

# **TUTORIAL: COMO USAR O GIT E O GITHUB NO SEU PROJETO**

---

Projeto em desenvolvimento de aplicação WEB -  
Parte 1

O **Git** é um sistema de controle de versão distribuído e um sistema de gerenciamento de código fonte criado por Linus Torvalds, também criador do sistema operacional Linux.

Já o **GitHub** é um repositório remoto e online para o controle de códigos com planos gratuitos, e planos pagos caso o usuário queira deixar seus projetos privados.

Ou seja, Git e GitHub são duas coisas completamente diferentes, mas que trabalham em conjunto.

- Para iniciar o processo de instalação do Git usando o **Git Bash**, acesse o [site](#) do Git para fazer o download do instalador.

## Downloads



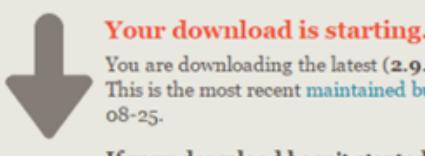
Latest source Release  
**2.9.3**  
Release Notes (2016-08-12)  
Downloads for Windows

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

**GUI Clients**  
Git comes with built-in GUI tools (`git-gui`, `gitk`), but there are several third-party tools for users looking for a platform-specific experience.  
[View GUI Clients →](#)

**Logos**  
Various Git logos in PNG (bitmap) and EPS (vector) formats are available for use in online and print projects.  
[View Logos →](#)

## Downloading Git



Your download is starting...  
You are downloading the latest (2.9.3) 64-bit version of Git for Windows. This is the most recent [maintained build](#). It was released 2 days ago, on 2016-08-25.

If your download hasn't started, [click here to download manually](#).

**Other Git for Windows downloads**

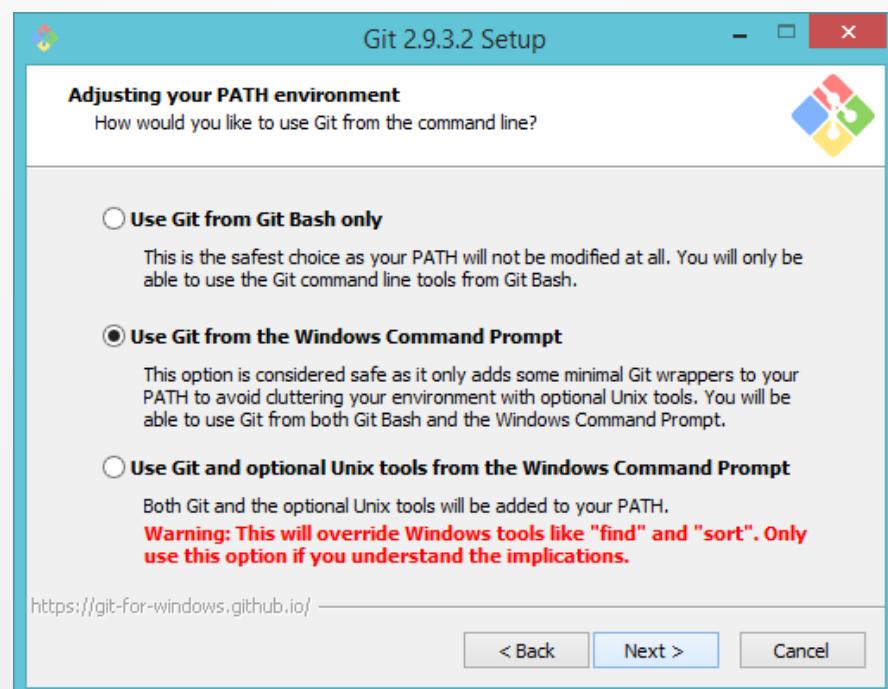
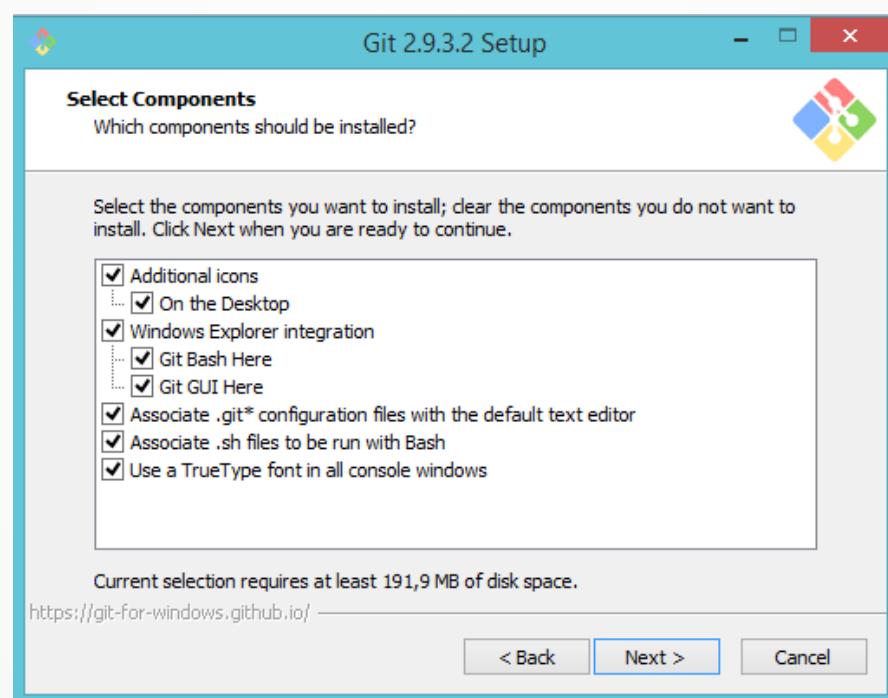
Git for Windows Setup  
[32-bit Git for Windows Setup.](#)

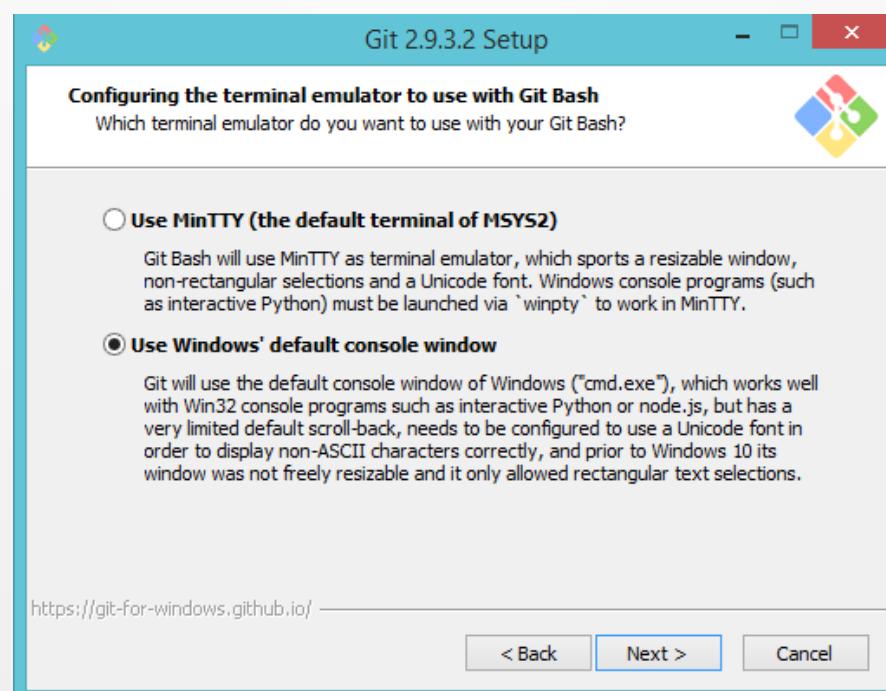
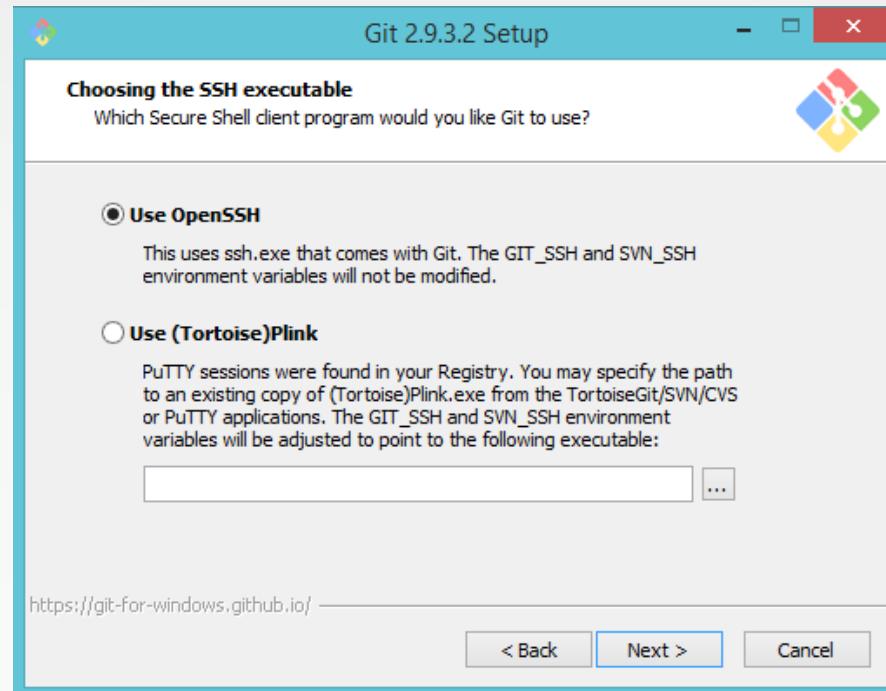
 [64-bit Git for Windows Setup.](#)

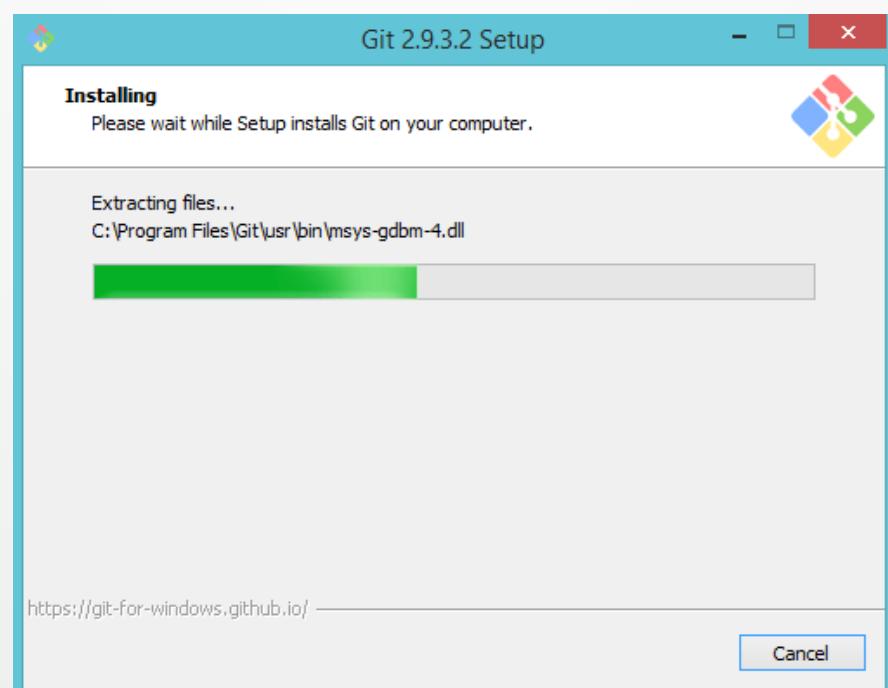
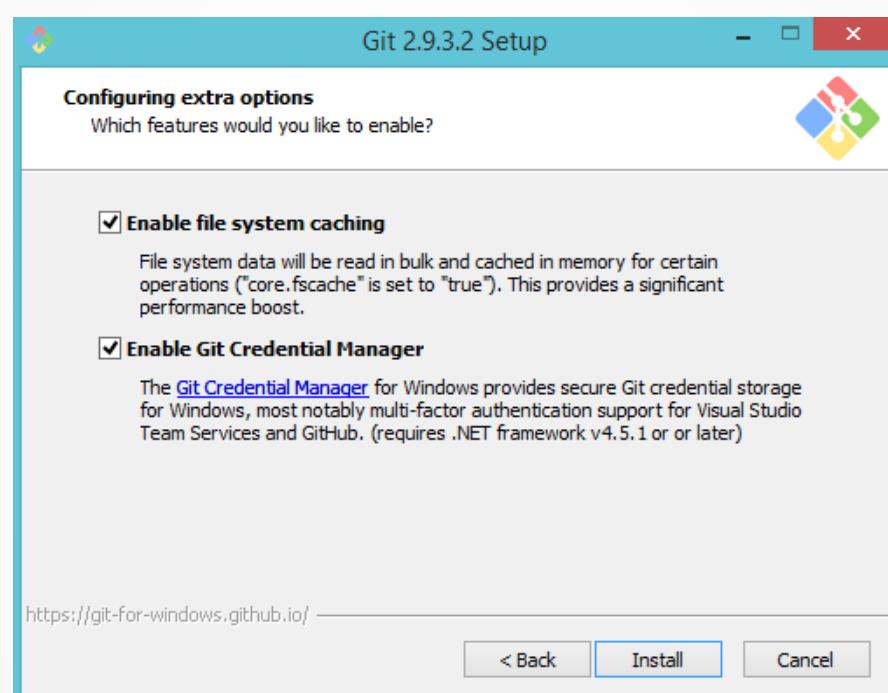
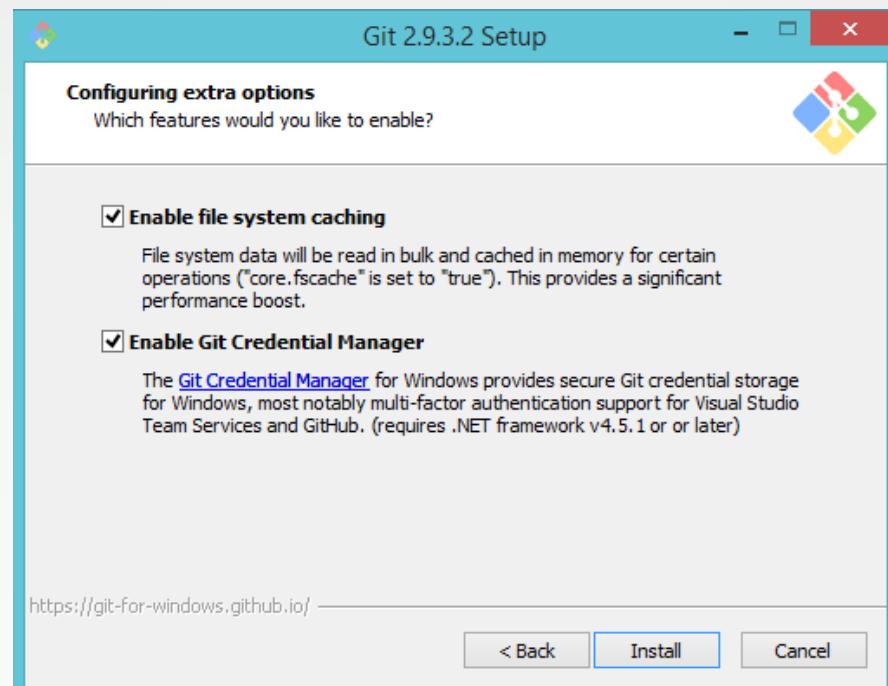
Git for Windows Portable ("thumbdrive edition")  
[32-bit Git for Windows Portable.](#)

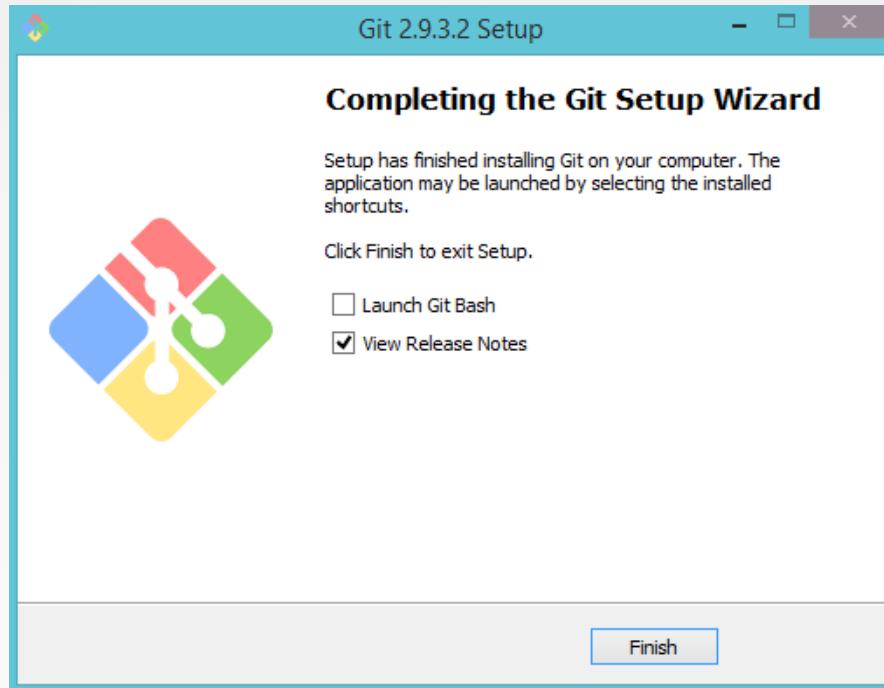
[64-bit Git for Windows Portable.](#)

The current source code release is version 2.9.3. If you want the newer version, you can build it from [the source code](#).









- Vamos usar uma ferramenta chamada Git Bash (que foi instalada junto com o Git) para executar os comandos do Git.
- Para verificar se a instalação do Git foi correta, vamos executar um primeiro comando que é verificar a versão do Git:

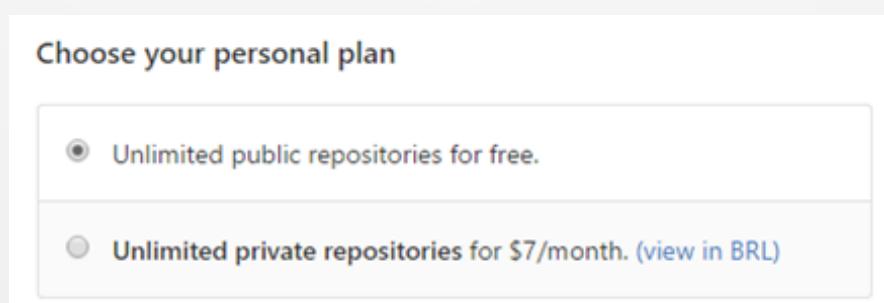
**git --version**

- Vamos iniciar duas configurações básicas no Git, cadastrar o nome do autor e o email do autor.

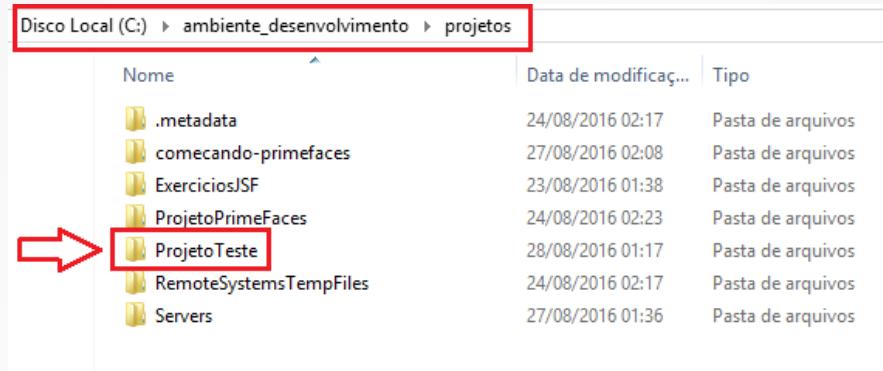
Cadastre seu nome: **git config --global user.name "seu nome aqui"**

Cadastre seu email: **git config --global user.email "seu email aqui"**

- Para se cadastrar no GitHub, vamos acessar [este link](#).
- Clique na opção Sign Up, informe um usuário (username), informe seu email e cadastre uma senha de no mínimo 7 dígitos, misturando letras e números (não coloque caracteres especiais como: @#\$%).
- Escolha a opção de plano Gratuita (Free – Unlimited public repositories for free), verifique no e-mail que você cadastrou e ative sua conta.



- Para criar o seu repositório local, inicie novamente o Git Bash e adicione um dos nossos projetos no repositório **locrtal**.
- Como exemplo, veja o contexto abaixo. Há um projeto chamado “ProjetoTeste”, é um projeto Java utilizando Maven e a IDE é o Eclipse (a IDE acaba sendo irrelevante, já que é um project Maven e pode ser importado em qualquer IDE – netbeans, eclipse, etc.).

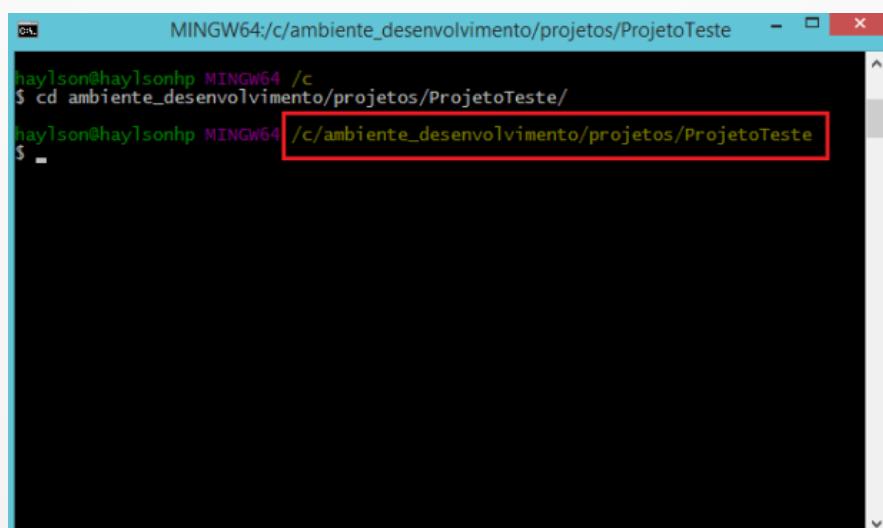


- A partir de agora, vamos fazer o controle de versão desse projeto já existente;
- Mais uma vez, lembrando que estamos utilizando o Windows, e o projeto se encontra no diretório:

**C:ambiente\_desenvolvimento/projetosProjetoTeste**

- Iniciamos o Git Bash e navegamos pela linha de comando até o diretório do nosso projeto “ProjetoTeste”. Assim:

**cd ambiente\_desenvolvimento/projetos/ProjetoTeste**



- Agora que navegamos pelo prompt do Git Bash para dentro do diretório que queremos controlar, executamos o seguinte comando:

**git init**

- Veja: na imagem a seguir, no próprio Git Bash, ao lado do diretório que queremos controlar, apareceu em azul a palavra “master” entre parênteses. Isso significa a branch padrão do git, e que agora essa pasta/diretório está sendo monitorada pelo Git.

```
MINGW64:c/ambiente_desenvolvimento/projetos/ProjetoTeste
$ cd ambiente_desenvolvimento/projetos/ProjetoTeste/
haylson@haylsonhp MINGW64 /c/ambiente_desenvolvimento/projetos/ProjetoTeste
$ git init
Initialized empty Git repository in C:/ambiente_desenvolvimento/projetos/ProjetoTeste/.git/
haylson@haylsonhp MINGW64 /c/ambiente_desenvolvimento/projetos/ProjetoTeste (master)
$ -
```

- Para verificarmos a situação dos arquivos dentro desse diretório **ProjetoTeste**, podemos dar o seguinte comando:

**git status**

```
MINGW64:c/ambiente_desenvolvimento/projetos/ProjetoTeste
$ git init
Initialized empty Git repository in C:/ambiente_desenvolvimento/projetos/ProjetoTeste/.git/
haylson@haylsonhp MINGW64 /c/ambiente_desenvolvimento/projetos/ProjetoTeste (master)
$ git status
On branch master
Initial commit
untracked files:
  (use "git add <file>..." to include in what will be committed)
    .classpath
    .project
    .settings/
    pom.xml
    src/
    target/
nothing added to commit but untracked files present (use "git add" to track)
haylson@haylsonhp MINGW64 /c/ambiente_desenvolvimento/projetos/ProjetoTeste (master)
$ -
```

- Com o comando **git status**, verificamos que aparecem alguns arquivos e até mesmo outras pastas dentro do diretório que estamos monitorando. Na imagem acima, eles aparecem em vermelho. Isso quer dizer que esses arquivos ainda não estão sendo monitorados ou trackeados, e temos que informar ao Git quais arquivos queremos que ele monitore a partir de agora. No nosso caso, vamos monitorar todos, e para fazer isso executamos o seguinte comando:

**git add .**

**git status**

```

MINGW64:/c/ambiente_desenvolvimento/projetos/ProjetoTeste
haylson@haylsonhp MINGW64 /c/ambiente_desenvolvimento/projetos/ProjetoTeste (master)
$ git add .
haylson@haylsonhp MINGW64 /c/ambiente_desenvolvimento/projetos/ProjetoTeste (master)
$ git status
on branch master
Initial commit

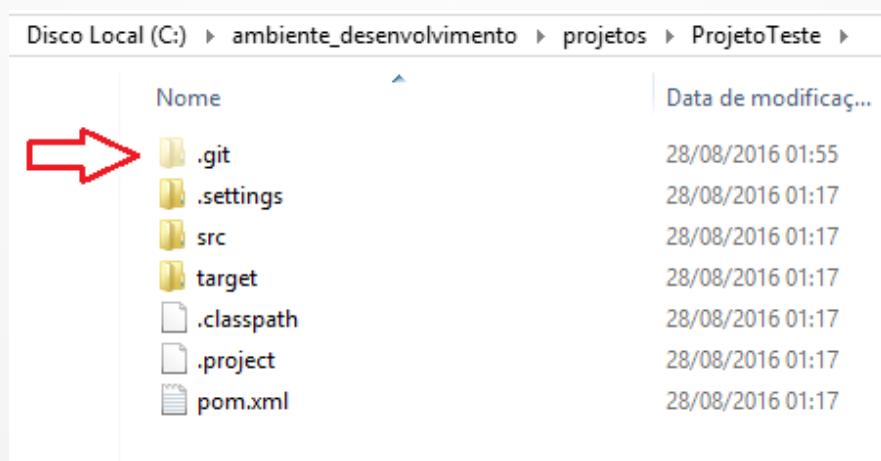
Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

    new file:  .classpath
    new file:  .project
    new file:  .settings/.jsdtscope
    new file:  .settings/org.eclipse.jdt.core.prefs
    new file:  .settings/org.eclipse.m2e.core.prefs
    new file:  .settings/org.eclipse.wst.common.component
    new file:  .settings/org.eclipse.wst.common.project.facet.core.xml
    new file:  .settings/org.eclipse.wst.jsdt.ui.superType.container
    new file:  .settings/org.eclipse.wst.jsdt.ui.superType.name
    new file:  .settings/org.eclipse.wst.validation.prefs
    new file:  pom.xml
    new file:  src/main/webapp/WEB-INF/web.xml
    new file:  target/m2e-wtp/web-resources/META-INF/MANIFEST.MF
    new file:  target/m2e-wtp/web-resources/META-INF/maven/info.hmartins/ProjetoTeste/pom.properties
    new file:  target/m2e-wtp/web-resources/META-INF/maven/info.hmartins/ProjetoTeste/pom.xml

haylson@haylsonhp MINGW64 /c/ambiente_desenvolvimento/projetos/ProjetoTeste (master)
$ 

```

- Perceba que executamos o comando **git add .** (com espaço antes do ponto final, um coringa que quer dizer “adicone tudo”) e, após adicionar todos os arquivos para serem monitorados, não nos foi mostrado nada na tela do Git Bash. Por esse motivo, executamos novamente o comando **git status** para vermos o status atual do nosso diretório monitorado.
- Após ver o status dos nossos arquivos, percebemos que eles estão na cor verde (como mostra a figura acima), ou seja, estão sendo trackeados/monitorados pelo Git a partir de agora, porém eles ainda não estão dentro do nosso repositório Git local, que fica no seu próprio computador, em uma pasta - provavelmente oculta - chamada **.git**, dentro do diretório que você está monitorando, como mostra a figura abaixo:



- Para enviar os arquivos para o repositório local, devemos fazer um commit desses arquivos com o seguinte comando:

**git commit -m “commit inicial”**

- Esse comando faz o envio para o repositório local do Git que está em nossos computadores, o parâmetro (-m) significa que queremos enviar uma mensagem explicando do que se trata esse commit para que possamos identificar do que se trata cada commit, e entre aspas duplas você informa a mensagem. No nosso caso, informamos que esse é o commit inicial do projeto.

```

MINGW64:/c/ambiente_desenvolvimento/projetos/ProjetoTeste - □ ×
haylson@haylsonhp MINGW64 /c/ambiente_desenvolvimento/projetos/ProjetoTeste (master)
$ git commit -m "commit inicial"
[master (root-commit) 6dcba1] commit inicial
 15 files changed, 158 insertions(+)
 create mode 100644 .classpath
 create mode 100644 .project
 create mode 100644 .settings/.jsdtscope
 create mode 100644 .settings/org.eclipse.jdt.core.prefs
 create mode 100644 .settings/org.eclipse.m2e.core.prefs
 create mode 100644 .settings/org.eclipse.wst.common.component
 create mode 100644 .settings/org.eclipse.wst.common.project.facet.core.xml
 create mode 100644 .settings/org.eclipse.wst.jsdt.ui.superType.container
 create mode 100644 .settings/org.eclipse.wst.jsdt.ui.superType.name
 create mode 100644 .settings/org.eclipse.wst.validation.prefs
 create mode 100644 pom.xml
 create mode 100644 src/main/webapp/WEB-INF/web.xml
 create mode 100644 target/m2e-wtp/web-resources/META-INF/MANIFEST.MF
 create mode 100644 target/m2e-wtp/web-resources/META-INF/maven/info.hmartins/ProjetoTeste/pom.properties
 create mode 100644 target/m2e-wtp/web-resources/META-INF/maven/info.hmartins/ProjetoTeste/pom.xml
haylson@haylsonhp MINGW64 /c/ambiente_desenvolvimento/projetos/ProjetoTeste (master)
$
```

- Verifique a mensagem do commit e veja que, no exemplo acima, ele “commitou” 15 arquivos, e os nomes dos arquivos não estão mais de vermelho ou verde, mas na cor branca.
- Execute novamente o comando **git status**:

```

MINGW64:/c/ambiente_desenvolvimento/projetos/ProjetoTeste - □ ×
create mode 100644 .settings/org.eclipse.wst.jsdt.ui.superType.name
create mode 100644 .settings/org.eclipse.wst.validation.prefs
create mode 100644 pom.xml
create mode 100644 src/main/webapp/WEB-INF/web.xml
create mode 100644 target/m2e-wtp/web-resources/META-INF/MANIFEST.MF
create mode 100644 target/m2e-wtp/web-resources/META-INF/maven/info.hmartins/ProjetoTeste/pom.properties
create mode 100644 target/m2e-wtp/web-resources/META-INF/maven/info.hmartins/ProjetoTeste/pom.xml

haylson@haylsonhp MINGW64 /c/ambiente_desenvolvimento/projetos/ProjetoTeste (master)
$ git status
On branch master
nothing to commit, working tree clean

haylson@haylsonhp MINGW64 /c/ambiente_desenvolvimento/projetos/ProjetoTeste (master)
$
```

- A mensagem de status que nos é informado após o commit é: **On branch master**: Estamos na branch master; **Nothing to commit**: Não há mais nada a ser commitado no momento; **Working tree clean**: Nosso diretório está limpo/vazio, ou sem mudanças até o momento.
- Caso você queira adicionar os arquivos e commitar em um único comando, poderíamos commitar diretamente acrescentando o parametro -a (de adicionar), ficando o comando da seguinte maneira:

**git commit -a -m “commitando e adicionando juntos”**

- Também podemos determinar que o Git não monitore determinados arquivos ou pastas dentro do diretório que ele esteja monitorando. Um exemplo bem comum são as pastas libs, resources ou de outras bibliotecas que são controladas, sem a necessidade que o Git monitore-as.
- Apenas para um exemplo didático, foi criado dentro do projeto um arquivo chamado arquivoignorado.txt e uma pasta chamada pastaignorada. Dentro dela temos outro arquivo, chamado ignorado.txt, conforme a imagem a seguir:

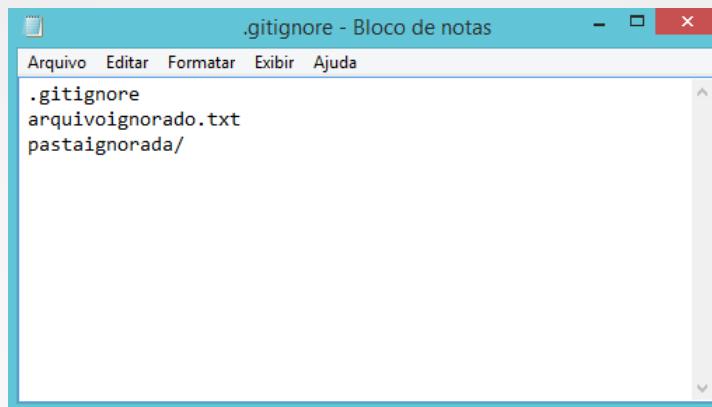
Disco Local (C:) > ambiente_desenvolvimento > projetos > ProjetoTeste >	
Nome	Data de modificaç...
.git	28/08/2016 02:48
.settings	28/08/2016 01:17
<b>pastaignorada</b>	28/08/2016 02:55
src	28/08/2016 01:17
target	28/08/2016 01:17
.classpath	28/08/2016 01:17
.project	28/08/2016 01:17
<b>arquivoignorado.txt</b>	28/08/2016 02:45
pom.xml	28/08/2016 01:17

- Criados esses dois arquivos, não será possível executá-los com o comando git status. Verifique que esses arquivos não estão sendo ignorados pelo monitoramento do Git conforme a imagem a seguir.

```
MINGW64:/c/ambiente_desenvolvimento/projetos/ProjetoTeste - □ ×
haylson@haylsonhp MINGW64 /c/ambiente_desenvolvimento/projetos/ProjetoTeste (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    → arquivoignorado.txt
    → pastaignorada/
nothing added to commit but untracked files present (use "git add" to track)
haylson@haylsonhp MINGW64 /c/ambiente_desenvolvimento/projetos/ProjetoTeste (master)
$
```

- Para informar ao Git que ignore e não monitore esses arquivos ou pastas, devemos criar dentro do diretório que está sendo monitorado. No exemplo, o diretório ProjetoTeste, um arquivo chamado “.gitignore” (com o ponto na frente), e dentro desse arquivo escrever os nomes dos arquivos a serem ignorados, incluindo o próprio arquivo .gitignore, conforme você pode ver na imagem a seguir:



- Nesse momento, ao verificar novamente o status do Git no diretório, pelo comando git status, verificamos que esses arquivos não estão mais sendo observados pelo Git no projeto, conforme imagem a seguir:

```
haylson@haylsonhp MINGW64 /c/ambiente_desenvolvimento/projetos/ProjetoTeste (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    arquivoignorado.txt
    pastaignorada/

nothing added to commit but untracked files present (use "git add" to track)

haylson@haylsonhp MINGW64 /c/ambiente_desenvolvimento/projetos/ProjetoTeste (master)
$ git status
On branch master
nothing to commit, working tree clean

haylson@haylsonhp MINGW64 /c/ambiente_desenvolvimento/projetos/ProjetoTeste (master)
$
```

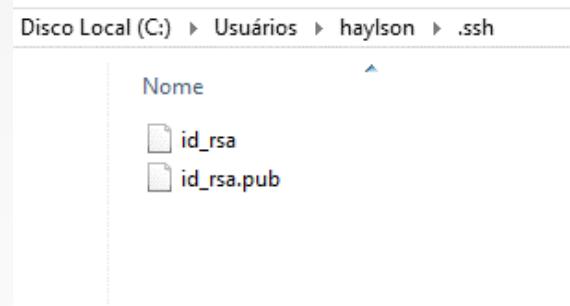
- Sua conta no Github já foi criada, mas antes, ainda no Git, vamos gerar uma chave de segurança para que você tenha acesso do seu computador local ao GitHub. Executamos o seguinte comando e, em seguida, damos enter sem preencher nada até finalizar:

### ssh-keygen

```
haylson@haylsonhp MINGW64 /c/ambiente_desenvolvimento/projetos/ProjetoTeste (master)
$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/haylson/.ssh/id_rsa):
/c/Users/haylson/.ssh/id_rsa already exists.
Overwrite (y/n)? y
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /c/Users/haylson/.ssh/id_rsa.
Your public key has been saved in /c/Users/haylson/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:V0QYPqSJZ/bTASL+3WSQA8bK5EtpJ0oV/v1z7Qa9Few haylson@haylsonhp
The key's randomart image is:
+---[RSA 2048]---+
| ..+==o
| o.o+.*++
| o B o^ o.=.
| . o B= o B .o
| . . + oS ==o.. .
| . . + .o.o E
| . o o o
| o +
+---[SHA256]---+

haylson@haylsonhp MINGW64 /c/ambiente_desenvolvimento/projetos/ProjetoTeste (master)
$
```

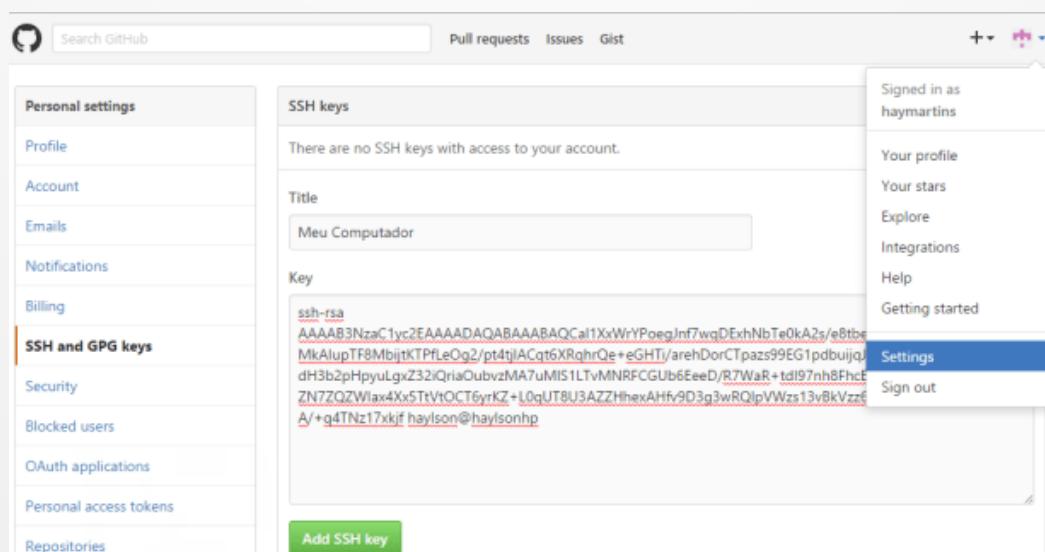
- Como você pode verificar na imagem acima, foram criadas duas chaves dentro da pasta de usuário do Windows e dentro de outro diretório chamado .ssh, que contém as chaves. Entre nesse diretório e verifique os arquivos que estão lá (**C:/Users/haylson/.ssh/**), conforme imagem abaixo:



- Verifique se a sua pasta .ssh não está oculta, e se contém os dois arquivos acima (id\_rsa e id\_rsa.pub). O arquivo importante é o **id\_rsa.pub**. Abra ele com um bloco de notas, e copie todo o código gerado nele, que será algo como na imagem a seguir:



- Agora acesse seu GitHub, procure a opções Settings -> SSH and GPG Keys, clique em New SSH key, em Title para inserir um título qualquer com o nome do seu computador, e na opção Key cole todo o conteúdo copiado no arquivo id\_rsa.pub. Em seguida, clique no botão verde “Add SSH Key”, conforme imagem a seguir:



This is a list of SSH keys associated with your account. Remove any keys that you do not recognize.

	Meu Computador	
	Fingerprint: ff:2b:ec:1e:12:94:89:78:e2:fd:fd:55:6a:14:c5:1d	
SSH	Added on 28 Aug 2016 — Never used	

Check out our guide to [generating SSH keys](#) or troubleshoot [common SSH Problems](#).

- Para criar um Repositório dentro do GitHub, vá na opção New repository, chame, por exemplo, de “meugithub”, coloque uma descrição qualquer, descrevendo do que se trata o repositório, marque a opção Public para torná-lo público, já que esta é a opção gratuita (para colocar a opção Private/Privado você tem que pagar um plano mensal no Github), e clique em Create repository, como mostra a imagem a seguir:

Create a new repository

A repository contains all the files for your project, including the revision history.

Owner haymartins

Great repository names are short and memorable. Need inspiration? How about [ideal-computing-machine](#).

Description (optional)  
Teste com GitHub

Public  
Anyone can see this repository. You choose who can commit.

Private  
You choose who can see and commit to this repository.

Initialize this repository with a README  
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: None Add a license: None

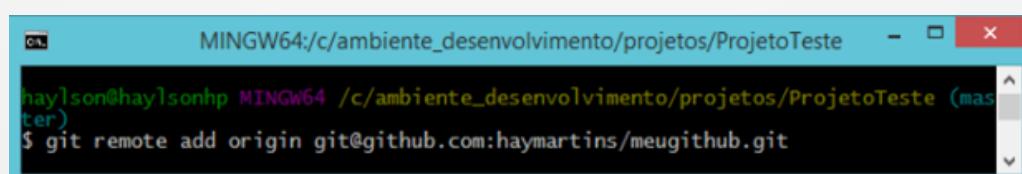
- Na tela seguinte, aparecem algumas opções como: criar um repositório do zero, criar um repositório existente ou importar código de outro repositório, como mostra a imagem a seguir:

The screenshot shows a GitHub repository page for 'haymartins / meugithub'. At the top, there are navigation links for 'Code', 'Issues 0', 'Pull requests 0', 'Wiki', 'Pulse', and 'Graphs'. Below these, a section titled 'Quick setup — if you've done this kind of thing before' provides instructions for cloning the repository. It offers three options: 'Set up in Desktop' (disabled), 'HTTPS' (disabled), and 'SSH' (selected and highlighted with a red box). A note below says 'We recommend every repository include a README, LICENSE, and .gitignore.' Another section, '...or create a new repository on the command line', contains a command-line script for initializing a new repository. A third section, '...or push an existing repository from the command line', contains the command 'git remote add origin git@github.com:haymartins/meugithub.git' and 'git push -u origin master', which is also highlighted with a red box. A final section, '...or import code from another repository', includes a 'Import code' button.

- Para criar um repositório de um projeto já existente em seu computador, o ProjetoTeste, vá na segunda opção “or push an existing repository from the command line”, coloque na opção SSH e copie o código gerado que vede ser utilizado no Git Bash para integrar o repositório local com o repositório online criado no Github.
- Abra novamente o Git Bash, navegue até a pasta do projeto ProjetoTeste (que foi criado no nosso repositório local) e execute o código que foi gerado dentro do GitHub, apontando para o repositório online que foi criado recentemente. No exemplo, o comando ficou assim (no seu caso, deve mudar o endereço do seu repositório criado):

```
git remote add origin github.com/haymartins/meugithub.git
```

```
git push -u origin master
```



```
MINGW64:/c/ambiente_desenvolvimento/projetos/ProjetoTeste
haylson@haylsonhp MINGW64 /c/ambiente_desenvolvimento/projetos/ProjetoTeste (master)
$ git remote add origin git@github.com:haymartins/meugithub.git
haylson@haylsonhp MINGW64 /c/ambiente_desenvolvimento/projetos/ProjetoTeste (master)
$ git push -u origin master
The authenticity of host 'github.com (192.30.253.113)' can't be established.
RSA key fingerprint is SHA256:nThbg6kXUpJWG17E1IGOCspRomTxdCARLviKw6E5SY8.
Are you sure you want to continue connecting (yes/no)? yes
```

```
MINGW64:/c/ambiente_desenvolvimento/projetos/ProjetoTeste
haylson@haylsonhp MINGW64 /c/ambiente_desenvolvimento/projetos/ProjetoTeste (master)
$ git remote add origin git@github.com:haymartins/meugithub.git
haylson@haylsonhp MINGW64 /c/ambiente_desenvolvimento/projetos/ProjetoTeste (master)
$ git push -u origin master
The authenticity of host 'github.com (192.30.253.113)' can't be established.
RSA key fingerprint is SHA256:nThbg6kXUpJWG17E1IGOCspRomTxdCARLviKw6E5SY8.
Are you sure you want to continue connecting (yes/no)? yes
warning: Permanently added 'github.com,192.30.253.113' (RSA) to the list of known hosts.
Counting objects: 28, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (16/16), done.
Writing objects: 100% (28/28), 3.66 KiB | 0 bytes/s, done.
Total 28 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), done.
To github.com:haymartins/meugithub.git
 * [new branch]      master -> master
Branch master set up to track remote branch master from origin.

haylson@haylsonhp MINGW64 /c/ambiente_desenvolvimento/projetos/ProjetoTeste (master)
$ git push -u origin master
warning: Permanently added the RSA host key for IP address '192.30.253.112' to the list of known hosts.
Branch master set up to track remote branch master from origin.
Everything up-to-date

haylson@haylsonhp MINGW64 /c/ambiente_desenvolvimento/projetos/ProjetoTeste (master)
$
```

- E finalmente o projeto está sincronizado com o repositório remoto no GitHub.

haymartins / meugithub

Code Issues Pull requests Wiki Pulse Graphs Settings

1 commit 1 branch 0 releases 1 contributor

Branch: master New pull request

Create new file Upload files Find file Clone or download

File	Commit	Time
.settings	commit initial	3 hours ago
src/main/webapp/WEB-INF	commit initial	3 hours ago
target/m2e-wtp/web-resources/META-INF	commit initial	3 hours ago
.classpath	commit initial	3 hours ago
.project	commit initial	3 hours ago
pom.xml	commit initial	3 hours ago

Help people interested in this repository understand your project by adding a README.

Add a README

- Na configuração inicial do Git Flow, aplica-se apenas a primeira vez:

**git flow init**

Which branch should be used for bringing forth production releases? [Qual branch deve ser usada para trazer liberações de produção?] – development – master.

Branch name for production releases: [master] => Enter [Nome da branch para versões de produção].

Which branch should be used for integration of the “next release”? [Qual branch deve ser usada para integração do “próximo release”?] – development Branch name for “next release” development: [] => “digitar” development [Nome da branch para a “proxima versao” de desenvolvimento].

How to name your supporting branch prefixes? Feature branches? [feature/]Bugfix branches? [bugfix/]Release branches? [release/]Hotfix branches? [hotfix/]Support branches? [support/]Version tag prefix? [] v- (padrão v-) Hooks and filters directory? [D:/ambiente\_desenvolvimento/workspace/nome-projeto/.git/hooks].

- Criando novas branchs:

**git flow feature start <nome-branch>**

git flow feature start [nome-da-branch-a-ser-criada]git flow bugfix start [nome-da-branch-a-ser-criada]git flow hotfix start [nome-da-branch-a-ser-criada]git flow release start [nome-da-branch-a-ser-criada].

[Termos]feature: uma funcionalidade nova. bugfix: correção de bug que ainda não foi para produção. hotfix: correção de bug em produção. release: liberação de uma nova release.

- Finalizando uma branch:

**git flow feature finish <nome-branch>**

git flow feature finish [nome-da-branch-a-ser-criada]git flow bugfix finish [nome-da-branch-a-ser-criada]git flow hotfix finish [nome-da-branch-a-ser-criada]git flow release finish [nome-da-branch-a-ser-criada]

- Se clicar em merge, corrigir, commitar e dar um push, depois dê um pull para atualizar a development.

- Remover Git Flow:

**git config -remove-section "gitflow.path"**

**git config -remove-section "gitflow.prefix"**

**git config -remove-section "gitflow.branch"**

- Para começar a rastrear arquivos, ou colocar no estado stage:

**git add .**

**git add nome-arquivo**

- Para reverter, colocar um arquivo que está stage, para o estado unstage:

**git reset HEAD <nome-arquivo>**

- Descartar todas as alterações feitas em todos os arquivos alterados:

**git checkout -- . (hífen, hífen, espaço depois ponto final)**

- Descartar alterações de apenas um arquivo específico:

**git checkout -- <caminho/nome-do-arquivo>**

- Criar uma branch local:

**git checkout -b nome-branch-local**

- Listar todas as branches locais:

**git branch**

- Listar branches remotas:

**git branch -r**

- Listar todas as branches remotas e locais:

**git branch -a**

- Empurrando/Enviando seu branch local para a branch remota (devem ter o mesmo nome):

**git push origin nome-da-branch-remota**

- Interface gráfica padrão do Git:

**gitk**

- Criar uma branch sendo cópia de outra branch:

**git checkout -b [nome-da-nova-branch-a-ser-criada] [nome-da-branch-que-deseja-copiar]**

- Criando uma branch remota (e efetuando o push da branch local ao mesmo tempo). Obs.: dê preferência para o mesmo nome da [branch-local]:

**git push -u origin [nome-branch-remota]**

- Criando uma branch remota (e efetuando o push da branch local). Obs.: com o mesmo nome da [branch-local]

**git push --set-upstream origin [nome-branch-remota-mesmo-nome-da-local]**

- Realizando merge entre duas branches:

**git checkout <branch-principal>** (ex.: dese, homol ou master por exemplo) **git pull** (faz o pull da branch-principal para atualizar) **git merge <branch\_feature>**. É a branch que voce criou/alterou algo, caso dê conflitos aqui, siga os passos abaixo:

-alterar manualmente os arquivos, resolvendo os conflitos. **git add . git commit -m "mensagem exemplo: merge de branch a com branch b"**

**git checkout <branch-principal>** (ex.: dese, homol ou master por exemplo) **git pull** (faz o pull da branch-principal para atualizar) **git pull origin <nome-da-branch-a-ser-adicionada-na-principal>** **git push** (caso dê conflito, corrigir manualmente)

- Excluindo branches:

**git branch -D <nome-da-branch>** (excluindo branch local) **git branch -dr origin/nome-da-branch-r**

**PUCRS** online       **uol**edtech