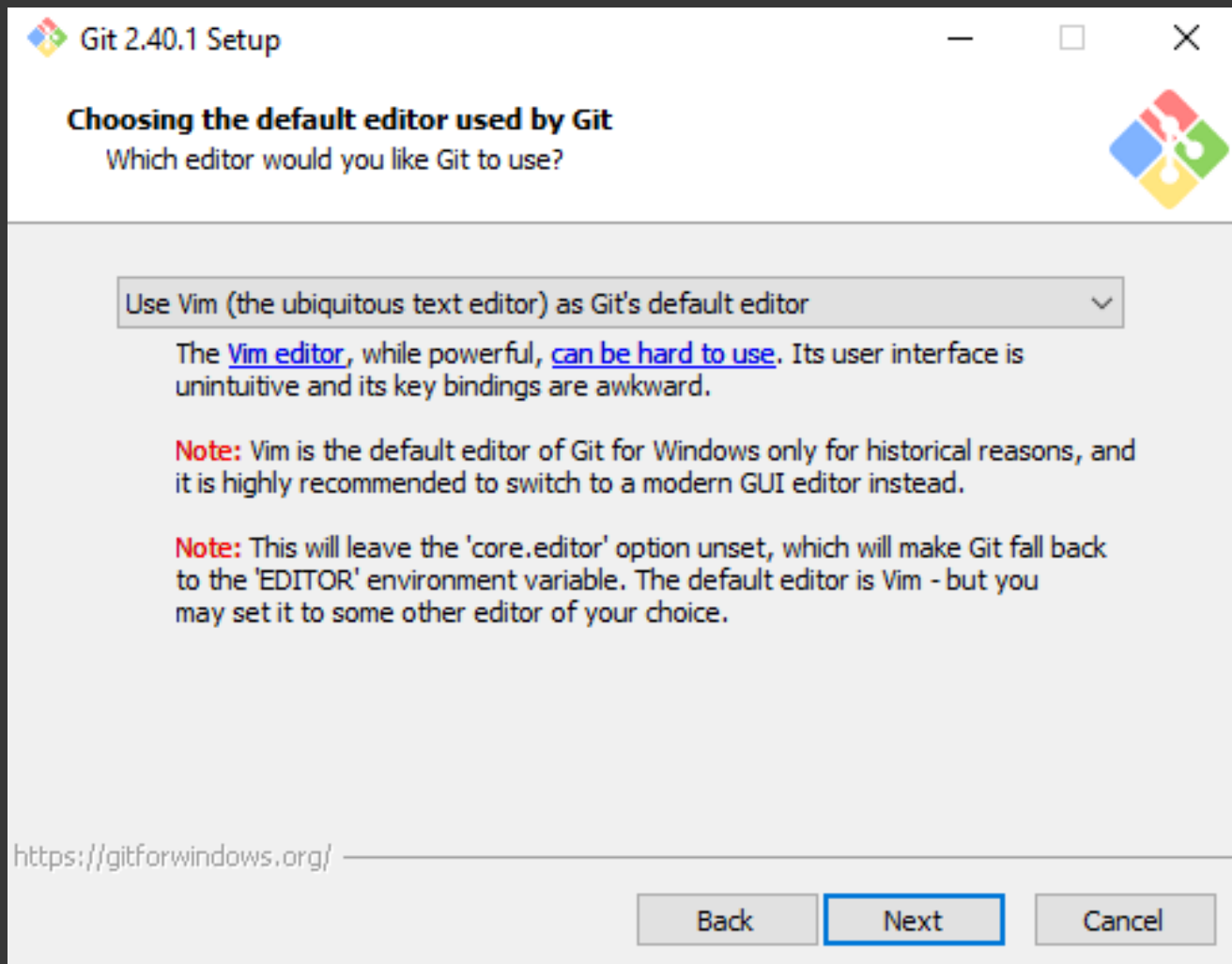
Basta seguir as instruções na tela, clicando em **Next**.  
Ao término, clique em **Finish** para concluir com êxito a  
instalação.

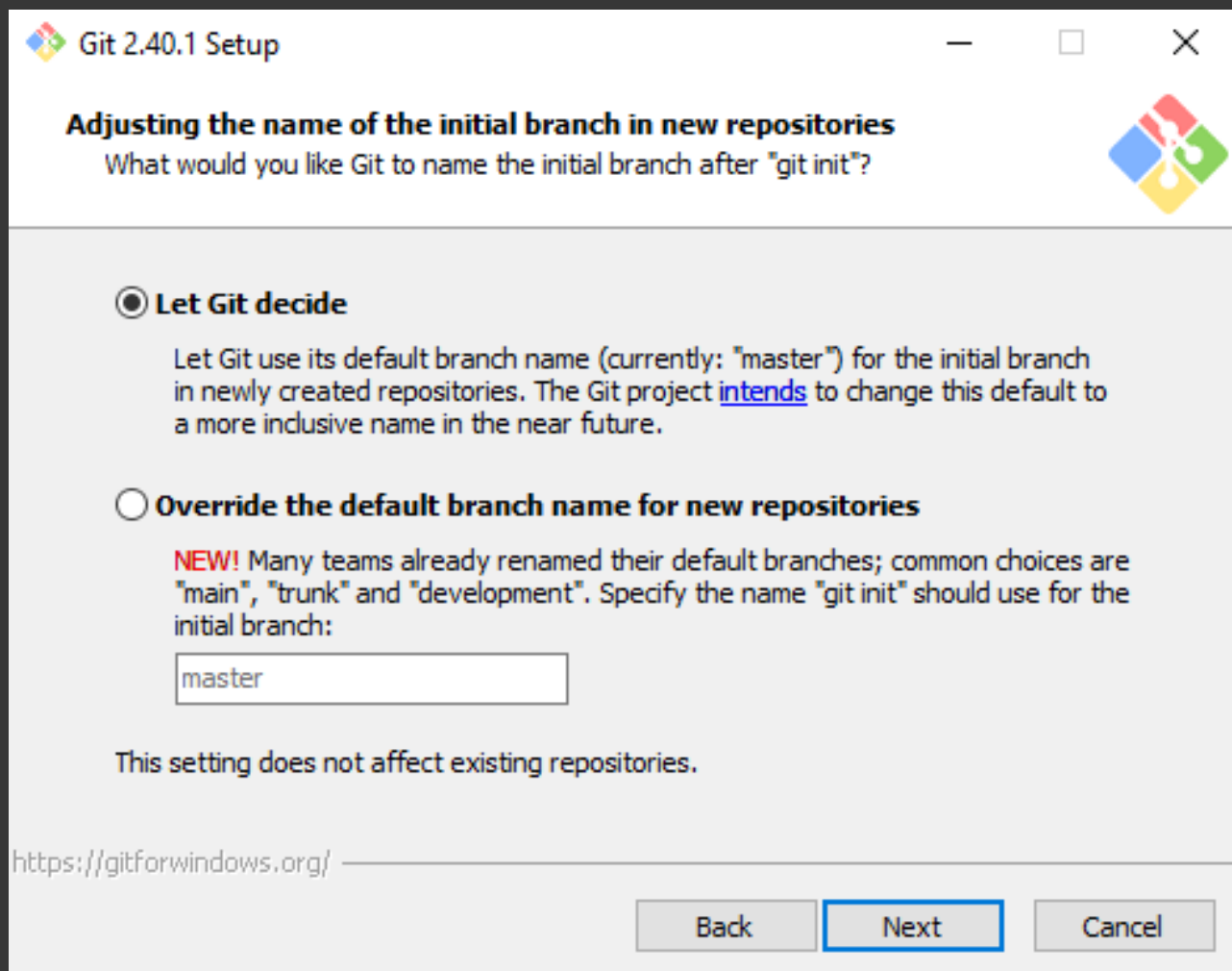


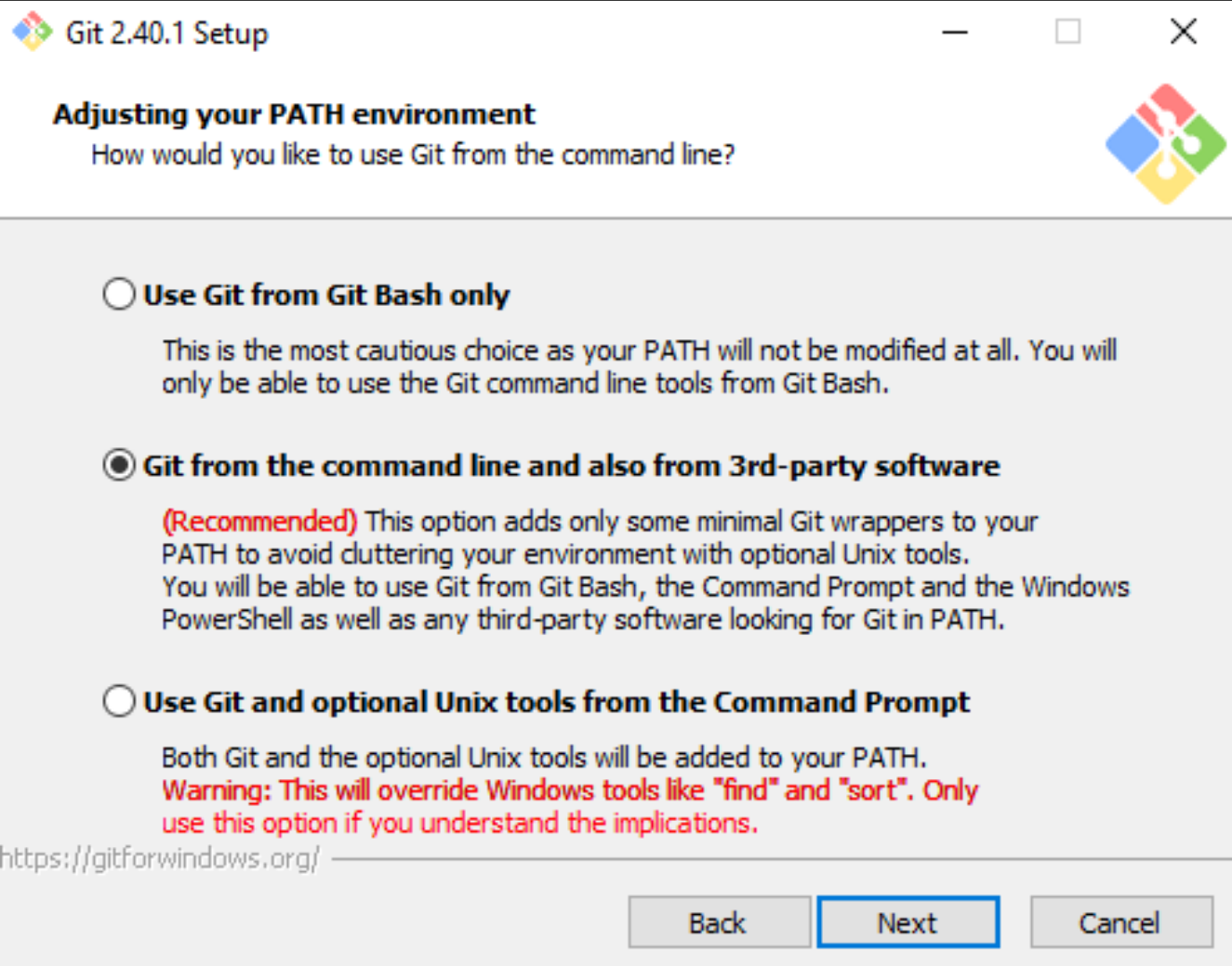


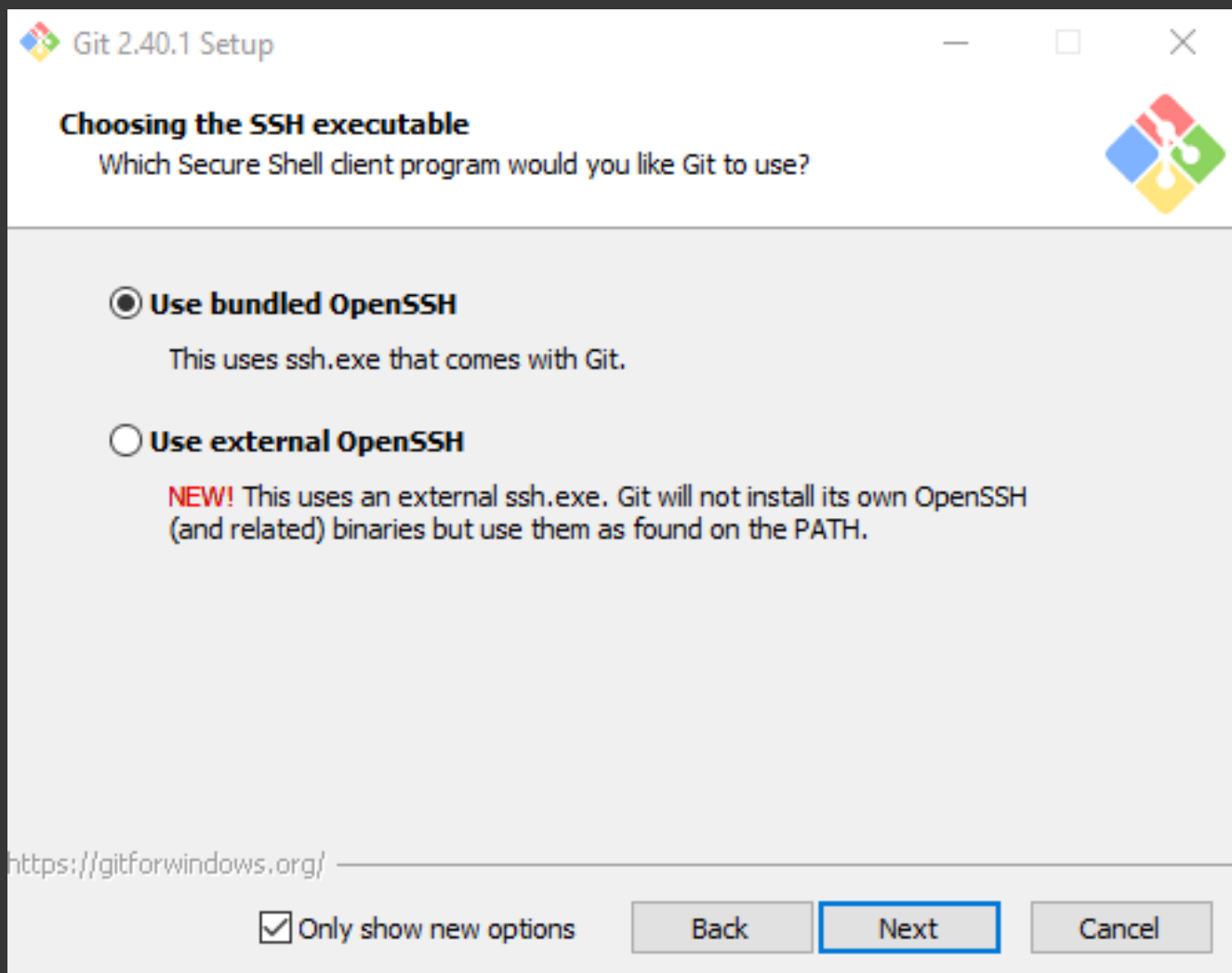


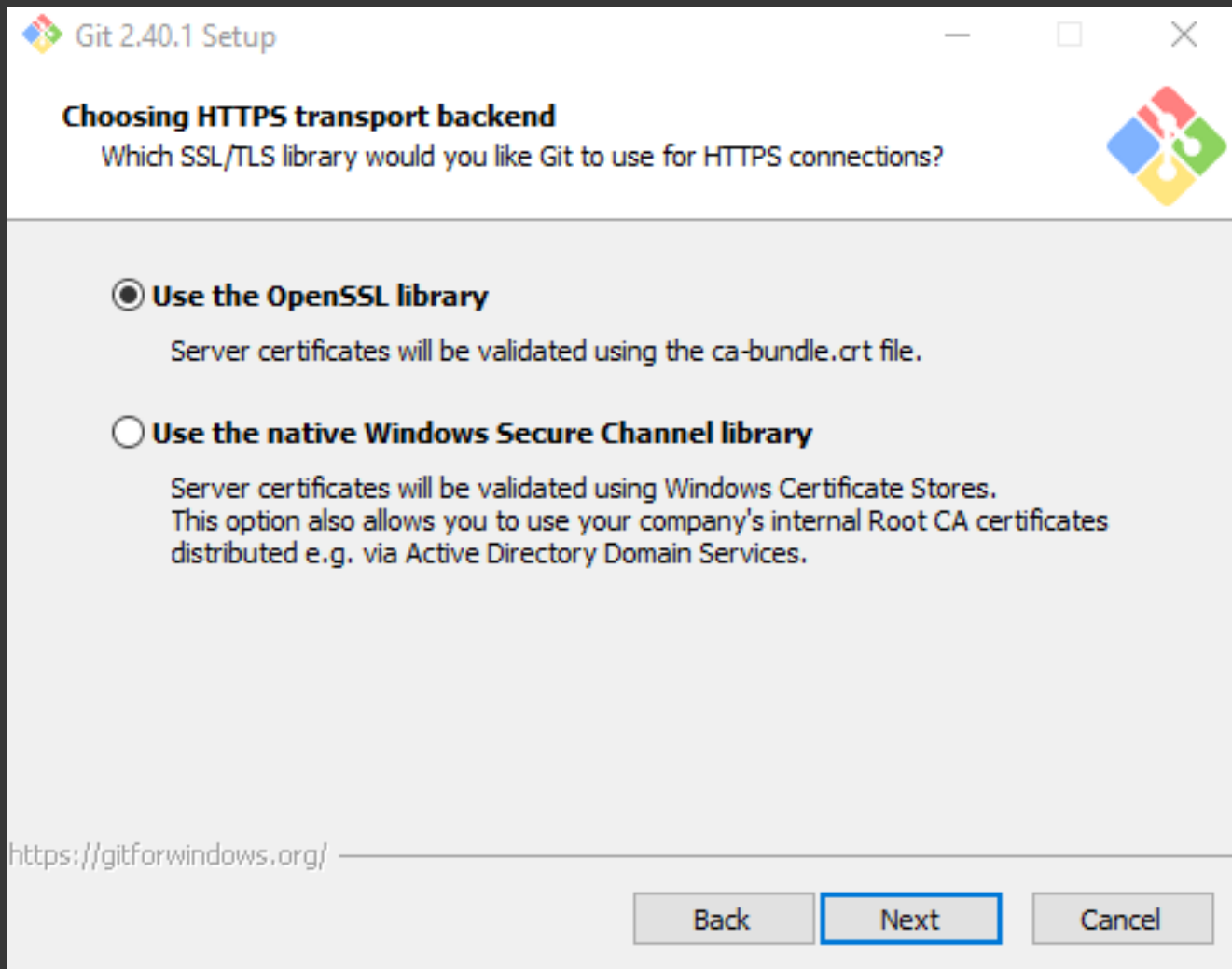




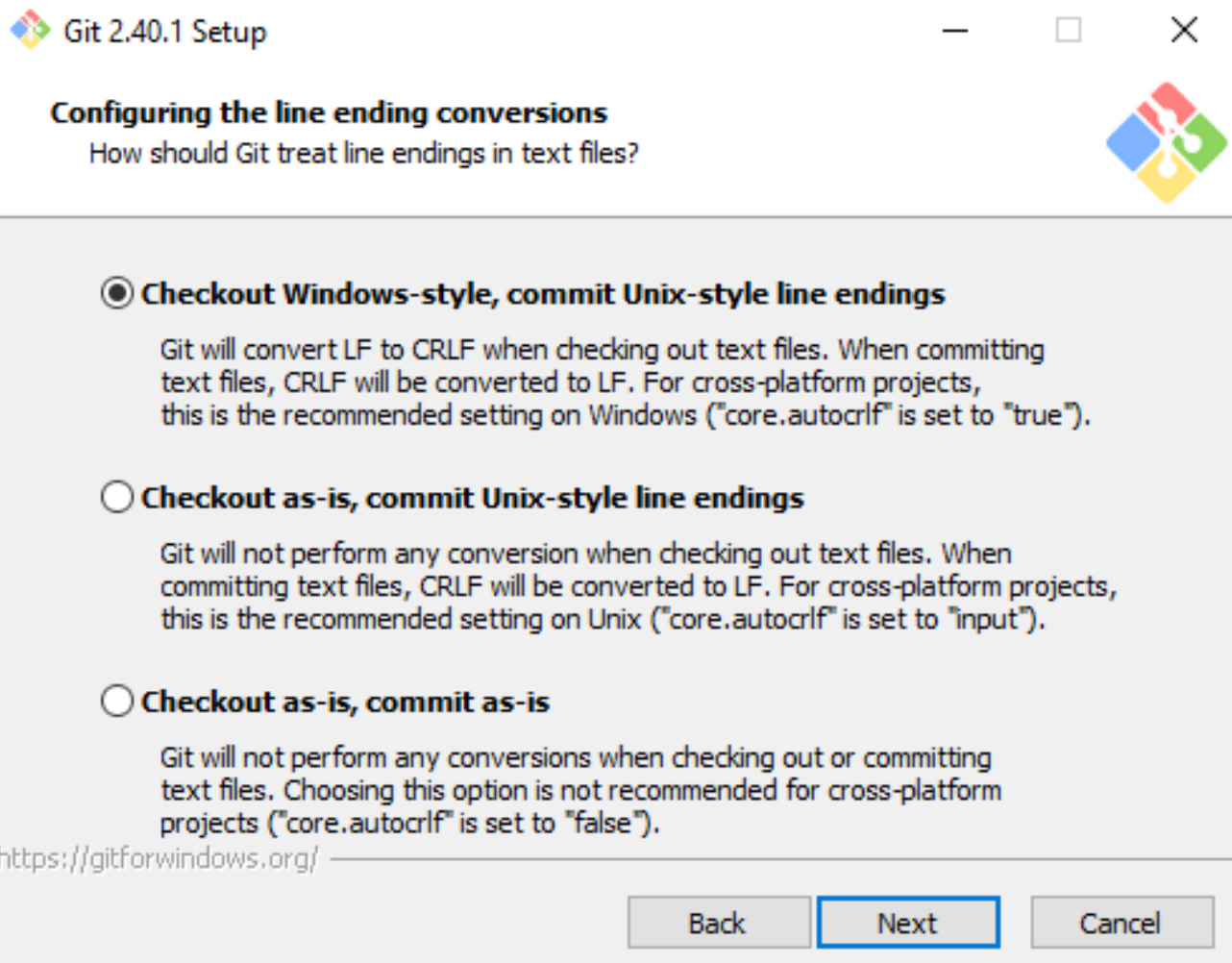












## Configuring the terminal emulator to use with Git Bash

Which terminal emulator do you want to use with your Git Bash?



☒ **Use MinTTY (the default terminal of MSYS2)**

Git Bash will use MinTTY as terminal emulator, which sports a resizable window, non-rectangular selections and a Unicode font. Windows console programs (such as interactive Python) must be launched via `wintty` to work in MinTTY.

☐ **Use Windows' default console window**

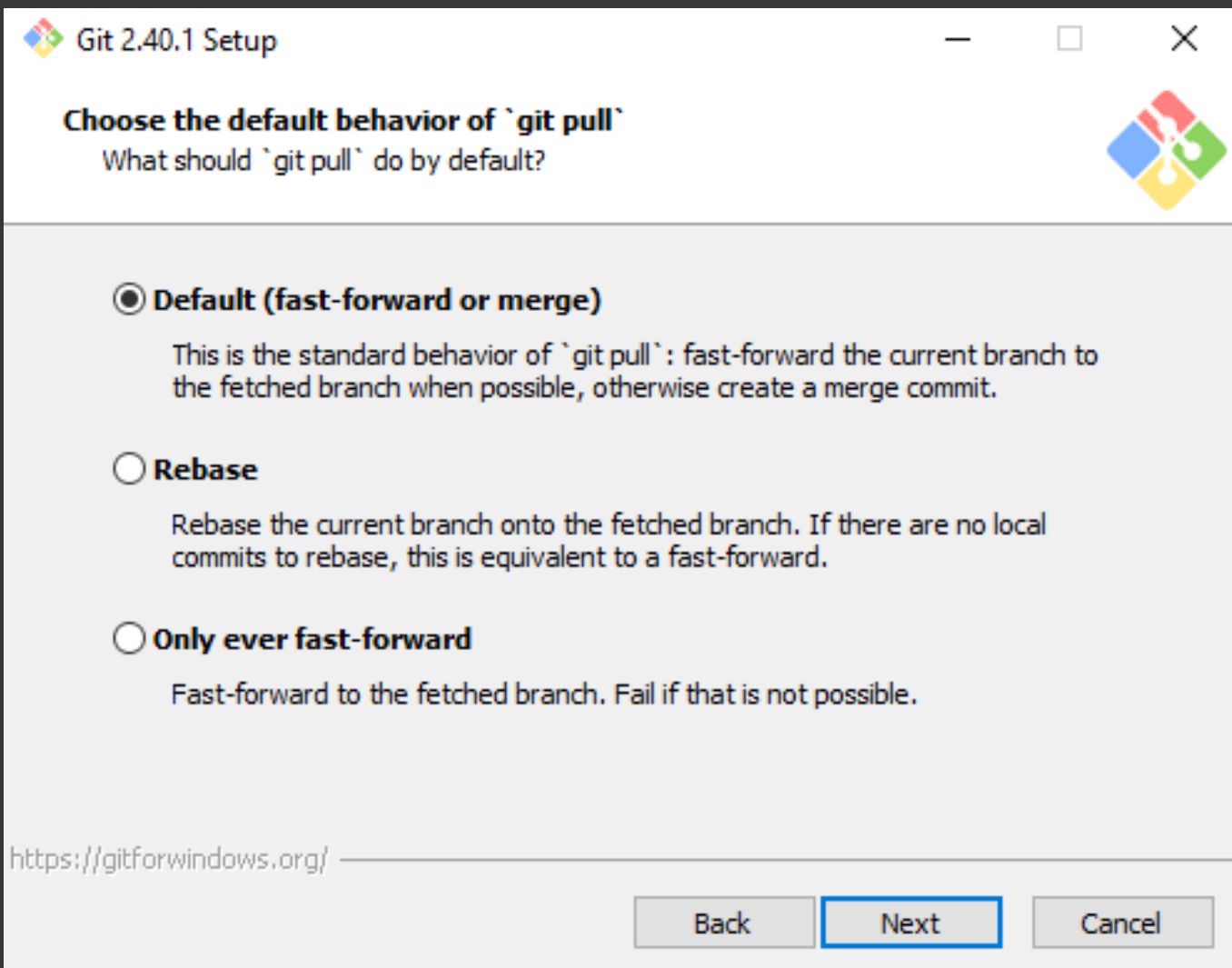
Git will use the default console window of Windows (`cmd.exe`), which works well with Win32 console programs such as interactive Python or `node.js`, but has a very limited default scroll-back, needs to be configured to use a Unicode font in order to display non-ASCII characters correctly, and prior to Windows 10 its window was not freely resizable and it only allowed rectangular text selections.

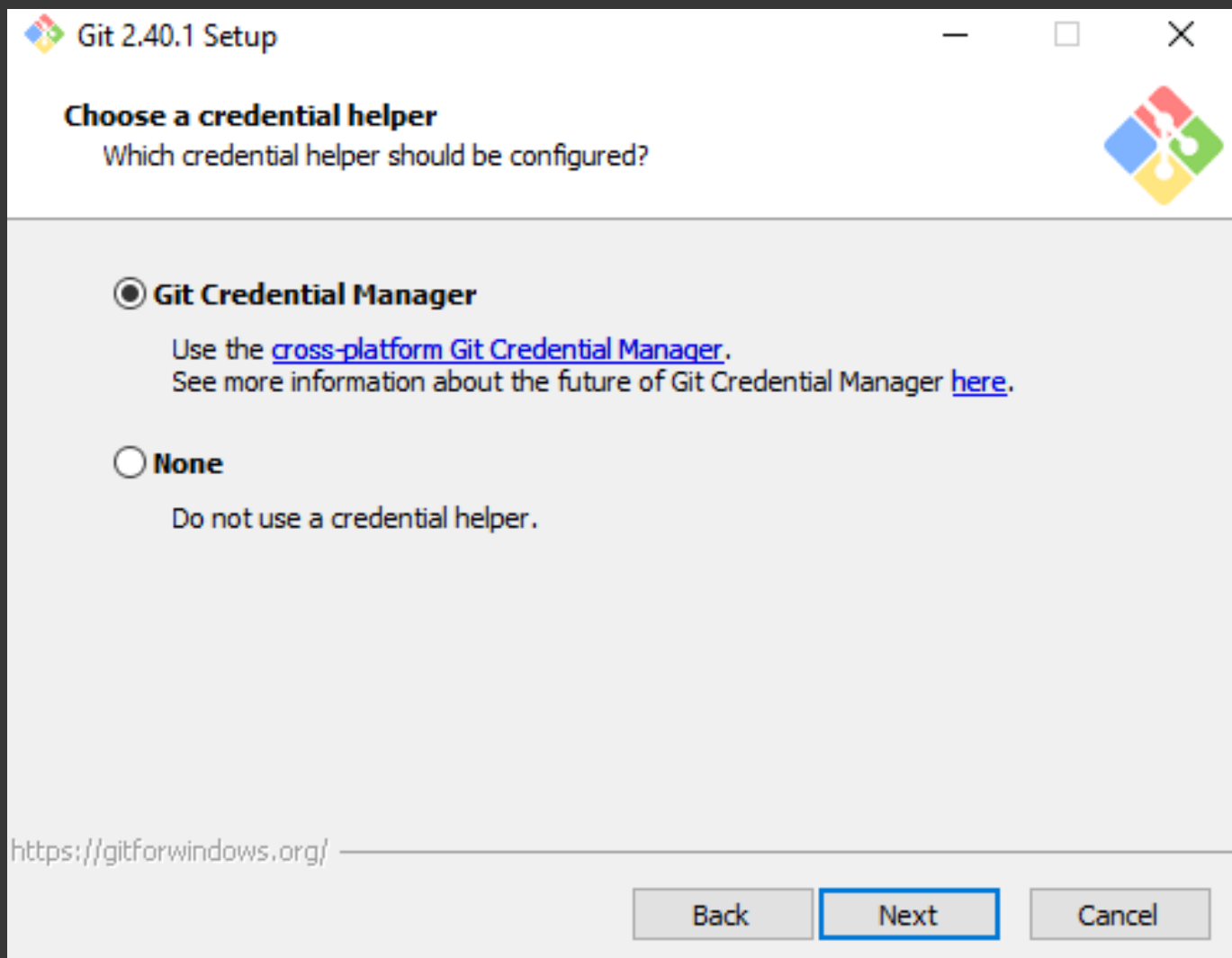
<https://gitforwindows.org/>

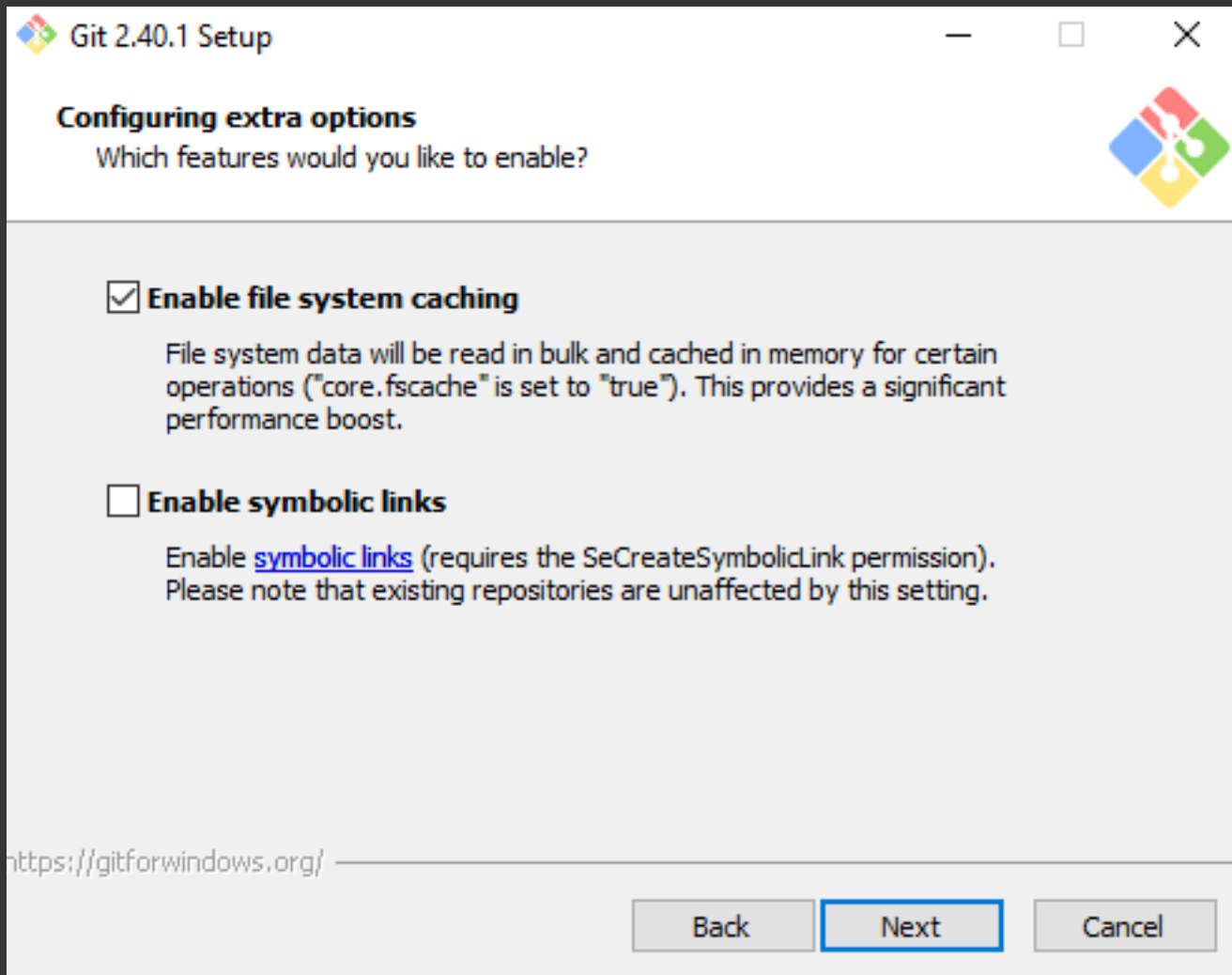
Back

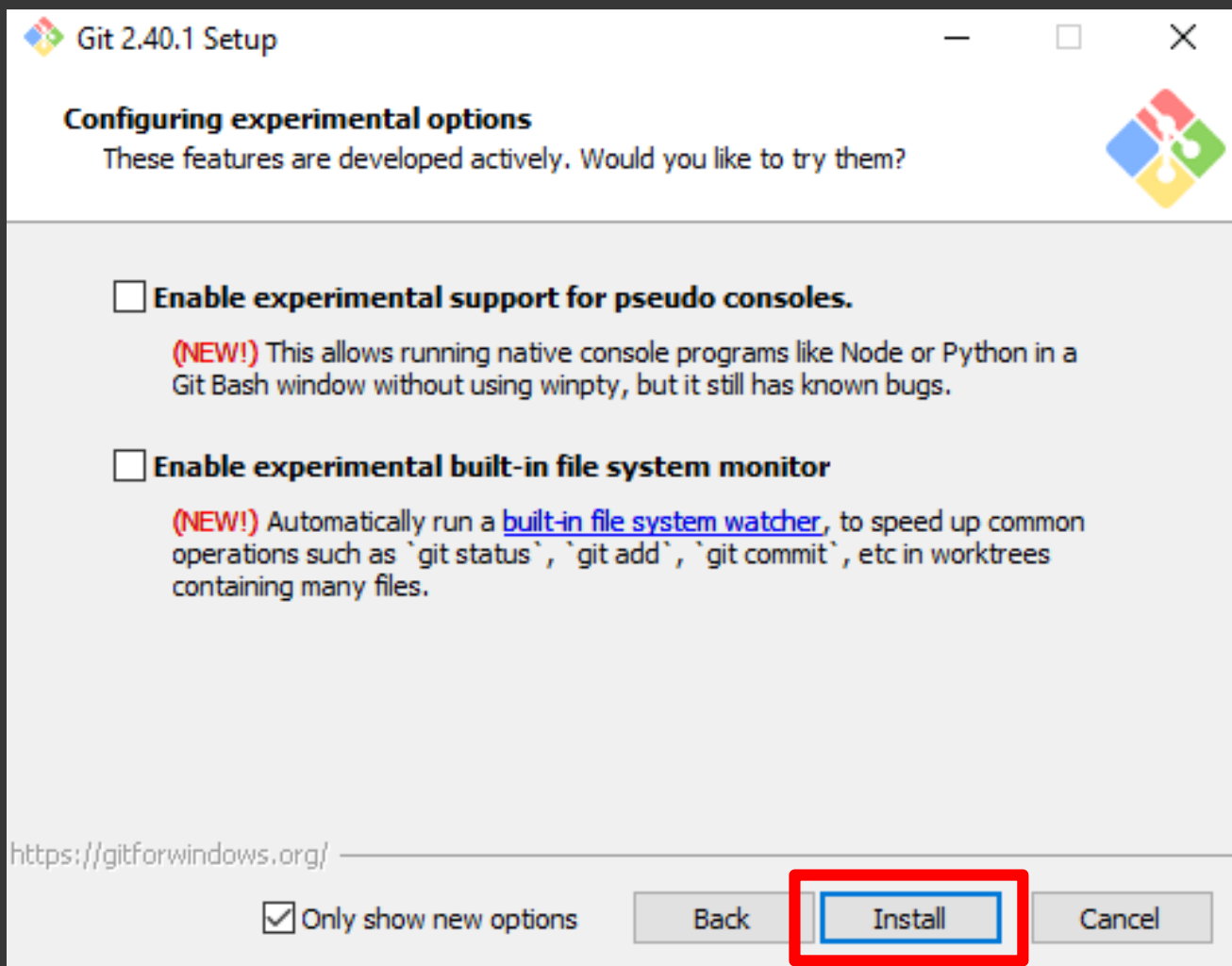
Next

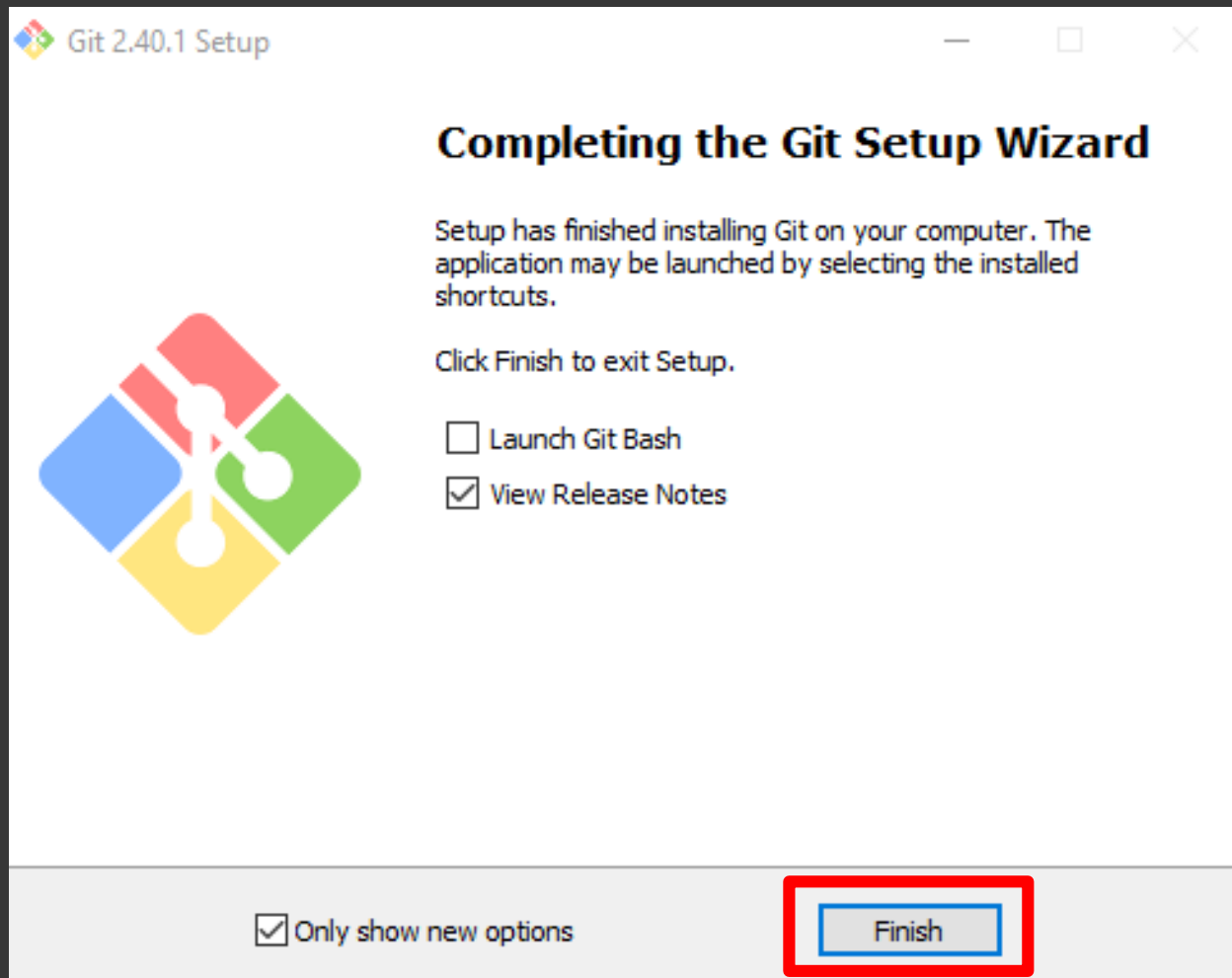
Cancel











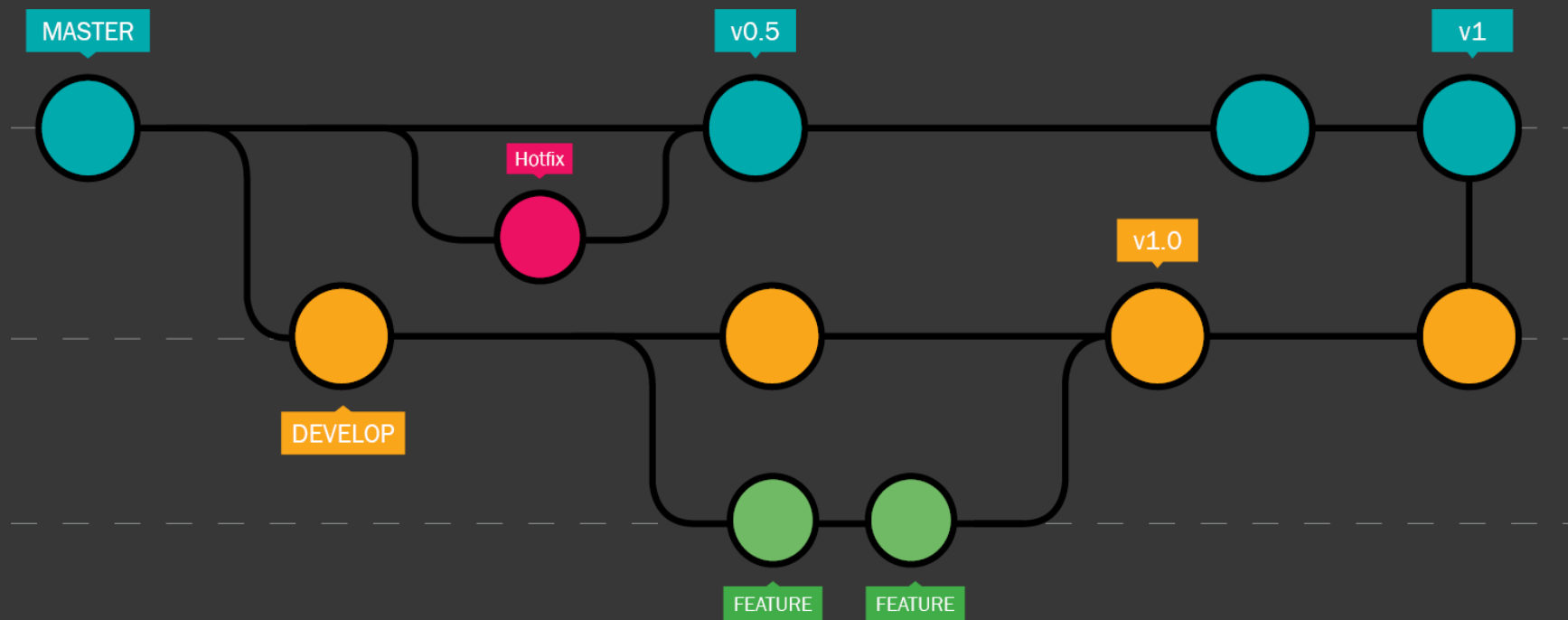
- Uma vez que o GIT está instalado e configurado no seu dispositivo, vamos explorar os conceitos básicos do GIT e como começar a usar o GIT.

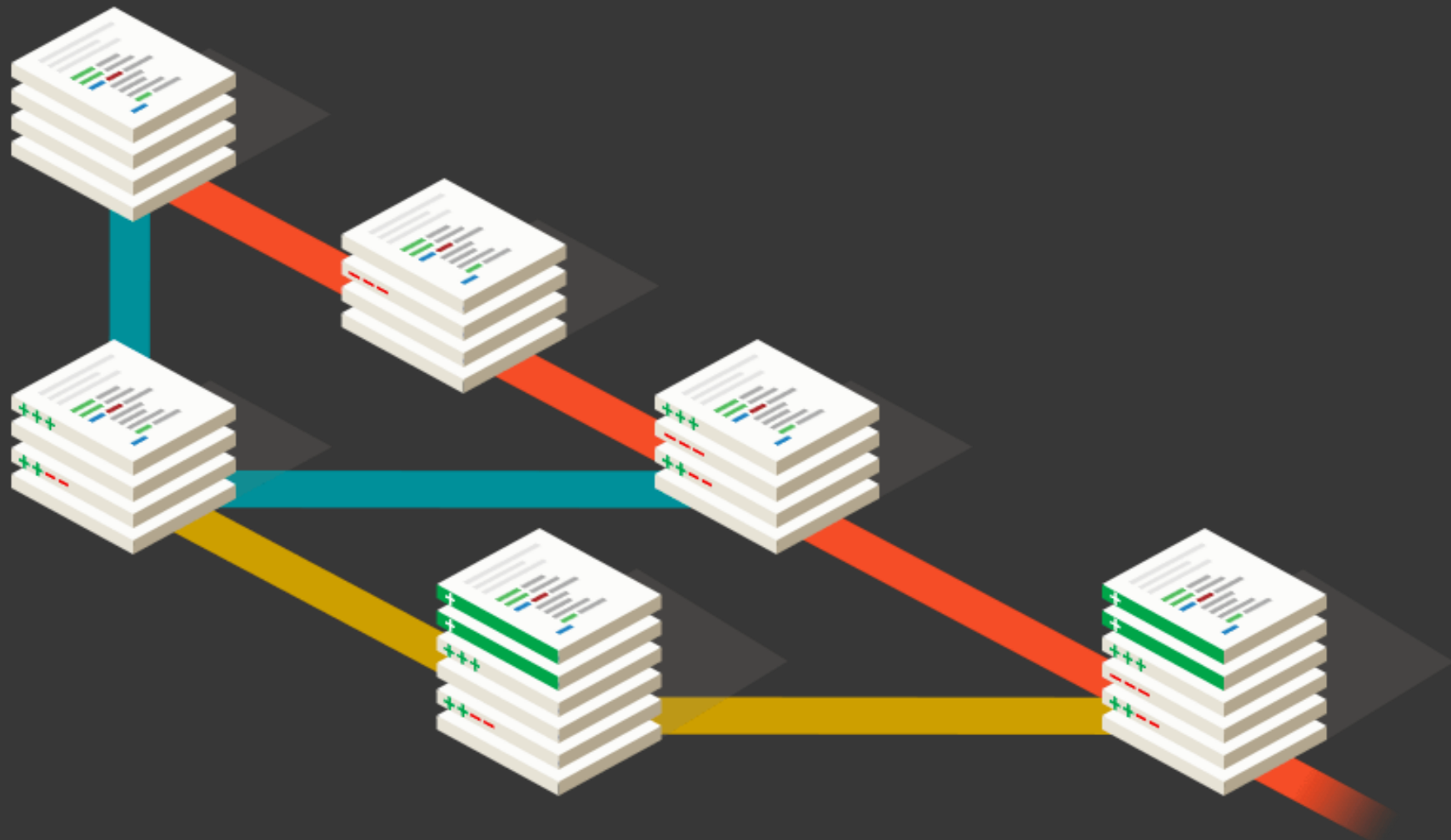


# Fluxo de trabalho GIT

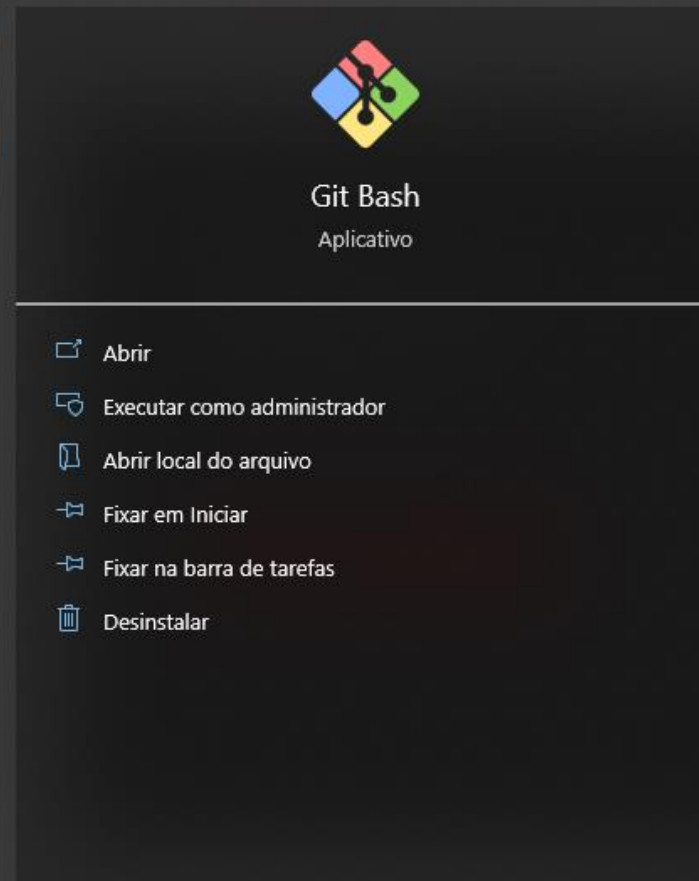
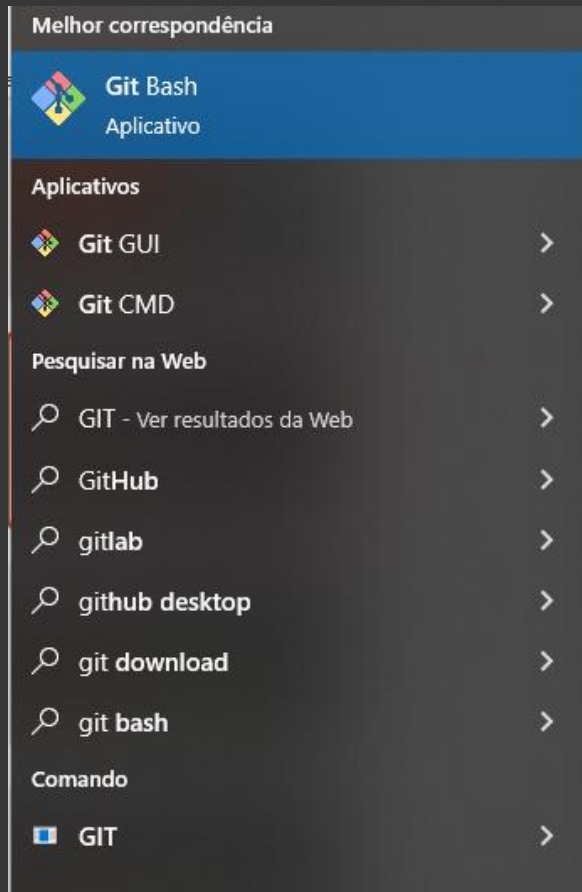
# Fluxo de trabalho

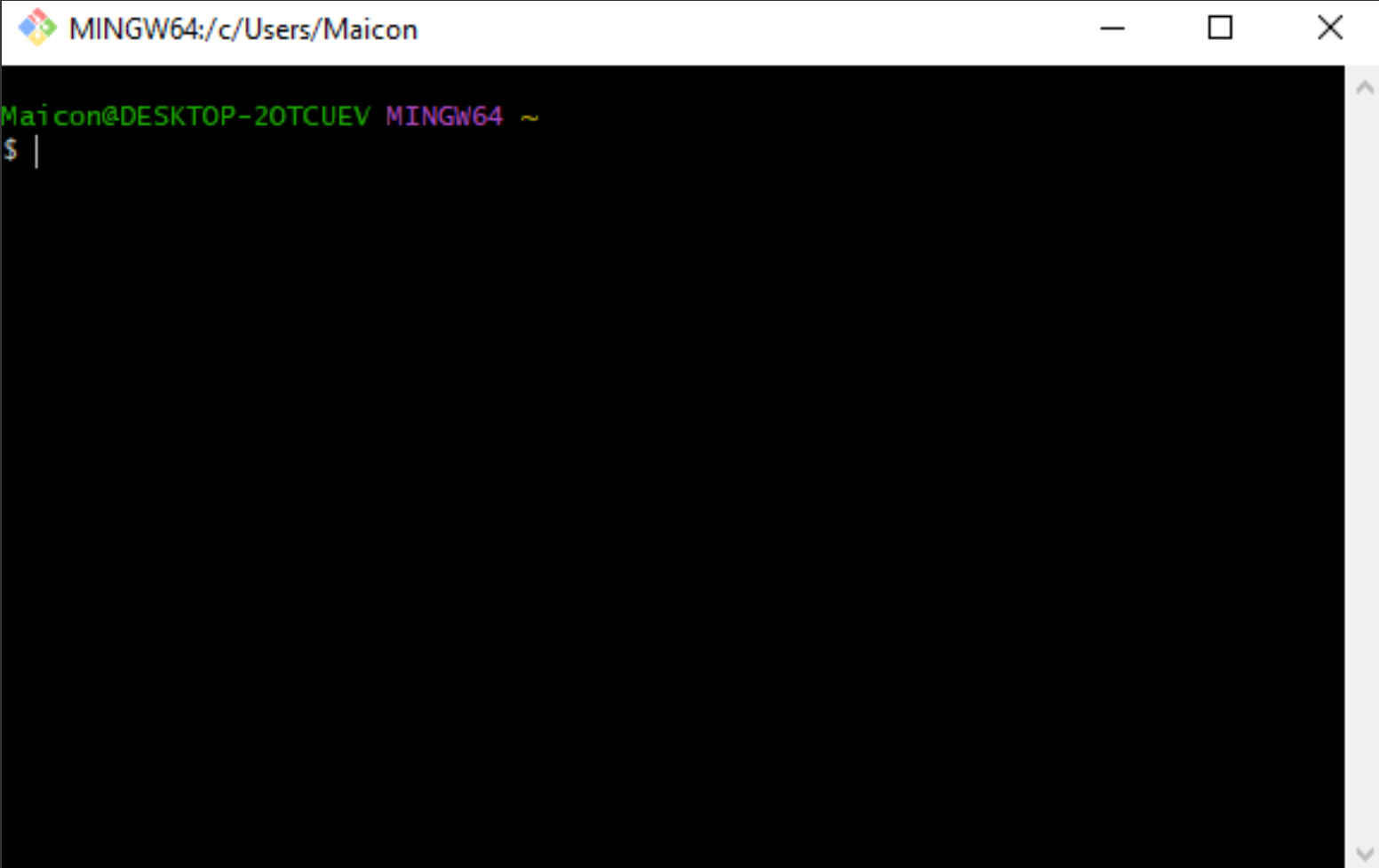
- Você modifica arquivos no seu diretório de trabalho.
- Você adiciona arquivos modificados para uma área de transferência.
- Você salva os arquivos que estão na área de preparo e armazena esses arquivos de forma permanente para o diretório Git.





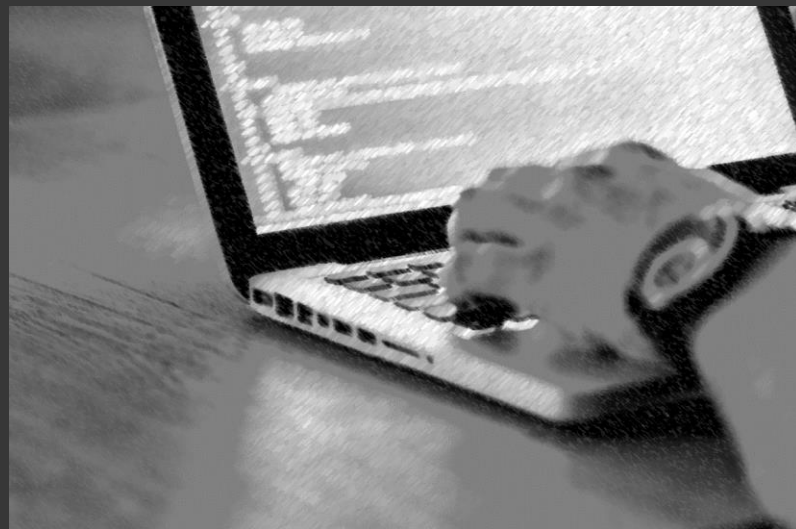
# Inicializando o **GIT BASH**



A screenshot of a MINGW64 terminal window. The title bar at the top shows the MINGW64 logo and the path "MINGW64:/c/Users/Maicon". The terminal area has a black background. The prompt "Maicon@DESKTOP-20TCUEV MINGW64 ~" is displayed in green and purple. Below it, a white dollar sign "\$" and a vertical cursor bar are visible.

```
MINGW64:/c/Users/Maicon

Maicon@DESKTOP-20TCUEV MINGW64 ~
$ |
```



Mãos à obra!

# GIT

Para configurar inicialmente suas credenciais no GIT, é necessário executar os comandos abaixo:

```
git config --global user.name "usuario"  
git config --global user.email "email"
```

Pronto, o GIT já está com suas credenciais.



# GIT

Para configurar inicialmente seu repositório no GIT, é necessário executar o comando abaixo:

**git init**

Pronto, o projeto já é um repositório Git.

Uma pasta chamada .git foi criada no diretório em que o comando foi executado.

# GIT

Para que todos os arquivos sejam versionados, você pode adiciona-los através do seguinte comando:

```
git add .
```

Será adicionado todos arquivos do diretório que ainda não foram salvos em comando add no Git.

# GIT

Caso desejar adicionar somente um arquivo, deverá informar o nome do arquivo. Você pode adicionar através do seguinte comando:

**git add “nomedoarquivo”**

Será adicionado somente o arquivo em que foi informado no comando add no Git.

# GIT

Podemos ver a situação dos arquivos do repositório GIT, acompanhando o status de cada modificação com o comando a seguir:

## **git status**

Será retornado o status dos arquivos que foram adicionados ou não foram ainda adicionados que constam no repositório.

# GIT

Para gravarmos as mudanças no repositório Git, devemos executar o comando:

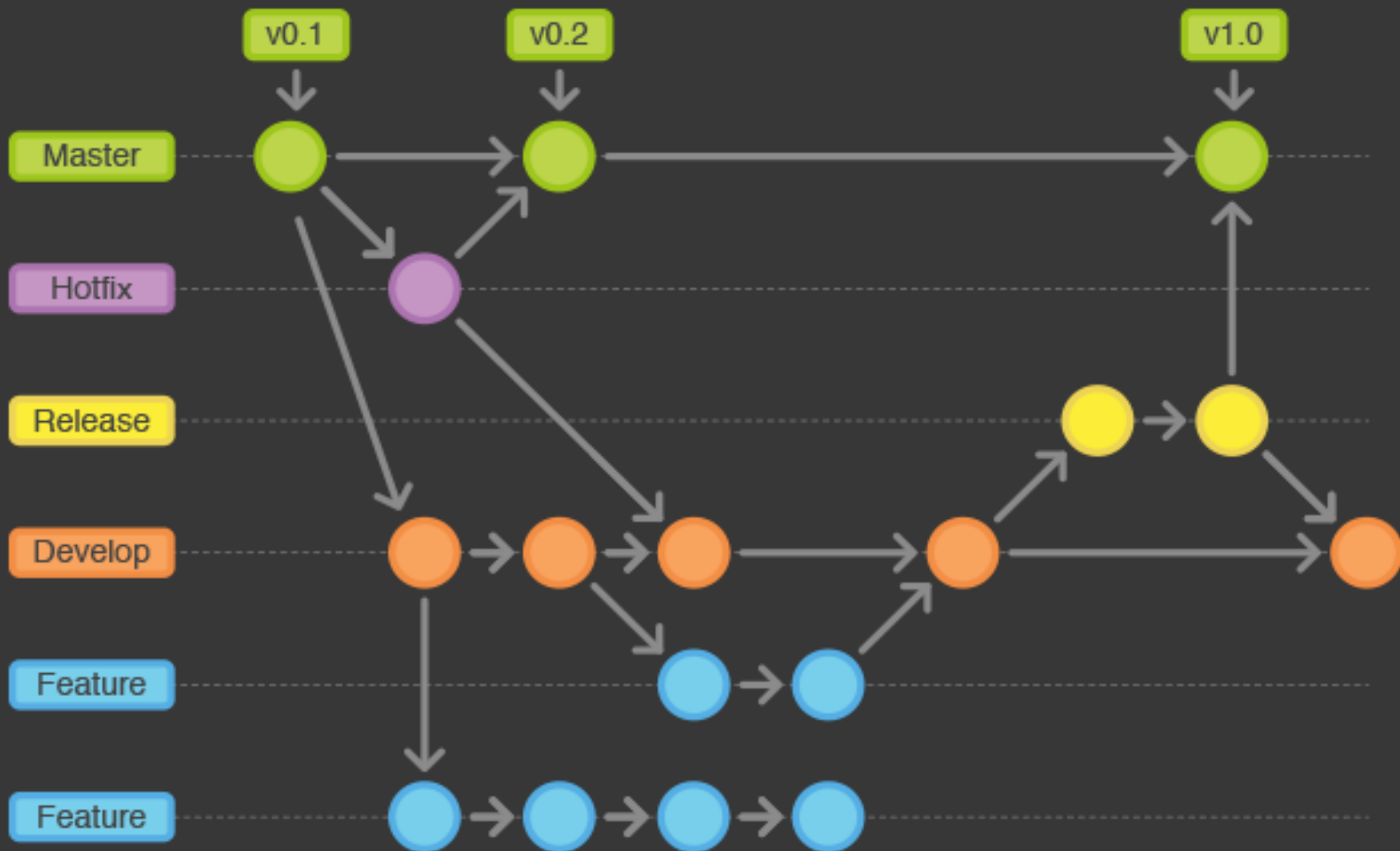
**git commit -m “mensagem”**

Será salvo uma versão do “arquivo.txt” no repositório do Git. Será exibido mensagem que foi adicionado um *commit* identificado e ele carrega a seguinte mensagem de *commit* “Arquivo inicial”.

# GIT

Um recurso muito utilizado do GIT é sua capacidade de permitir que criem vários ramos (branches) independentes dentro de um único projeto.

# Branches



# GIT

O branch padrão em qualquer projeto é sempre o master branch. Um novo ramo (branch) pode ser criado usando o seguinte comando:

**git checkout -b nomebranch**

A *branch* será criada no **REPOSITÓRIO LOCAL**.



# GIT

Outra maneira de se criar uma nova branch é utilizando o comando **Branch**. Um novo ramo (branch) pode ser criado usando o seguinte comando:

```
git branch nomebranch
```

A nova *branch* será criada no **REPOSITÓRIO LOCAL**.

# GIT

Para excluir uma Branch que já existe, é possível utilizar o comando abaixo informando o parâmetro `-d` antes do nome da branch.

```
git branch -d nomebranch
```

A *branch* será deletada do **REPOSITÓRIO LOCAL**.

# GIT

Porém caso você desejar retornar para uma *branch* existente, pode ser utilizado apenas o comando *checkout* da seguinte forma:

```
git checkout master
```

Seu repositório local agora será utilizado com os arquivos do repositório Git master.

# GIT

Agora que temos duas branches, para juntar duas branches, com diferentes *commits*, podemos unir as *branches* precisamos rodar o comando:

## git merge outrabranch

Para que o git merge funcione, precisamos estar na *branch* que irá receber os *commits*. Ao rodar o comando a branch será atualizada

# GIT

Porém, caso você queira obter a cópia de um repositório do Git já existente:

**git clone** url

Pronto, será obtido uma cópia/clone com todos os dados deste repositório remoto no diretório onde você executou o comando.

# GIT

Para tornar o *branch* disponível para outros usuários, você terá que **EMPURRAR** para o repositório remoto. Para fazer isso, use o seguinte comando:

```
git push origin nomebranch
```

A *branch* será enviada para o repositório remoto Git.

# GIT

Caso você queira atualizar seu diretório de trabalho local para uma versão mais recente do repositório remoto, você pode **PUXAR** com o simples comando:

**git pull**

**git fetch**

Será atualizado o repositório local com os dados do repositório remoto do Git.

# GIT

Para visualizar o histórico de commits existentes, basta utilizar o comando:

## git log

Será listado todos os commits que foram realizados no repositório do Git. É um histórico!



# GIT

Para inserir um repositório remoto podemos utilizar o comando abaixo:

```
git remote add origin <url>
```

Será inserido no ambiente remoto o novo repositório conforme URL.

# GIT

Para remover um repositório remoto podemos utilizar o comando abaixo:

```
git remote rm origin
```

O repositório será removido do ambiente remoto.

# GIT

Desfaz as alterações nos arquivos de trabalho. Permite limpar por completo as alterações que não estão no repositório.

**git reset --hard**

Para que o git reset funcione é necessário que exista arquivos na branch adicionados.

# GIT

Desfaz o commit do repositório remoto. Realiza a reversão de um commit de forma fácil e segura.`rever`

**git revert** <commit>

O comando git revert sempre precisa de uma referência de commit para que seja revertido. Caso não seja informado, deverá passar o último commit como referência para seu correto funcionamento.

# GIT

Restaura caminhos especificados do trabalho, mas se não existir o ponto de restauração, ele pode ser removido para à origem.

```
git restore .
```

```
git restore --source <commit>
```

# GIT

Você pode pegar todas as alterações que foram comitadas em uma branch e mesclar em outro branch

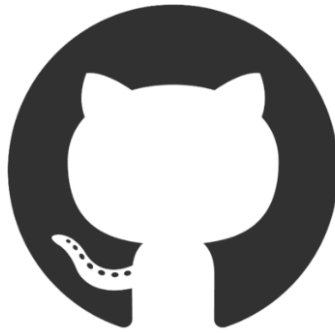
**git rebase <outrabbranch>**

\*\*\* Lembre-se de executar o git rebase somente em um repositório local.

\*\*\* Git rebase oferece muitos riscos, necessário muita atenção.



**Bit Bucket**



**GitHub**



**GitLab**

## 1. GIT (GitHub, GitLab, Bitbucket)

2. Helix Core
3. Apache Subversion
4. IBM Rational ClearCase
5. Team Foundation Server
6. Visual Studio Team Services (VSTS)
7. Mercurial
8. Version Control Systems (CVS)
9. Bazaar
10. Monotone
11. SourceGear Vault
12. Polytron Version Control System (PVCS)
13. Darcs Advanced Revision Control System
14. AccuRev SCM
15. Vault
16. Revision Control System (RCS)
17. GNU arch
18. CA Harvest Software Change Manager
19. Plastic SCM
20. Code Co-op
21. ArX
22. SourceAnywhere Hosted
23. Gerrit
24. Source Code Control System (SCCS)
25. Sourcetree
26. SourceForge
27. Repo





Beanstalk



AWS CodeCommit



git



# GITHUB Desktop



GitHub, uma  
aplicação Web que  
possibilita criar  
de repositórios Git  
e de forma visual.



GitHub é um site no qual você pode fazer criação e atualização de projetos Git.

Crie uma conta no GitHub:

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

# IMPORTANTE!!!

A diferença entre GIT e GITHUB é que o GIT é uma ferramenta para versionar projetos, enquanto o GITHUB é a aplicação (site) no qual você colocará esses projetos Git versionados.



[Overview](#) [Release Notes](#) [Help](#)

# GitHub Desktop

Focus on what matters instead of fighting with Git. Whether you're new to Git or a seasoned user, GitHub Desktop simplifies your development workflow.

**Download for Windows (64bit)**

Download for [macOS](#) or [Windows \(msi\)](#)

By downloading, you agree to the [Open Source Applications Terms](#).

# NA PRÁTICA

---

# ATIVIDADES

---

## 1. Atividades com git:

- ✓ Crie uma conta no GitHub e um novo repositório;
- ✓ Faça o clone do repositório criado para o seu computador;
- ✓ Crie um arquivo README.md e faça o commit para o repositório;
- ✓ Faça o push das mudanças para o repositório no GitHub.



## 2. Continue a partir do exercício 1:

- ✓ Crie uma nova branch no seu repositório;
- ✓ Faça algumas mudanças em um arquivo existente no seu repositório;
- ✓ Faça o commit das mudanças na nova branch;
- ✓ Abra um pull request para mesclar a nova branch com a branch principal do repositório;
- ✓ Peça para um colega de equipe revisar o seu pull request;
- ✓ Se houver comentários ou alterações sugeridas, faça as mudanças necessárias e atualize o pull request;
- ✓ Depois que o pull request for aprovado, faça o merge da nova branch com a branch principal do repositório.

## Aprendemos

- ✓ Instalar o Git,
- ✓ Criar repositórios,
- ✓ Adicionar arquivos,
- ✓ Verificar históricos,
- ✓ Visualizar estados,
- ✓ Fazer commits,
- ✓ Obter e Enviar versões,
- ✓ Criar branches,
- ✓ Navegar entre branches.



git

Obrigado!



**Prof: Me. Maicon dos Santos**