

Desenvolvimento de Sistemas

Comandos de versionamento: operações e funcionalidades

Introdução

Neste conteúdo, você conhecerá como funciona o processo de versionamento e entender a importância de cada uma dessas etapas. Para isso, serão digitados vários comandos na interface de linha de comando, para ver, na prática, como se trabalha com a ferramenta Git. Então, para aproveitar melhor este conteúdo, certifique-se de já ter instalado a ferramenta GitBash no seu computador.

Após compreender o ciclo de versionamento, você conhecerá algumas possíveis expansões dos comandos principais que aprenderá. Além disso, também aprenderá como realizar o versionamento de *software* utilizando ferramentas com interfaces gráficas, neste caso, o GitHub Desktop. Recomenda-se que você já tenha também essa ferramenta instalada.

Você pode consultar o conteúdo **Repositórios remotos e sistemas de gerenciamento de configuração** para informações sobre instalação e primeiros passos com GitHub Desktop e GitBash.

Operações e funcionalidades

Para exemplificar o versionamento de um projeto Java, será criado um novo projeto no Apache NetBeans IDE (*integrated development environment*, em português ambiente de desenvolvimento integrado) e, dentro da pasta do projeto, será iniciado o versionamento. Para exemplo, o projeto será chamado de **GitExemplo**.

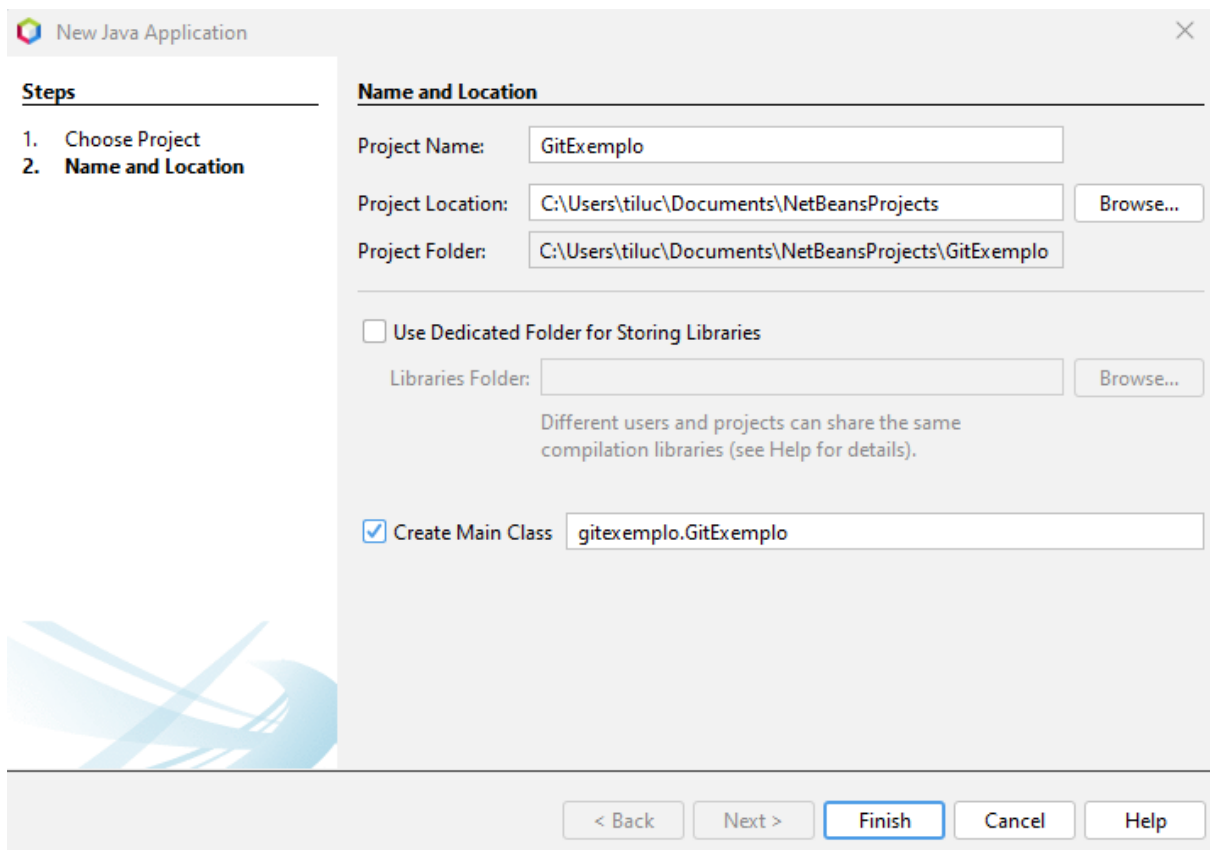
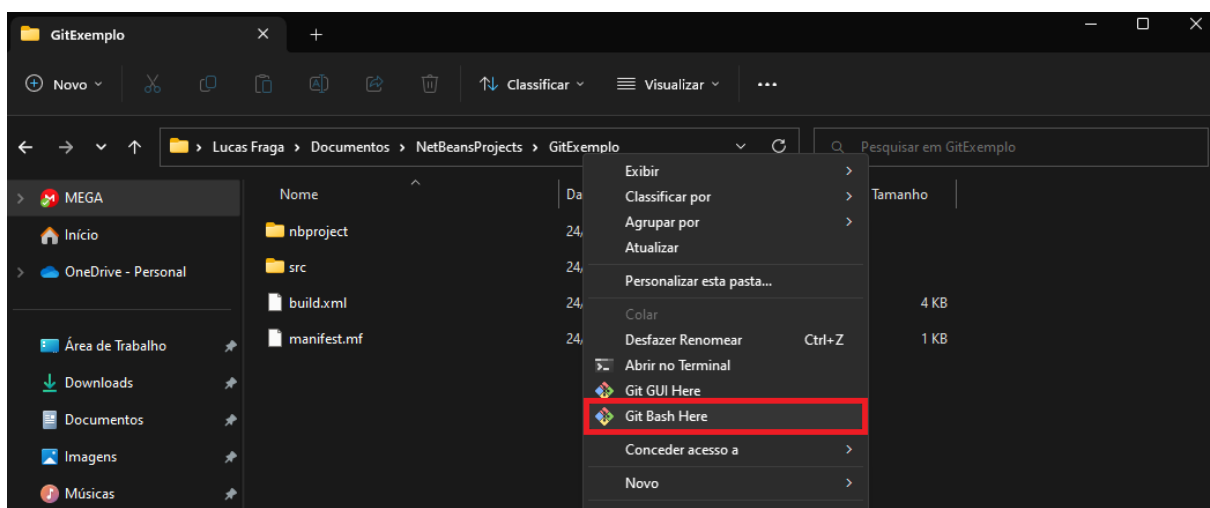


Figura 1 – Criação do projeto **GitExemplo** no NetBeans

Fonte: Apache NetBeans IDE (2023)

Durante a criação do projeto, lembre-se de verificar o local onde o projeto será criado por meio do campo **Project Location**.

Após criar o projeto, acesse esse diretório e abra o GitBash, clicando com o botão direito do *mouse* e selecionando a opção **GitBash here**.



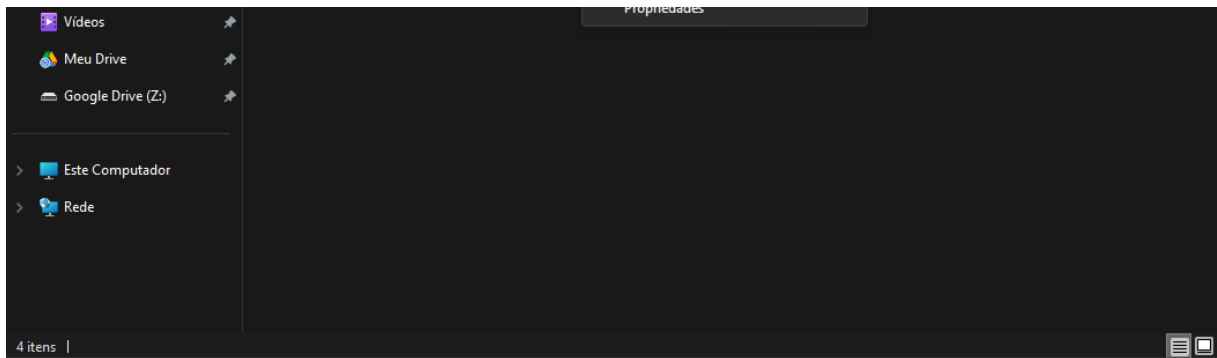


Figura 2 – Acessando o GitBash no diretório do projeto

Fonte: Senac EAD (2023)

Iniciando o versionamento

Um repositório Git rastreia e salva o histórico de todas as alterações feitas nos arquivos em um projeto. Ele salva esses dados em um diretório chamado **.git**, que é responsável por rastrear e salvar o histórico de todas as alterações feitas nos arquivos do diretório.

Para iniciar o versionamento em um diretório, utiliza-se o seguinte comando:

git init

```
MINGW64:/c/Users/tiluc/Documents/NetBeansProjects/GitExemplo
tiluc@NZXT-PHANTOM MINGW64 ~/Documents/NetBeansProjects/GitExemplo
$ git init
Initialized empty Git repository in C:/Users/tiluc/Documents/NetBeansProjects/GitExemplo/.git/
tiluc@NZXT-PHANTOM MINGW64 ~/Documents/NetBeansProjects/GitExemplo (master)
$
```

Figura 3 – Terminal do GitBash

Fonte: GitBash (2023)

Após executá-lo, será criada a pasta **.git**.

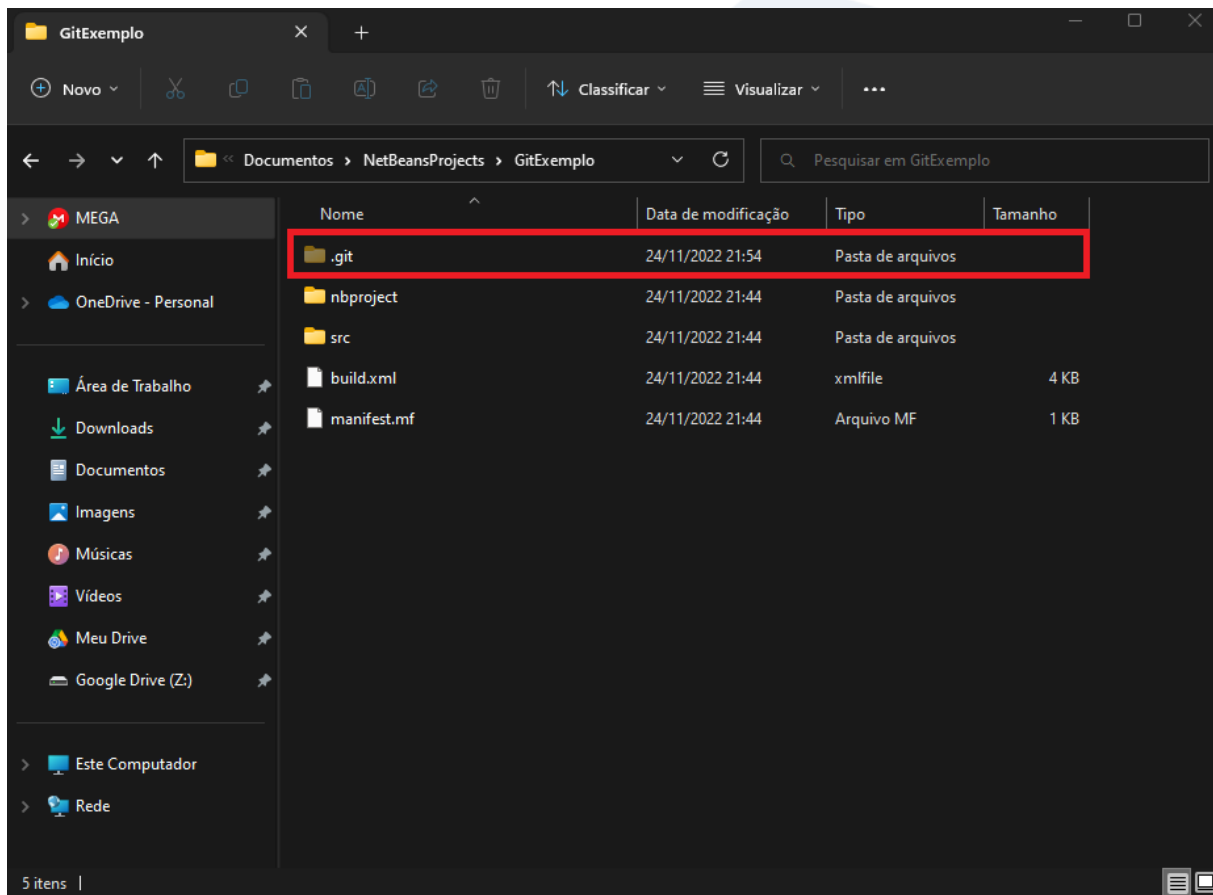


Figura 4 – Pasta do projeto com a pasta **.git**

Fonte: Senac EAD (2023)

Caso a pasta **.git** não apareça no seu diretório, isso pode significar que o seu explorador de arquivos está configurado para não exibir itens ocultos. Se deseja habilitar a exibição de itens ocultos, abra o **Explorador de Arquivos** e selecione **Exibir > Mostrar > Itens ocultos**.

Vale lembrar que a exibição dos itens ocultos não interferirá no versionamento do projeto. Então, caso não deseje habilitar essa opção, você pode ignorá-la.

Configurando o versionamento

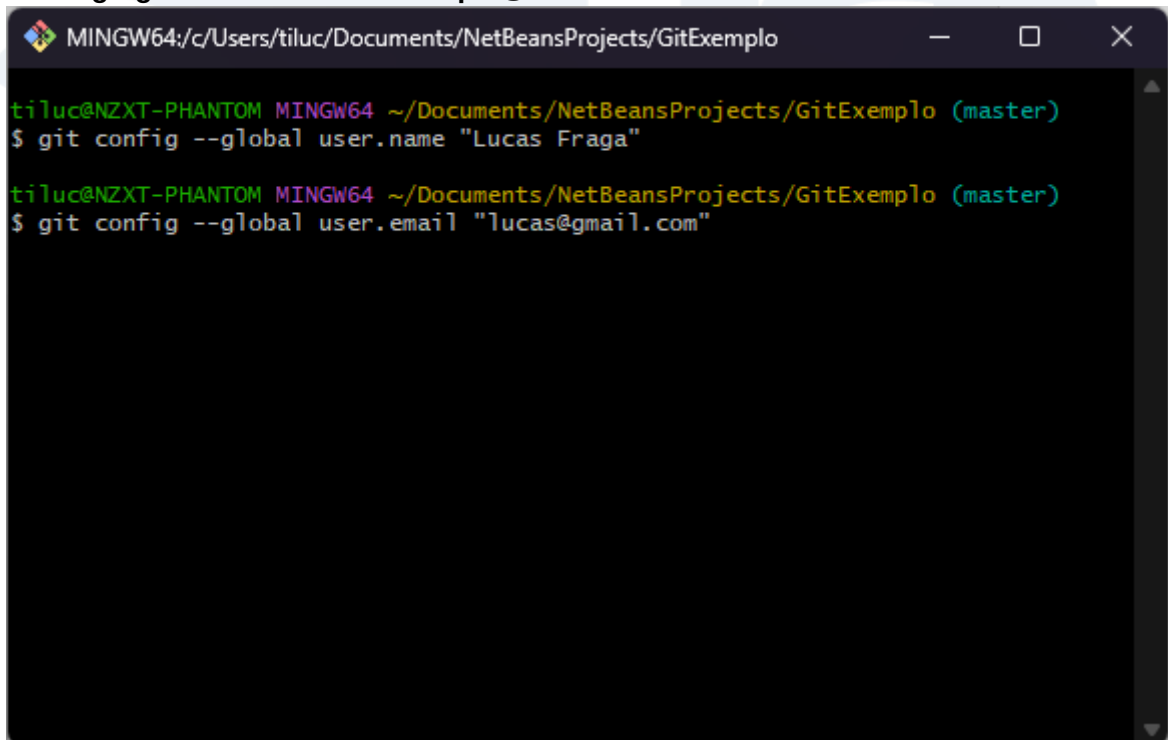
Se é a primeira vez que você está utilizando o GitBash no seu computador, então é necessário configurar o nome e o *e-mail* do seu usuário para que o versionamento reconheça quem é o responsável pelas alterações nos arquivos. Para isso, use dois comandos, um para configurar o nome e outro para configurar o *e-mail*.

Para adicionar um nome de autor do usuário atual:

git config --global user.name "Nome Sobrenome"

Para adicionar um endereço de *e-mail* a todos os **commits** do usuário atual:

git config --global user.email "exemplo@email.com"

A screenshot of a GitBash terminal window. The title bar shows the path 'MINGW64:/c/Users/tiluc/Documents/NetBeansProjects/GitExemplo'. The terminal content shows two commands being executed: first, 'git config --global user.name "Lucas Fraga"' and second, 'git config --global user.email "lucas@gmail.com"'. The prompt is '\$' and the user is 'tiluc@NZXT-PHANTOM' in a 'MINGW64' environment, currently in the directory '~/Documents/NetBeansProjects/GitExemplo' on the 'master' branch.

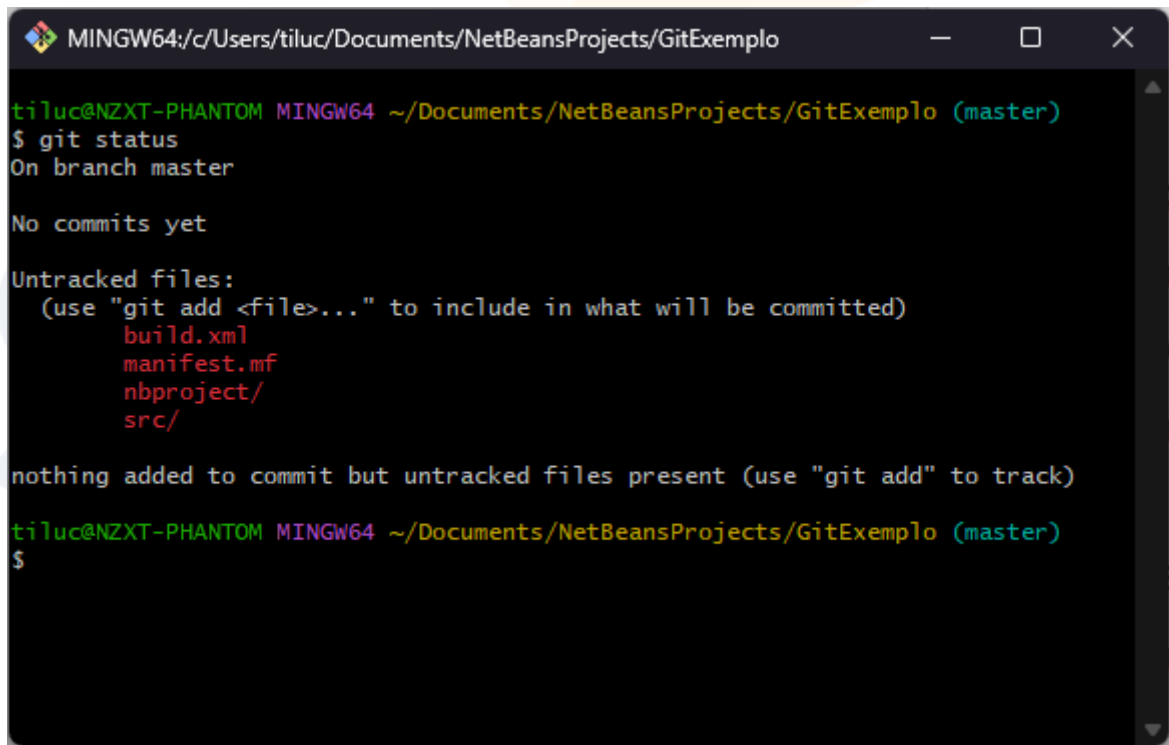
```
MINGW64:/c/Users/tiluc/Documents/NetBeansProjects/GitExemplo
tiluc@NZXT-PHANTOM MINGW64 ~/Documents/NetBeansProjects/GitExemplo (master)
$ git config --global user.name "Lucas Fraga"
tiluc@NZXT-PHANTOM MINGW64 ~/Documents/NetBeansProjects/GitExemplo (master)
$ git config --global user.email "lucas@gmail.com"
```

Figura 5 – Terminal do GitBash

Fonte: GitBash (2023)

Verificando o *status*: git status

Para verificar o *status* do versionamento, utiliza-se o comando **git status**. O comando é relativamente simples e retorna informações muito úteis sobre o que está acontecendo no versionamento dos arquivos e diretórios. Se o comando **git status** for executado no diretório do projeto, você terá a seguinte saída:



```
MINGW64:/c/Users/tiluc/Documents/NetBeansProjects/GitExemplo

tiluc@NZXT-PHANTOM MINGW64 ~/Documents/NetBeansProjects/GitExemplo (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        build.xml
        manifest.mf
        nbproject/
        src/

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

tiluc@NZXT-PHANTOM MINGW64 ~/Documents/NetBeansProjects/GitExemplo (master)
$
```

Figura 6 – Terminal do GitBash

Fonte: GitBash (2023)

Perceba que, atualmente, o versionamento identificou os arquivos **build.xml** e **manifest.mf** e os diretórios **nbproject** e **src** com o *status* **untracked**, que significa “não rastreado”.

Dentro de um repositório, os seus arquivos e as suas pastas podem ter dois *status*: **untracked** (do inglês, “não rastreado”) e **tracked** (do inglês, “rastreado”).

O *status* **untracked** é o estado inicial de todo arquivo adicionado ou editado. Logo, sempre que você iniciar o versionamento em um repositório local, os arquivos serão reconhecidos com o estado de “não rastreado”.

Sabe-se que o Git é uma ferramenta de rastreamento de versão. Essa

ferramenta nunca começará a rastrear um arquivo até que você peça especificamente a ela para fazer isso. Se você fizer alguma alteração nos arquivos, o estado deles permanecerá “não rastreado” até usar o comando para rastrear.

Preparando os arquivos

O comando **git add** adiciona arquivos novos ou alterados em seu diretório à área de preparo do Git, marcando-os para inclusão no próximo rastreamento.

Às vezes, esse comando pode passar a impressão de ser uma etapa desnecessária. Afinal, por que é preciso preparar o rastreamento dos arquivos e só depois realizá-lo de fato? Pois bem. O que faz o comando **git add** tão importante é que, por meio dele, é possível moldar a história sem alterar a forma como se trabalha.

Imagine que, no seu dia a dia como desenvolvedor de *software*, você precise programar uma determinada funcionalidade para um sistema (a qual será chamada de **funcionalidade A**). Por ser algo simples, você aproveitou para fazer a implementação de mais um recurso para adiantar o seu trabalho (**funcionalidade B**).

Após implementar e testar, você deve fazer um relatório de tudo aquilo que você fez. Esse relatório terá todo o registro do que foi feito e desfeito dentro do projeto, incluindo cada linha de código escrita ou modificada. Se algo no sistema projetado der errado, basta seguir o relatório para desfazer tudo que foi feito. Aqui, há as seguintes possibilidades:

1. Você pode implementar as duas funcionalidades e fazer um único relatório.
2. Você pode implementar a **funcionalidade A** e fazer um relatório; depois pode implementar a **funcionalidade B** e fazer mais um relatório.

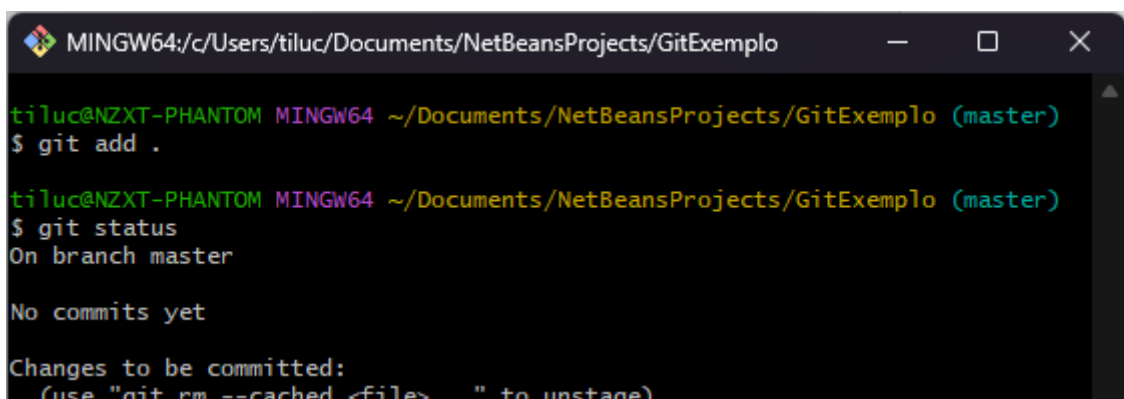
3. Você pode implementar as duas funcionalidades e fazer dois relatórios separados.

Se você optar pela opção 1, caso algo dê errado, você terá perdido um dia inteiro de trabalho, pois terá de desfazer tudo o que foi feito caso o sistema apresente algum problema. Se você optou pelas opções 2 e 3, você poderá corrigir o problema do sistema sem afetar a outra funcionalidade implementada. Então, se o problema parou de funcionar durante a implementação da **funcionalidade B**, você poderá desfazer tudo relacionado a essa funcionalidade sem afetar o que foi implementado na **funcionalidade A**.

Esta é exatamente uma das principais vantagens que o versionamento de projetos proporciona: o registro de tudo o que foi feito em um projeto, que pode ser usado para restaurar a versão anterior caso algum problema aconteça. Esse “registro” é chamado, na linguagem técnica, de **commit**, e o que se faz com o comando **git add** é dizer quais arquivos serão adicionados ao registro – em outras palavras, quais arquivos serão rastreados (**tracked**).

Como você está na fase inicial de desenvolvimento do seu projeto (afinal, apenas criou o projeto no Apache NetBeans IDE), é conveniente adicionar todos os arquivos e todas as pastas ao registro do **commit**. Para isso, é possível informar individualmente todos os arquivos ou usar o caractere ponto-final, que se refere a “tudo que estiver dentro dessa pasta”.

Nesse exemplo, você executará o segundo comando **git add .** e depois verificará o estado do versionamento com o comando **git status**.



```
MINGW64:/c/Users/tiluc/Documents/NetBeansProjects/GitExemplo
tiluc@NZXT-PHANTOM MINGW64 ~/Documents/NetBeansProjects/GitExemplo (master)
$ git add .

tiluc@NZXT-PHANTOM MINGW64 ~/Documents/NetBeansProjects/GitExemplo (master)
$ git status
On branch master

No commits yet

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



```
new file:   build.xml
new file:   manifest.mf
new file:   nbproject/build-impl.xml
new file:   nbproject/genfiles.properties
new file:   nbproject/private/private.properties
new file:   nbproject/private/private.xml
new file:   nbproject/project.properties
new file:   nbproject/project.xml
new file:   src/gitexemplo/GitExemplo.java

tiluc@NZXT-PHANTOM MINGW64 ~/Documents/NetBeansProjects/GitExemplo (master)
$ |
```

Figura 7 – Terminal do GitBash

Fonte: GitBash (2023)

Perceba que, agora, seus arquivos estão na cor verde, o que significa que eles estão prontos para ser rastreados.

Se você quisesse informar os arquivos e as pastas individualmente, o comando ficaria da seguinte maneira:

```
git add build.xml manifest.mf nbproject/* src/*
```

Perceba que, no caso das pastas, usa-se o asterisco no final. O motivo disso é simples: na interface de linha de comandos, o asterisco é conhecido como um curinga (também chamado de *wildcard*, no inglês) para completar o nome de algo. Pensando no seu comando, é preciso informar cada um dos arquivos que você quer, digitando o nome de cada um deles manualmente. Porém, graças ao asterisco, você está dizendo para o comando que usará **nome da pasta/qualquer coisa que estiver dentro dela**, o que faz economizar bastante tempo ao digitar os comandos.

Se esses conceitos são novos para você, pesquise na internet por “manipulação de arquivos e diretórios em linha de comando”, para conhecer e aprofundar-se mais no uso de interfaces de linha de comando, também conhecido como CLI (Command Line Interface).

Salvando rastreamento

No Git, um **commit** é um *snapshot* do seu repositório em um ponto específico no tempo, que permite guardar o estado do seu repositório. Dessa forma, ao longo do tempo, haverá uma “linha do tempo” do repositório, na qual estará registrado tudo o que aconteceu para que se chegasse ao resultado final.

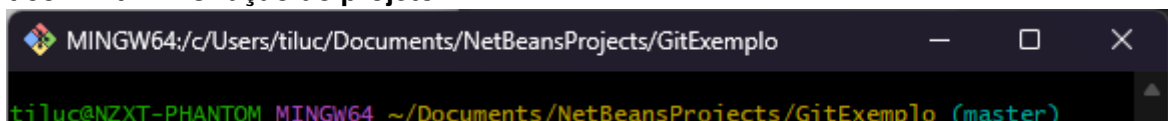
Após selecionar quais arquivos irão para o registro de **commit**, você pode salvar o rastreamento com o comando **git commit**. Esse comando tem algumas possíveis sintaxes, mas a forma mais utilizada é **git commit -m “Uma mensagem aqui...”**, na qual se inclui uma breve mensagem para que se saiba a que esse **commit** se refere (o argumento **-m** é a sintaxe que diz ao comando que será adicionada uma mensagem). Procure sempre ser breve e claro nessa mensagem, para que você, futuramente, saiba o que foi implementado nessa versão do *software*. Futuramente, você trabalhará com outros programadores no mesmo projeto, e essa informação será muito importante para que vocês consigam diferenciar uma versão da outra.

Veja algumas dicas para escrever uma boa mensagem nos seus **commits**.

- ◆ Tenha cuidado com a gramática, para não escrever palavras que possam causar confusão.
- ◆ Tente realizar um **commit** para cada alteração feita no código. Dessa forma, caso seja necessário retonar algum **commit** anterior, poucos trechos serão afetados.
- ◆ Seja transparente e breve em relação ao que foi feito. Caso implemente apenas parte de uma funcionalidade, procure especificar exatamente o que foi feito com poucas palavras.

Como você está realizando o seu primeiro **commit**, realize-o com a mensagem “Criação do projeto”, pois foi exatamente isso que você fez nesta etapa. Então, o seu comando ficará com a seguinte sintaxe:

git commit -m “Criação do projeto”



```
$ git commit -m "Criação do projeto"
[master (root-commit) 9b950b3] Criação do projeto
 9 files changed, 2023 insertions(+)
 create mode 100644 build.xml
 create mode 100644 manifest.mf
 create mode 100644 nbproject/build-impl.xml
 create mode 100644 nbproject/genfiles.properties
 create mode 100644 nbproject/private/private.properties
 create mode 100644 nbproject/private/private.xml
 create mode 100644 nbproject/project.properties
 create mode 100644 nbproject/project.xml
 create mode 100644 src/gitexemplo/GitExemplo.java

tiluc@NZXT-PHANTOM MINGW64 ~/Documents/NetBeansProjects/GitExemplo (master)
$ git status
On branch master
nothing to commit, working tree clean

tiluc@NZXT-PHANTOM MINGW64 ~/Documents/NetBeansProjects/GitExemplo (master)
$
```

Figura 8 – Comando **git commit** no terminal do GitBash

Fonte: GitBash (2023)

Caso o nome e o *e-mail* do usuário não tenham sido corretamente configurados, uma mensagem (“Author identity unknown”) aparecerá como resposta ao comando. Basta usar os comandos **git config --global user.email** e **git config --global user.name**, explicados anteriormente.

Após isso, conclui-se o rastreamento da versão do projeto. A partir daqui, sempre que você fizer alguma mudança no projeto, pode realizar um novo *snapshot* dele com os comandos **git add** e **git commit**.

O termo *snapshot* refere-se a uma captura instantânea de volume ou um retrato fiel de uma situação do ambiente. O termo vem de uma analogia à fotografia, já que é a captura de um estado em um determinado ponto no tempo.

Agora, volte para o seu código do Apache NetBeans IDE. Altere o código do seu arquivo principal e escreva um “Hello World”. Aproveitando, também limpe os comentários gerados na criação do projeto para deixar o seu código mais limpo. O código ficará da seguinte maneira:

```
package gitexemplo;  
  
public class GitExemplo {  
    public static void main(String[] args) {  
        System.out.println("Hello World");  
    }  
}
```

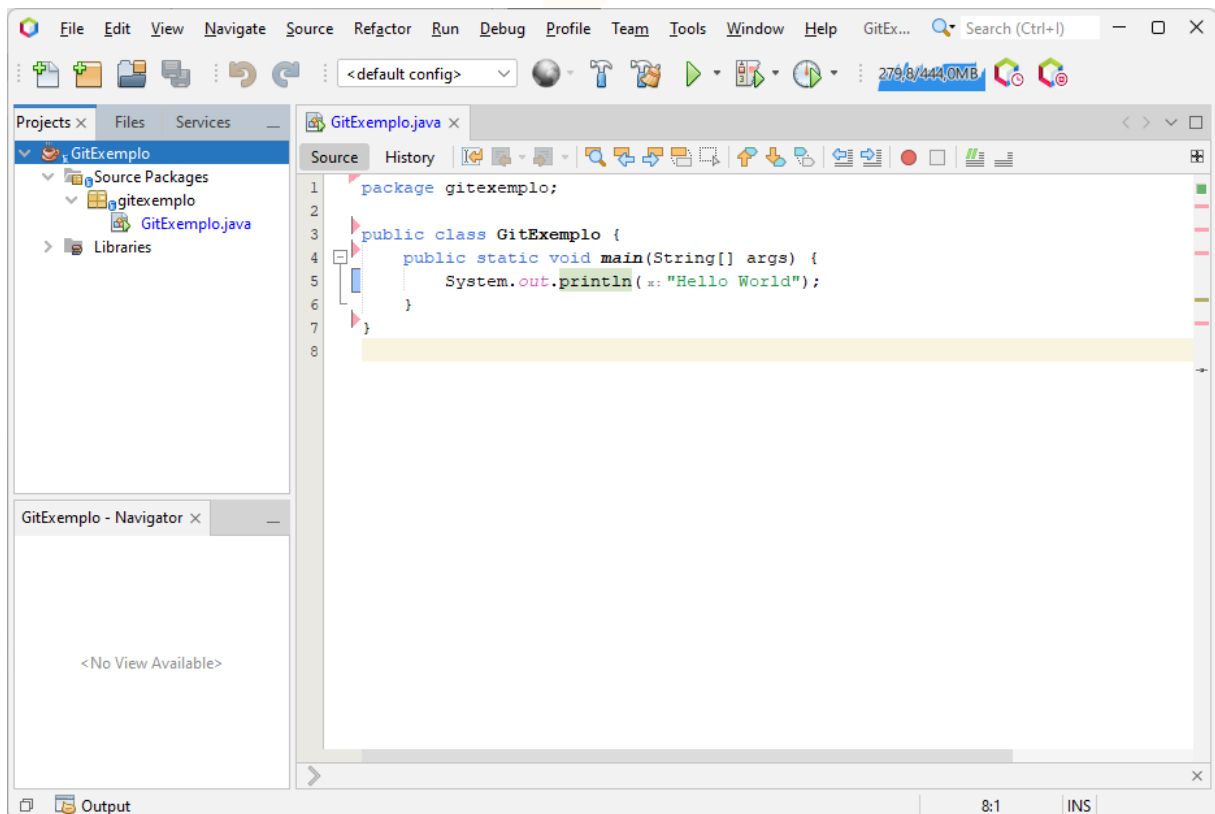


Figura 9 – Código-fonte de Hello World em Java

Fonte: Apache NetBeans IDE (2023)

No Apache NetBeans IDE, você pode ter percebido que, ao lado das linhas de código, apareceram algumas setas vermelhas. Se você clicar em alguma delas, verá que o editor de código está apontando quais trechos de código foram removidos em comparação ao último **commit** realizado. Esse recurso pode ser muito útil caso você precise consultar rapidamente as mudanças feitas no código direto pelo Apache NetBeans IDE.

Voltando ao terminal do GitBash, agora que você já concluiu as alterações do seu projeto, precisa digitar apenas dois comandos:

```
git add .  
git commit -m "Hello World"
```

Pronto! Com isso, você conclui o seu segundo **commit**. Guarde bem essa informação, pois voltará nesse assunto em breve.

Senac