

Tutorial

Controle de Versão, Git e GitHub – Parte I

O que é Controle de Versão?

Ao iniciar um novo projeto de software, é importante definir onde os arquivos desse projeto ficarão armazenados. Isso se torna ainda mais necessário quando se trata de um trabalho envolvendo uma equipe com vários desenvolvedores. Nesse caso, o projeto deve ser armazenado em um local acessível a todos os membros da equipe, de modo que novos arquivos possam ser incluídos, e arquivos já existentes possam ser alterados ou excluídos por esses membros.

A princípio, esse local poderia ser um servidor conectado à rede, ao qual os membros da equipe teriam acesso. Esse servidor teria redundância de armazenamento e *backups* programados, para recuperação de arquivos excluídos acidentalmente. Embora pareça suficiente, essa estrutura de armazenamento é bastante limitada, pois não há como desfazer alterações no projeto de forma seletiva. Também não existem informações suficientes para, por exemplo, saber em qual *backup* está a versão de um determinado arquivo que necessita ser recuperado.

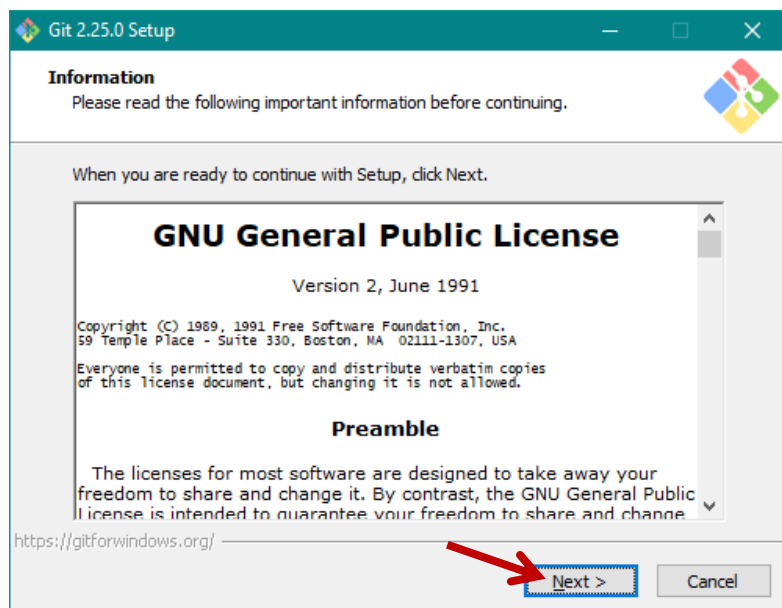
Para solucionar esses problemas foram criadas **ferramentas de controle de versão (ou versionamento)**, softwares que têm a finalidade de gerenciar diferentes versões durante o desenvolvimento de qualquer tipo de documento. Essas ferramentas são comumente utilizadas no desenvolvimento de software para controlar as diferentes versões dos códigos-fontes e da documentação do projeto.

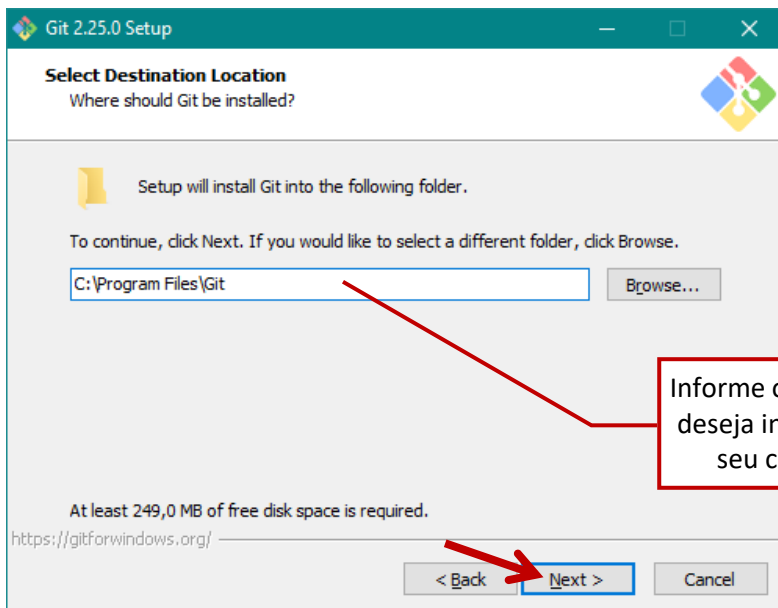
Ferramentas de controle de versão são muito presentes em empresas e instituições de tecnologia e desenvolvimento de software (livre ou comercial). Entre as mais comuns encontram-se as soluções livres: Git, CVS, SVN (Subversion) e Mercurial, e as comerciais: SourceSafe, TFS, PVCS (Serena) e ClearCase.

Uma das ferramentas de controle de versões mais usadas no desenvolvimento de software é o **Git**, um software livre usado em muitos projetos de código aberto em todo o mundo, que também pode ser utilizado para projetos comerciais. Ele possui diversas ferramentas de hospedagem de repositórios na Web, sendo um dos mais conhecidos o **GitHub**, que permite criar gratuitamente repositórios abertos.

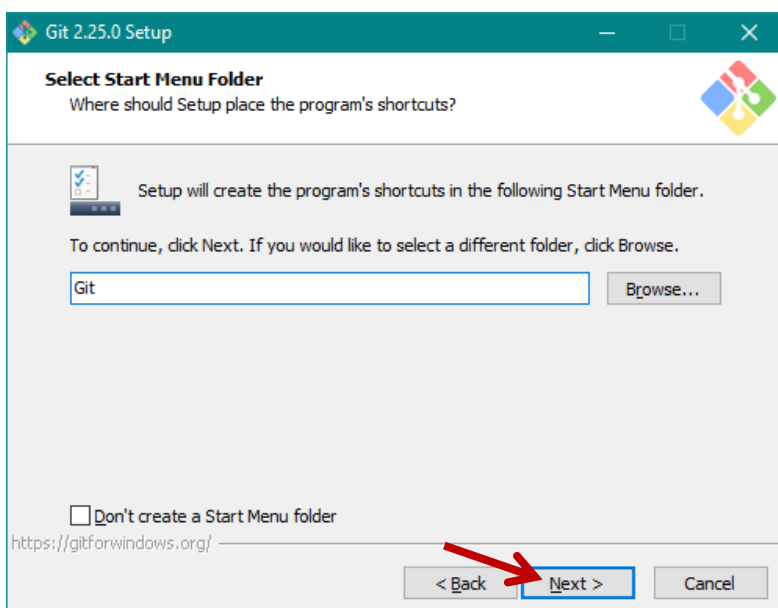
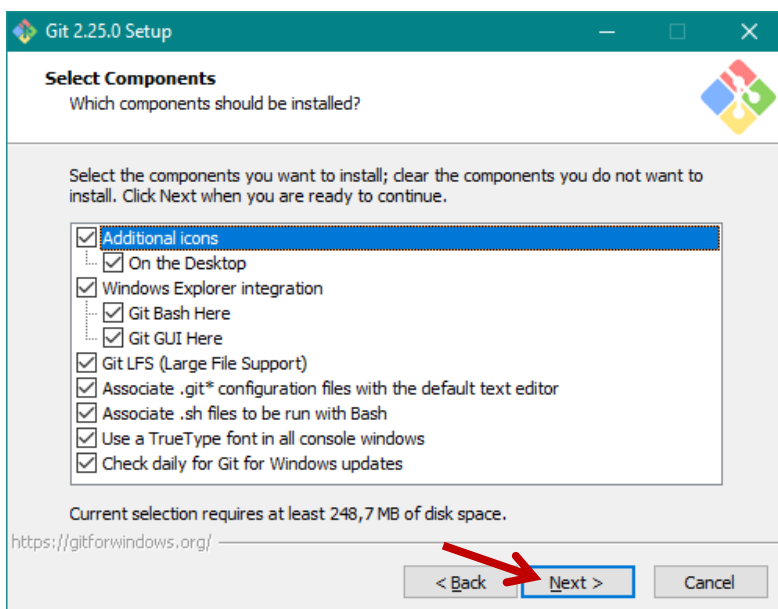
Download e Instalação do Git (Windows)

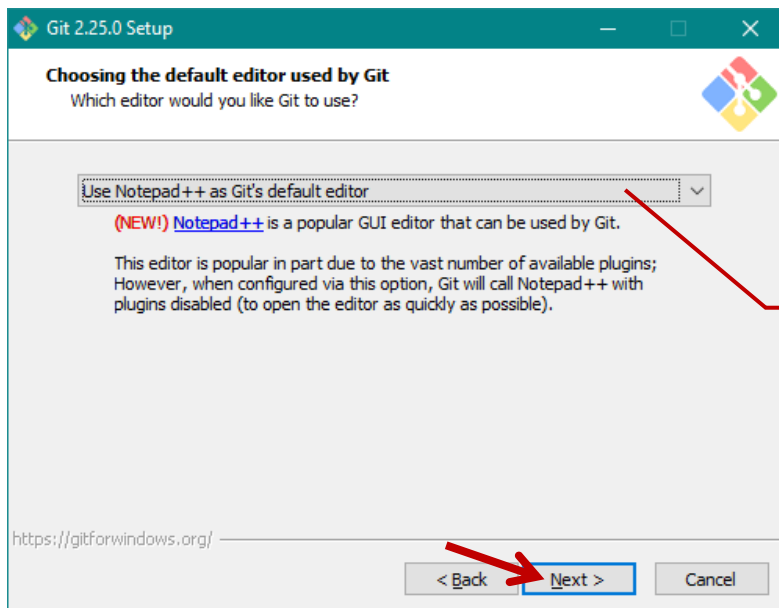
1. Acesse a página do Git: <https://gitforwindows.org/> e baixe o instalador. No momento da escrita deste tutorial, a versão mais recente do instalador é a **2.25.0**.
2. Execute o instalador e siga as instruções a seguir:



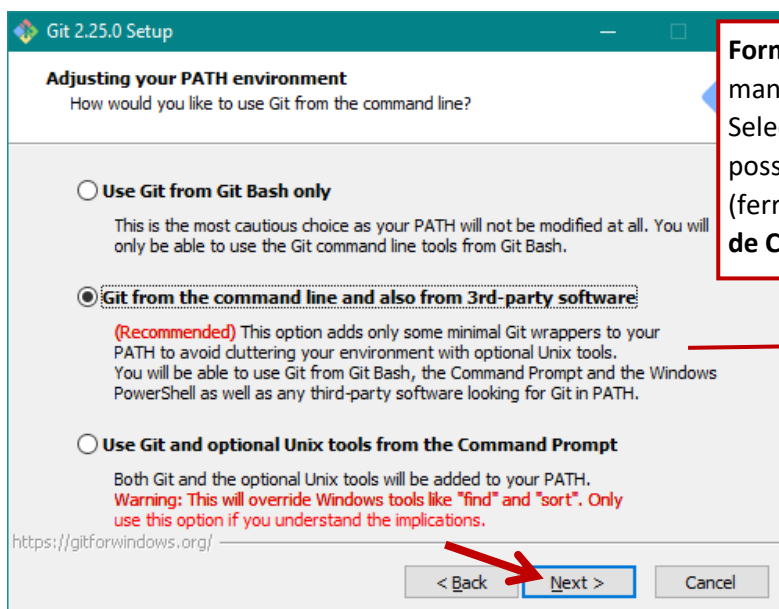


Informe o diretório onde deseja instalar o **Git** em seu computador.

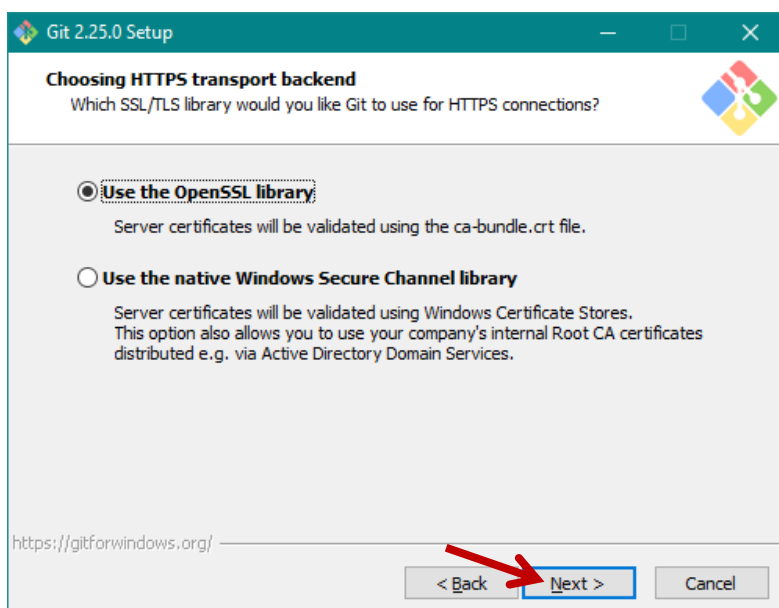


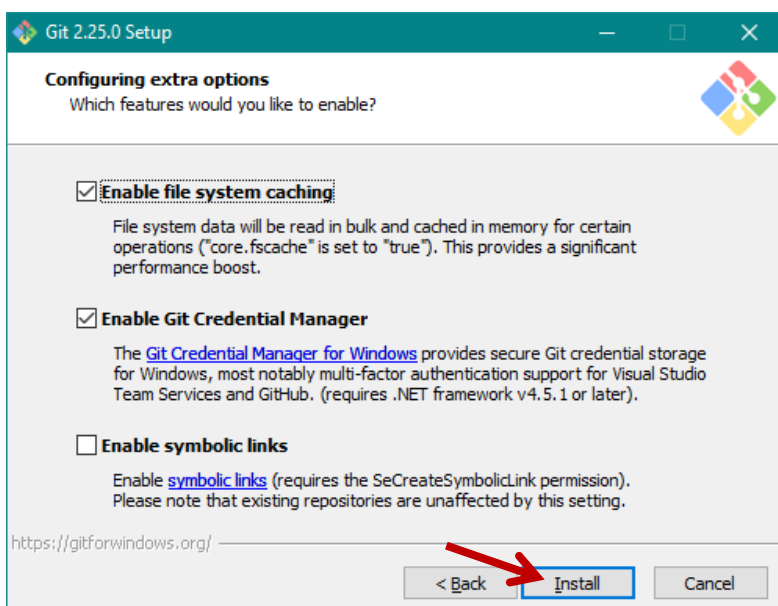
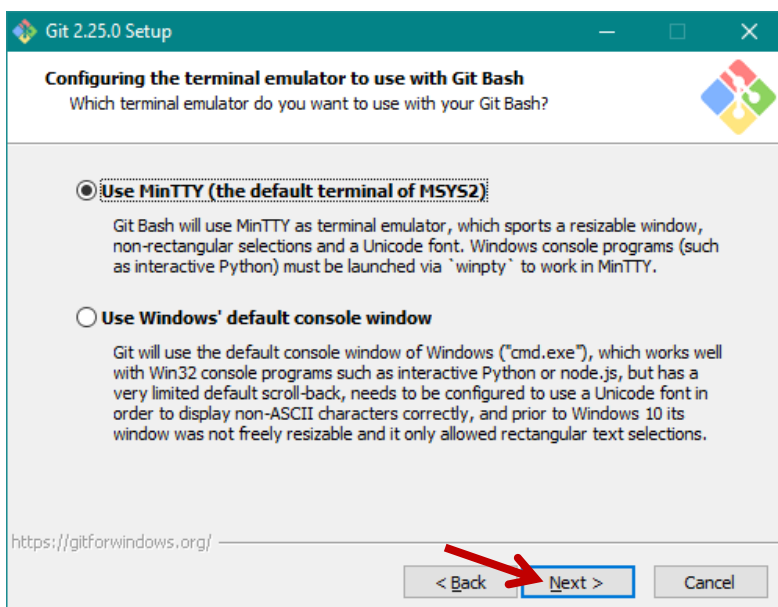
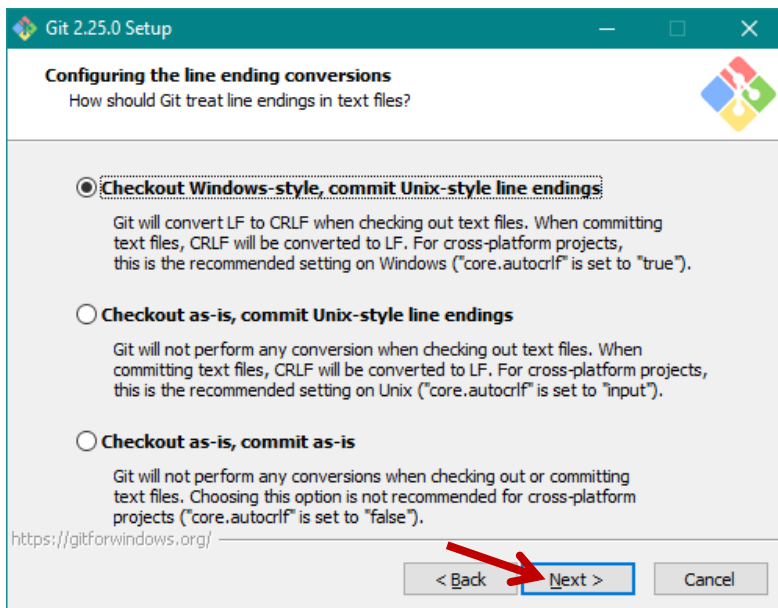


Escolha o editor padrão para usar com o Git.



Formas de usar o Git: Existem algumas formas para manipular os repositórios de arquivos do Git. Selecione a opção recomendada para que seja possível usar o Git por meio do Git Bash (ferramenta instalada junto com o Git), o Prompt de Comando do Windows ou pelo Windows





Criação da Conta no GitHub

Agora com o Git instalado, é possível utilizar os serviços do **GitHub**. Porém, antes de mais nada, é preciso criar uma conta:

1. Acesse a página do **GitHub**: <https://github.com/pricing>
2. Escolha o tipo de conta **Free** e prossiga preenchendo os dados solicitados (e-mail, usuário, senha etc). Após finalizar o cadastro, será solicitado para verificar seu e-mail.
3. Entre em sua conta de e-mail. Abra o e-mail recebido do GitHub e clique no link de verificação contido no corpo do e-mail. Ao clicar no link, você será redirecionado para a página do **GitHub**, onde será solicitada a criação de um novo repositório. Por enquanto, não crie este repositório.

Geração da Chave de Segurança

A chave de segurança será responsável por identificar nossa máquina durante a comunicação entre nosso repositório local e o repositório do GitHub, de modo que possamos utilizar o serviço com segurança.

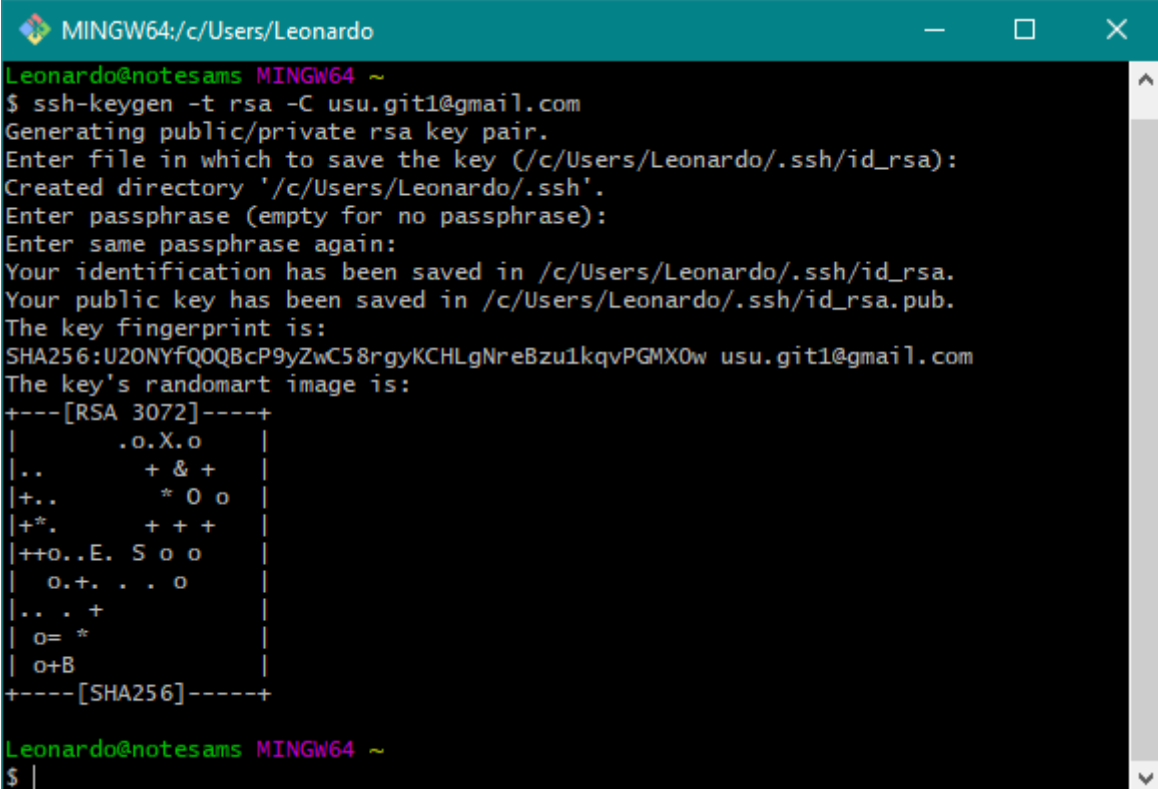
1. Abra o terminal do **Git Bash** e execute o seguinte comando:

`ssh-keygen -t rsa -C seu_email@provedor.com` (e-mail informado na criação da conta no GitHub)

O terminal vai perguntar em que local você quer salvar sua chave de segurança. Para evitar problemas, mantenha a opção padrão (C:\Users\<usuário>\.ssh).

Em seguida, será solicitada a entrada de uma senha (*passphrase*) para a chave de segurança. Caso o computador seja público ou compartilhado, é recomendado que sua chave esteja protegida por uma senha. Caso contrário, ela pode ser ignorada, pressionando **Enter** para a solicitação e a confirmação da senha.

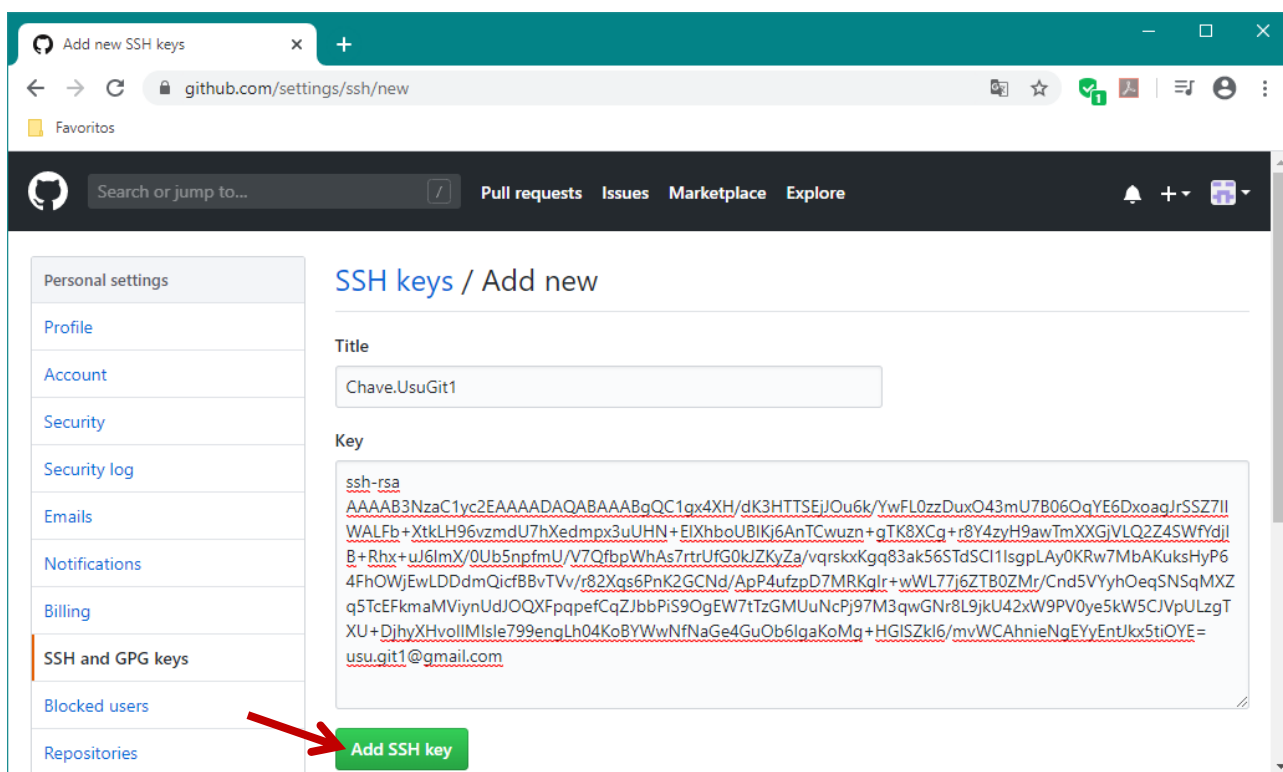
A tela a seguir, apresenta a execução desse processo. Como resultado, dois arquivos são gerados no diretório C:\Users\<usuário>\.ssh: uma chave privada (arquivo **id_rsa**) e uma chave pública (arquivo **id_rsa.pub**).



```
MINGW64:/c/Users/Leonardo
Leonardo@notesams MINGW64 ~
$ ssh-keygen -t rsa -C usu.git1@gmail.com
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/Leonardo/.ssh/id_rsa):
Created directory '/c/Users/Leonardo/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /c/Users/Leonardo/.ssh/id_rsa.
Your public key has been saved in /c/Users/Leonardo/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:U20NYfQOQBcP9yZwC58rgyKCHLgNreBzu1kqvPGMX0w usu.git1@gmail.com
The key's randomart image is:
+---[RSA 3072]-----+
|      .o.X.o      |
|..      + & +    |
|+.      * O o    |
|+*      + + +    |
|++o..E. S o o    |
|  o.+ . . . o    |
|.. . +          |
| o= *           |
| o+B           |
+----[SHA256]-----+
Leonardo@notesams MINGW64 ~
$ |
```

2. Faça login no **GitHub** (<https://github.com/>) e clique em seu avatar (no canto superior direito da página). Selecione **Settings** >> **SSH and GPG keys** >> **New SSH key**.

3. Dê um título para chave no campo **Title** e inclua o conteúdo do arquivo **id_rsa.pub** no campo **Key**. Cole este conteúdo exatamente como ele está, sem adição de espaços ou quebras de linha.



Cópia de um Repositório do GitHub para o Repositório Local – Comando clone

Como exemplo, vamos copiar um repositório hospedado no GitHub para o nosso repositório local. Para isso, usaremos o comando **clone** que, além de copiar os arquivos do projeto, traz todos os arquivos necessários para que o controle de versão dos arquivos continue sendo feito.

1. Abra o **Git Bash** e navegue até a pasta que será o seu repositório local, por exemplo:

`cd Documents/Projetos`

2. Execute o comando que copiará o repositório do GitHub para essa pasta:

`git clone https://github.com/<seu_usuario_GitHub>/repositorio.git` (usuário informado na criação da conta no GitHub)

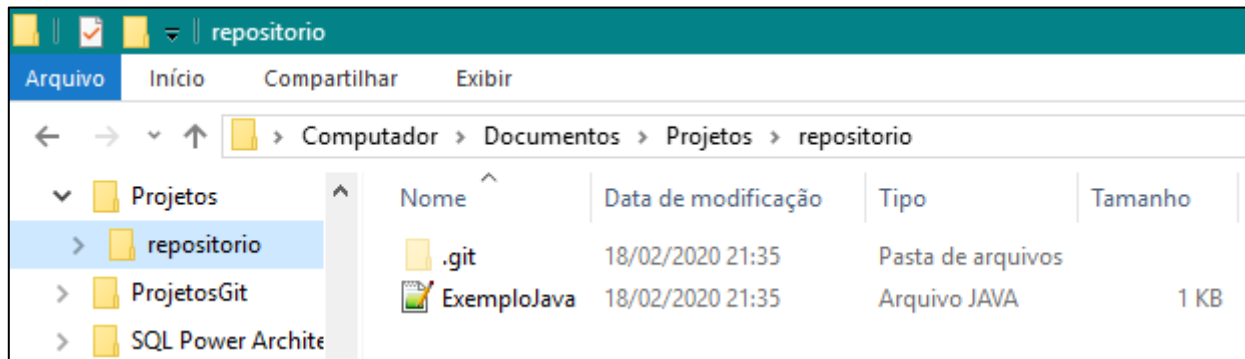
```
MINGW64:/c:/Users/Leonardo/Documents/Projetos

Leonardo@notesams MINGW64 ~
$ cd Documents/Projetos

Leonardo@notesams MINGW64 ~/Documents/Projetos
$ git clone https://github.com/usugit1/repositorio.git
Cloning into 'repositorio'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 906 bytes | 5.00 KiB/s, done.

Leonardo@notesams MINGW64 ~/Documents/Projetos
$ |
```

O comando **clone** criará uma pasta com o mesmo nome do repositório do GitHub, ou seja, "*repositorio*", e copiará para essa pasta todos os arquivos contidos nele. No projeto de exemplo, há apenas um arquivo JAVA. Note que também foi criada uma pasta oculta **.git**, na qual estão os arquivos do **Git** responsáveis pelo controle de versão do repositório.



Controle Versões – Comandos tag e checkout

Conforme um projeto de software evolui, ele vai ganhando novos *releases*. Com isso, o software passa a ter várias versões: 1.0, 1.1, 2.0, 3.0 e assim por diante.

O **Git** permite adicionar “etiquetas” no repositório, para marcar seu estado atual com um determinado número de versão. Esse processo é chamado de **tag**. É possível criar *tags* com qualquer nome, por exemplo: "v1.0", "v1.1" para definir versões preliminares, conforme a necessidade do projeto. Para visualizar as *tags* existentes num projeto, basta usar o seguinte comando:

```
git tag
```

Se um projeto possuir várias *tags*, é possível atualizar o repositório para qualquer versão. Por exemplo: Para visualizar todos os arquivos de um projeto na versão v1.0, basta recuperar esta versão. Esse processo é chamado de **checkout** e pode ser executado com o seguinte comando:

```
git checkout <tag>
```

```
MINGW64:/c/Users/Leonardo/Documents/Projetos/repositorio
Leonardo@notesams MINGW64 ~/Documents/Projetos
$ cd repositorio

Leonardo@notesams MINGW64 ~/Documents/Projetos/repositorio (master)
$ git tag
v1.0
v1.1

Leonardo@notesams MINGW64 ~/Documents/Projetos/repositorio (master)
$ git checkout v1.0
Note: switching to 'v1.0'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

  git switch -c <new-branch-name>

Or undo this operation with:

  git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at 919db0b Primeira versão do arquivo ExemploJava.java.
Leonardo@notesams MINGW64 ~/Documents/Projetos/repositorio ((v1.0))
$
```

Abra o arquivo **ExemploJava.java**, localizado no repositório local, antes e depois de fazer o **checkout**. Note que o conteúdo deles são diferentes para as versões v1.0 e v1.1.

Controle Versões – Comandos diff e blame

Para descobrir o que mudou entre uma versão e outra (e descobrir se um *bug* foi inserido em uma determinada versão, por exemplo), pode-se usar o comando **diff**, que mostrar a diferença entre duas versões de qualquer elemento do projeto. Para visualizar as diferenças entre uma versão v1.0 e outra v1.1, o seguinte comando seria usado:

```
git diff v1.0 v1.1
```

Com isso, serão mostradas as linhas existentes numa versão que não estão na outra versão, além das alterações efetuadas entre as versões selecionadas.

Também é possível descobrir quem realizou as alterações em um arquivo linha a linha. Esse recurso é útil quando não sabemos o motivo pelo qual uma alteração foi feita ou porque ela foi implementada de uma determinada maneira. Sabendo quem foi o responsável pela modificação, podemos entrar em contato para tirar esse tipo de dúvida.

Para consultar o autor de cada linha de um arquivo, pode-se usar o seguinte comando:

```
git blame <arquivo.extensão>
```

Para sair da execução do comando **blame**, basta pressionar a tecla **q**.