

RESPONSIVE WEB DEVELOPMENT

# GIT E GITHUB

---

## LISTA DE FIGURAS

Figura 6.1 - Site oficial do GIT .....	5
Figura 6.2 - Instalando o GIT Fonte: Elaborado pelo autor (2018) .....	5
Figura 6.3 - Criando uma pasta.....	6
Figura 6.4 - Comando git init no terminal .....	7
Figura 6.5 - Comando git status Fonte: Elaborado pelo autor (2018).....	8
Figura 6.6 - Commitando um arquivo .....	9
Figura 6.7 - Git Log .....	10
Figura 6.8 - Alteração no arquivo .....	10
Figura 6.9 - Git status .....	11
Figura 6.10 - git diff .....	11
Figura 6.11 - git add index.....	12
Figura 6.12 – Microsoft compra Github Fonte: MSDN Blogs.....	14

## SUMÁRIO

6 GIT E GITHUB .....	4
6.1 Sistema de Controle de Versão.....	4
6.1.2 Instalação e configuração do GIT.....	5
6.1.3 Estados dos arquivos e commits.....	8
6.1.4 Navegação entre as versões.....	9
6.1.5 Desfazendo alterações.....	12
6.2 Organizando o trabalho .....	12
6.2.1 Trabalhando com Branches e Tags .....	13
6.2.2 Merge .....	13
6.2.3 Resolução de conflitos .....	13
6.2.4 Ignorando arquivos do repositório .....	13
6.3 Conhecendo o Git Hub.....	14
6.3.1 Trabalhando com repositórios remotos .....	14
6.3.2 GIT PULL .....	14
6.3.3 GIT PUSH .....	15
6.4 Boas práticas .....	15
REFERÊNCIAS.....	16

## 6 GIT E GITHUB

A vida como programador não é nada fácil. A cobrança por entregas rápidas e atualizações de códigos é constante! Mas, apesar da urgência, temos que prestar atenção no que estamos fazendo, por mais que seja uma alteração simples. Corremos ainda o risco de fazer alguma alteração e "quebrar" o que estava funcionando. Complicado, não é mesmo?

### 6.1 Sistema de Controle de Versão

Para entendermos todas as mudanças em nosso projeto não basta comentar o código, temos que criar um histórico de alterações. Mas, imagina o seguinte cenário: criar um histórico de um projeto em produção com uma equipe de vários desenvolvedores trabalhando ao mesmo tempo. Com certeza nosso projeto ficará confuso, e é neste cenário que entrará o sistema de controle de versão.

Os sistemas de controle de versão existem desde os anos 1990, mas foi em 2005 que Linus Torvalds, sim o criador do Linux, desenvolveu um novo sistema chamado GIT.

Hoje em dia, saber como usar o GIT virou uma habilidade fundamental para todo desenvolvedor.

GIT e GitHub, são a mesma coisa?

Não, o GitHub é uma famosa rede social de compartilhamentos de código, é um repositório de códigos fontes que podem ser privados ou abertos, de desenvolvedores iniciantes até grandes empresas. O GitHub foi criado em 2008 e comprado pela Microsoft 10 anos mais tarde.

Agora que você já sabe o que é um sistema de controle de versão, vamos colocar a mão na massa?

### 6.1.2 Instalação e configuração do GIT

Antes de trabalharmos com o GIT, precisamos instalá-lo em nossa máquina.

Para instalar, você pode acessar o site do GIT: [git-scm.com](https://git-scm.com)

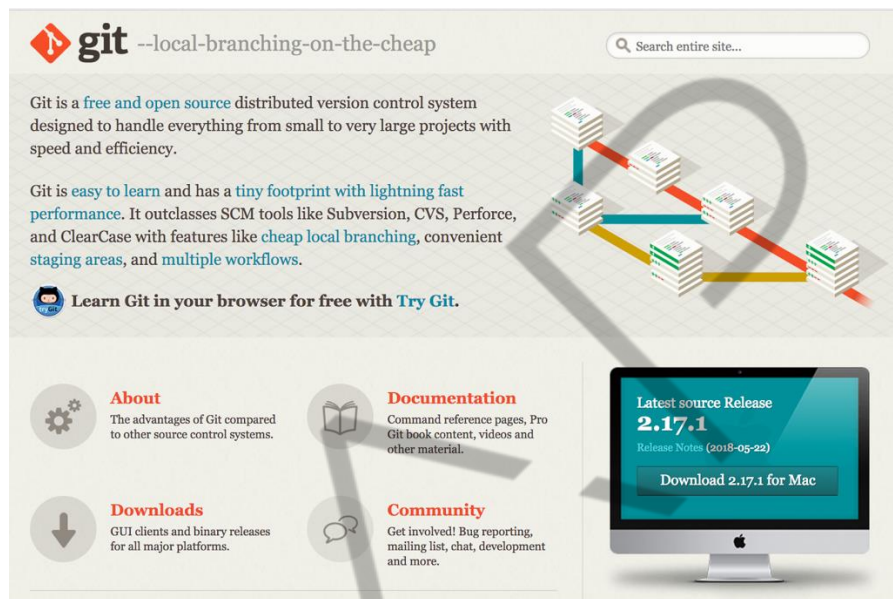


Figura 6.1 - Site oficial do GIT  
Fonte: [git.scm.com](https://git-scm.com)

O site irá identificar o seu sistema operacional, e então, é só clicar no botão download. Após baixar o arquivo executável, o processo de instalação é simples, easy-to-install:

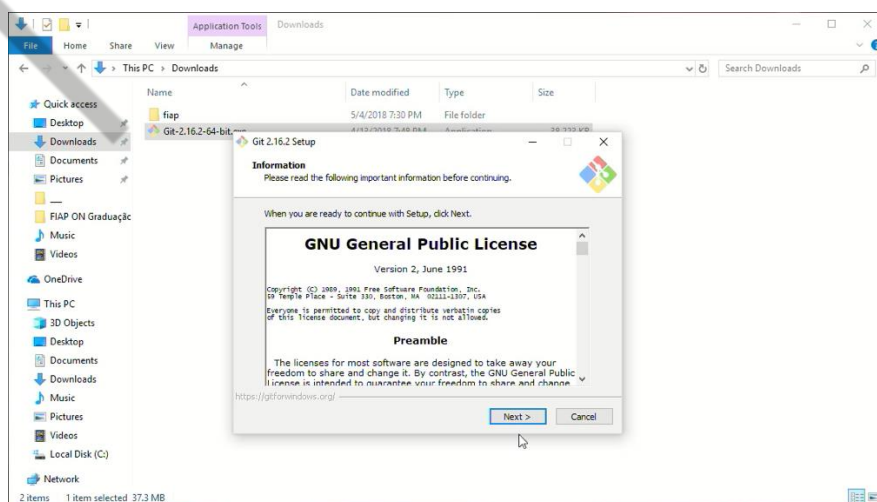


Figura 6.2 - Instalando o GIT  
Fonte: Elaborado pelo autor (2018)

Com o GIT instalado em sua máquina, chegou a hora de nos apresentarmos para ele, ou seja, configurar o sistema para ele identificar o desenvolvedor responsável pelas alterações.

Para criarmos o nosso primeiro diretório, você deverá criar uma pasta (pode ser no desktop mesmo) e com o botão direito clicar na opção Git Bash Here:

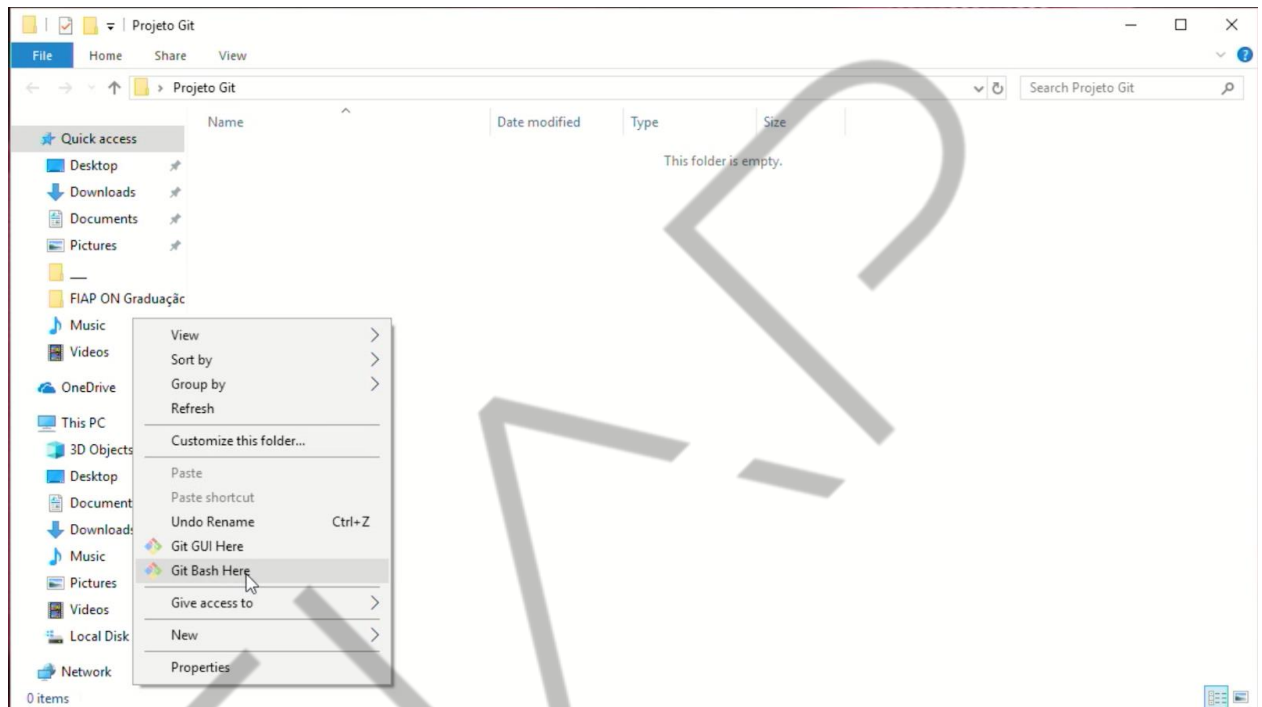
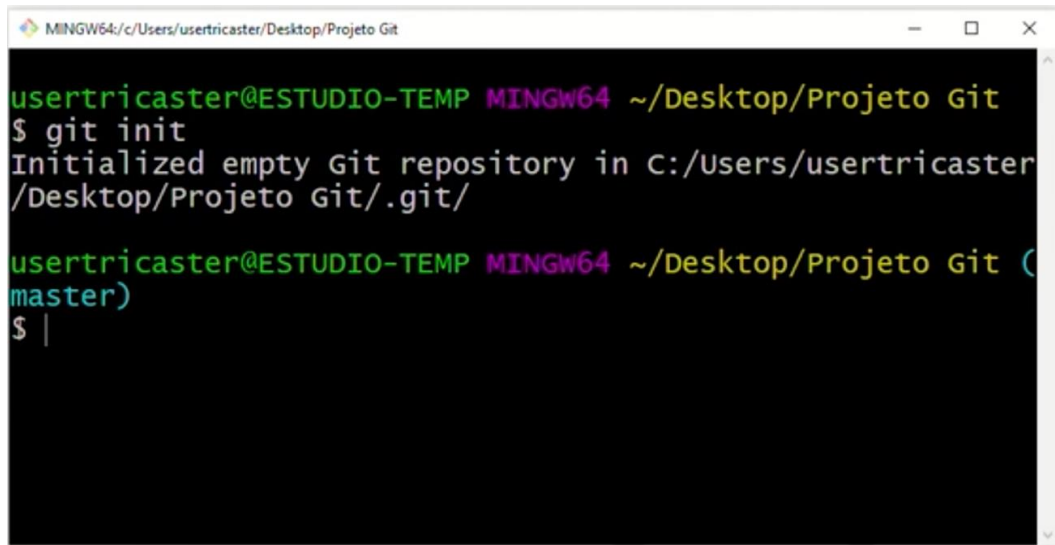


Figura 6.3 - Criando uma pasta  
Fonte: Elaborado pelo autor (2018)

No exemplo “Criando uma pasta”, o nome da pasta criada é: Projeto Git

Após clicar em Git Bash Here o nosso terminal se abrirá, e você digitará o primeiro comando: **git init**.

A screenshot of a terminal window titled 'MINGW64: c:/Users/usertricaster/Desktop/Projeto Git'. The prompt is 'usertricaster@ESTUDIO-TEMP MINGW64 ~/Desktop/Projeto Git'. The command '\$ git init' has been entered, and the output is 'Initialized empty Git repository in C:/Users/usertricaster/Desktop/Projeto Git/.git/'. The prompt is now 'usertricaster@ESTUDIO-TEMP MINGW64 ~/Desktop/Projeto Git (master)' with a cursor on a new line.

```
usertricaster@ESTUDIO-TEMP MINGW64 ~/Desktop/Projeto Git
$ git init
Initialized empty Git repository in C:/Users/usertricaster/Desktop/Projeto Git/.git/

usertricaster@ESTUDIO-TEMP MINGW64 ~/Desktop/Projeto Git (master)
$ |
```

Figura 6.4 - Comando git init no terminal  
Fonte: Elaborado pelo autor (2018)

Após o comando git init você notará uma pasta oculta chamada .git, isto quer dizer que iniciamos o nosso repositório, mas precisamos configurá-lo com o nome e o e-mail.

Para isso, basta digitar no terminal o comando:

```
git config --global user.name "Gustavo Torrente"
```

```
git config --global user.email "email@fiap.com.br"
```

Dica: realizamos esta configuração global para evitar que em todo projeto seja necessária uma nova configuração.

Para verificar se o nome e e-mail foram setados corretamente, basta entrar com os comandos:

```
git config user.name
```

```
git config user.email
```

### 6.1.3 Estados dos arquivos e commits

Dentro da sua pasta: Projeto Git, crie um arquivo index.txt.

E agora em seu terminal, execute o comando: git status, este comando nos informa o estado de nosso arquivo:

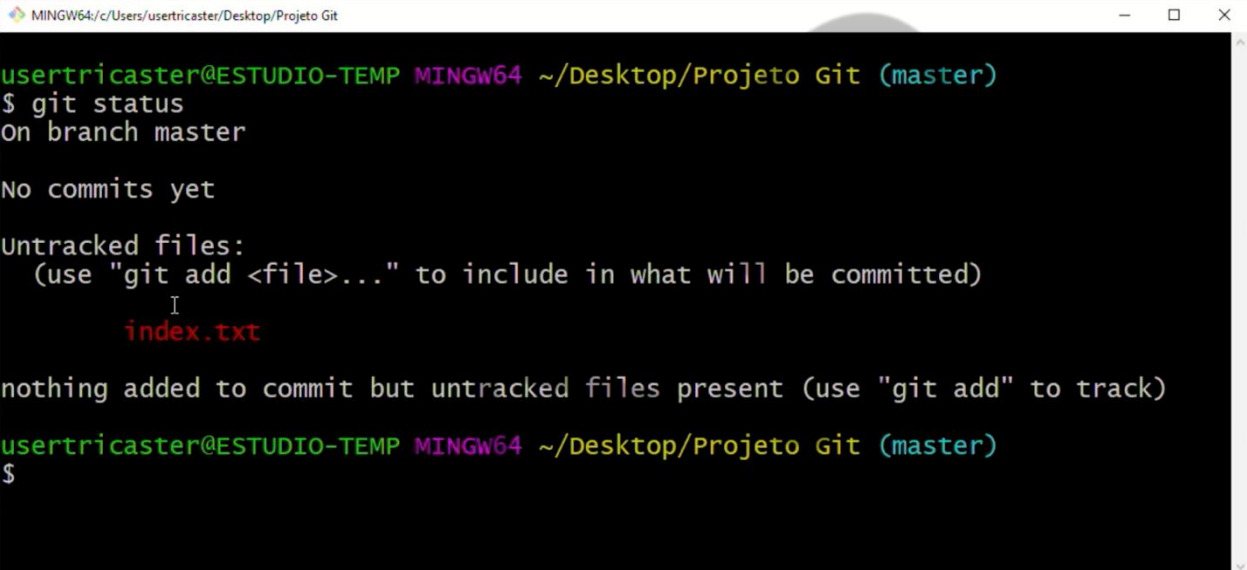
A screenshot of a terminal window titled 'MINGW64/c/Users/usertricaster/Desktop/Projeto Git'. The prompt is 'usertricaster@ESTUDIO-TEMP MINGW64 ~/Desktop/Projeto Git (master)'. The user enters '\$ git status'. The output is: 'on branch master', 'No commits yet', 'Untracked files:', '(use "git add <file>..." to include in what will be committed)', 'index.txt', and 'nothing added to commit but untracked files present (use "git add" to track)'. The prompt is then '\$'.

Figura 6.5 - Comando git status  
Fonte: Elaborado pelo autor (2018)

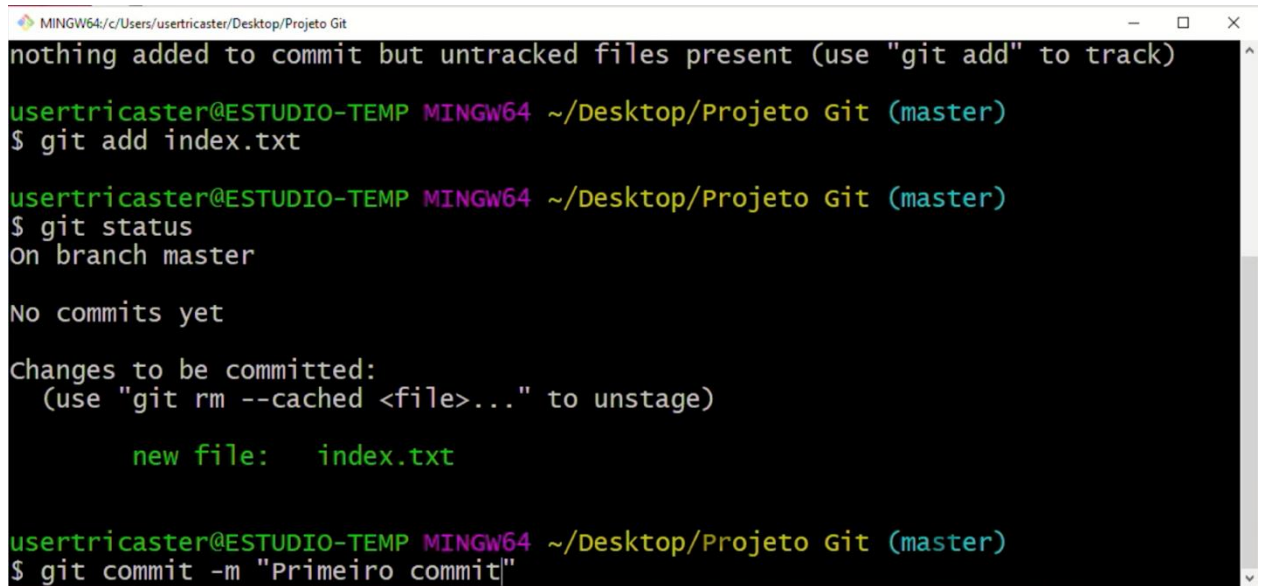
Neste caso, a mensagem retornada é: untracked, ou seja, um arquivo desconhecido para o GIT.

O GIT entende que existe um arquivo, mas não identifica nenhum estado.

Precisamos informar para o GIT que este arquivo deve ser versionado e, para isso, vamos executar dois comandos.

O primeiro comando é seguido do nome do arquivo: git add index.txt. E o segundo comando é responsável pela gravação no repositório:





```
nothing added to commit but untracked files present (use "git add" to track)
usertricaster@ESTUDIO-TEMP MINGW64 ~/Desktop/Projeto Git (master)
$ git add index.txt

usertricaster@ESTUDIO-TEMP MINGW64 ~/Desktop/Projeto Git (master)
$ git status
On branch master

No commits yet

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

        new file:   index.txt

usertricaster@ESTUDIO-TEMP MINGW64 ~/Desktop/Projeto Git (master)
$ git commit -m "Primeiro commit"
```

Figura 6.6 - Commitando um arquivo  
Fonte: Elaborado pelo autor (2018)

**git commit -m "Primeiro commit"**

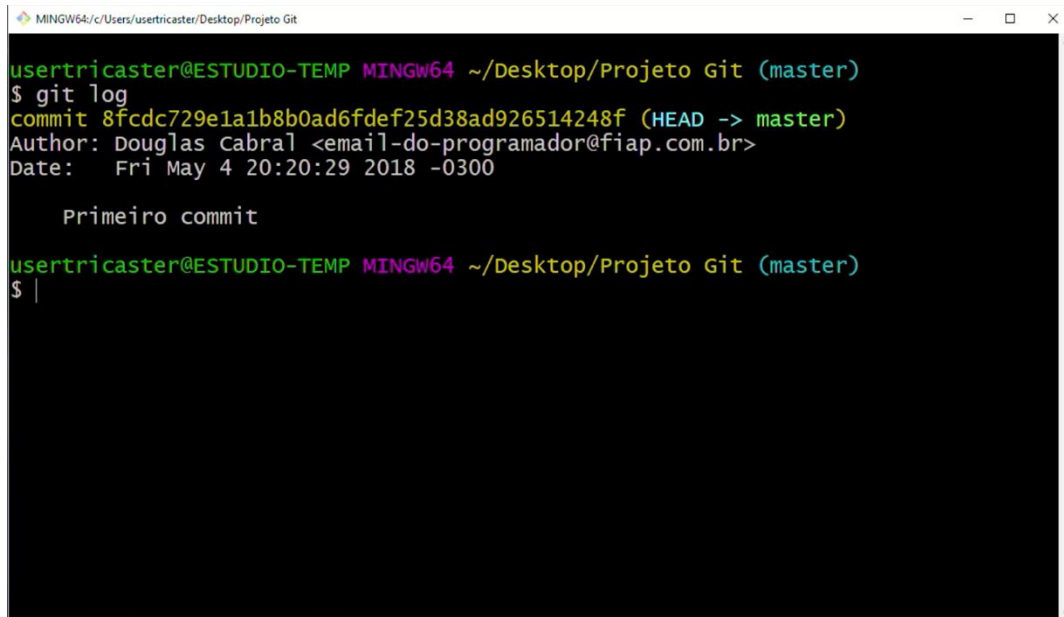
Pronto! Agora temos o primeiro versionamento do nosso arquivo.

#### 6.1.4 Navegação entre as versões

Agora que você já sabe como instalar, configurar e criar versionamentos do seu projeto, vamos aprender a navegar entre as versões.

Para verificar todo histórico de mudanças em nosso repositório, utilizamos o seguinte comando:

**git log**



```
usertricaster@ESTUDIO-TEMP MINGW64 ~/Desktop/Projeto Git (master)
$ git log
commit 8fcdc729e1a1b8b0ad6fdef25d38ad926514248f (HEAD -> master)
Author: Douglas Cabral <email-do-programador@fiap.com.br>
Date:   Fri May 4 20:20:29 2018 -0300

    Primeiro commit

usertricaster@ESTUDIO-TEMP MINGW64 ~/Desktop/Projeto Git (master)
$ |
```

Figura 6.7 - Git Log  
Fonte: Elaborado pelo autor (2018)

O git log nos retorna um ID, o autor, data e a própria alteração realizada.

Como não realizamos mudanças em nosso arquivo ainda não temos como navegar, então abra o seu arquivo em branco index.txt e salve uma alteração. Conforme o exemplo abaixo:

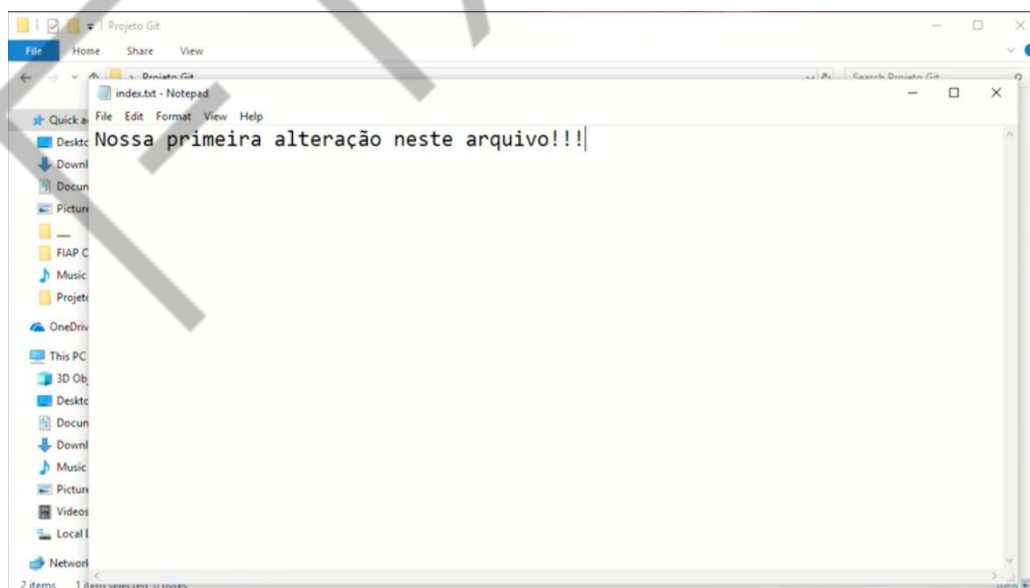
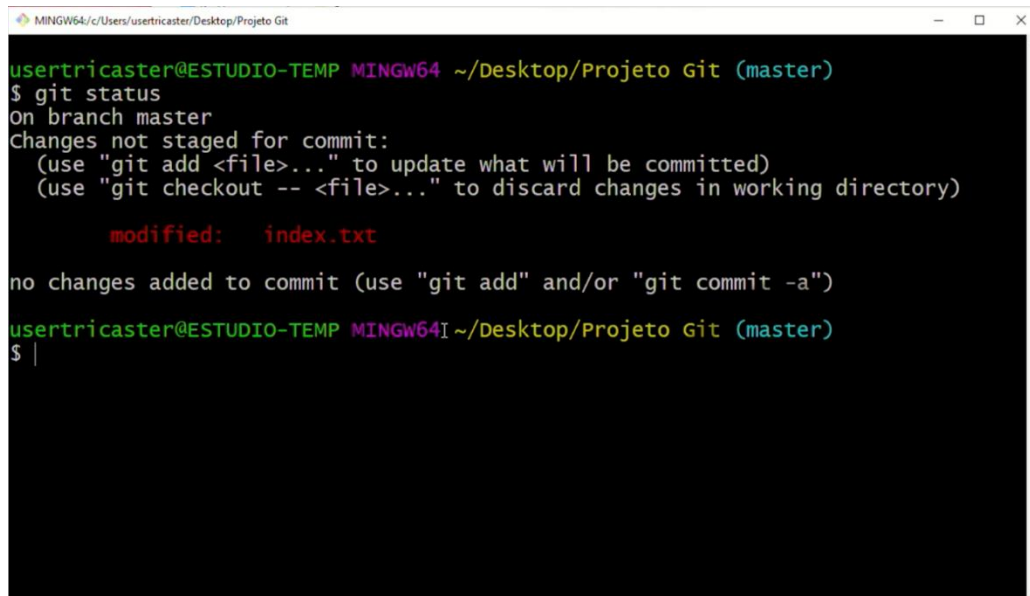


Figura 6.8 - Alteração no arquivo  
Fonte: Elaborado pelo autor (2018)

Agora com nossa primeira alteração feita, vamos retornar ao terminal/prompt de comando para executar o comando `git status`:

A terminal window titled 'MINGW64/c/Users/usertricaster/Desktop/Projeto Git' showing the output of the 'git status' command. The output indicates that the file 'index.txt' has been modified but is not yet staged for commit. It provides instructions on how to stage the changes using 'git add' or discard them using 'git checkout --'.

```
usertricaster@ESTUDIO-TEMP MINGW64 ~/Desktop/Projeto Git (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

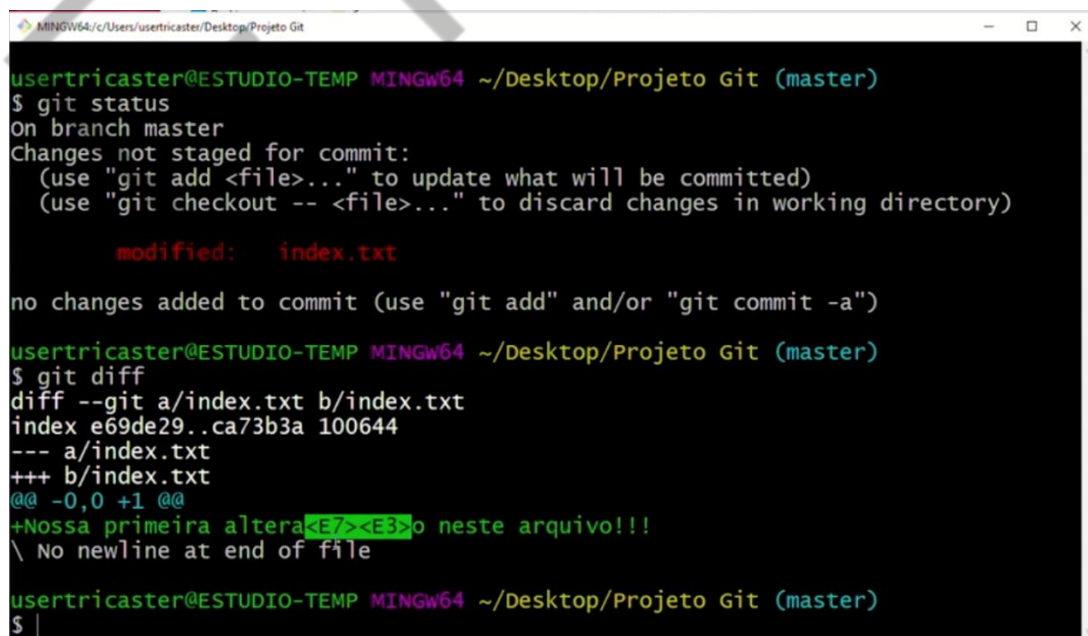
        modified:   index.txt

no changes added to commit (use "git add" and/or "git commit -a")
usertricaster@ESTUDIO-TEMP MINGW64I ~/Desktop/Projeto Git (master)
$ |
```

Figura 6.9 - Git status  
Fonte: Elaborado pelo autor (2018)

Repare que ao executar este comando a mensagem em vermelho informa: *modified*.

O GIT já sabe que aquele arquivo foi alterado e para visualizarmos esta mudança, um novo comando: **git diff**

A terminal window showing the output of the 'git diff' command. It displays the differences between the current working directory and the last commit for the file 'index.txt'. The output shows that the file has been modified and provides a diff summary. A custom message is also visible in the diff output.

```
usertricaster@ESTUDIO-TEMP MINGW64 ~/Desktop/Projeto Git (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

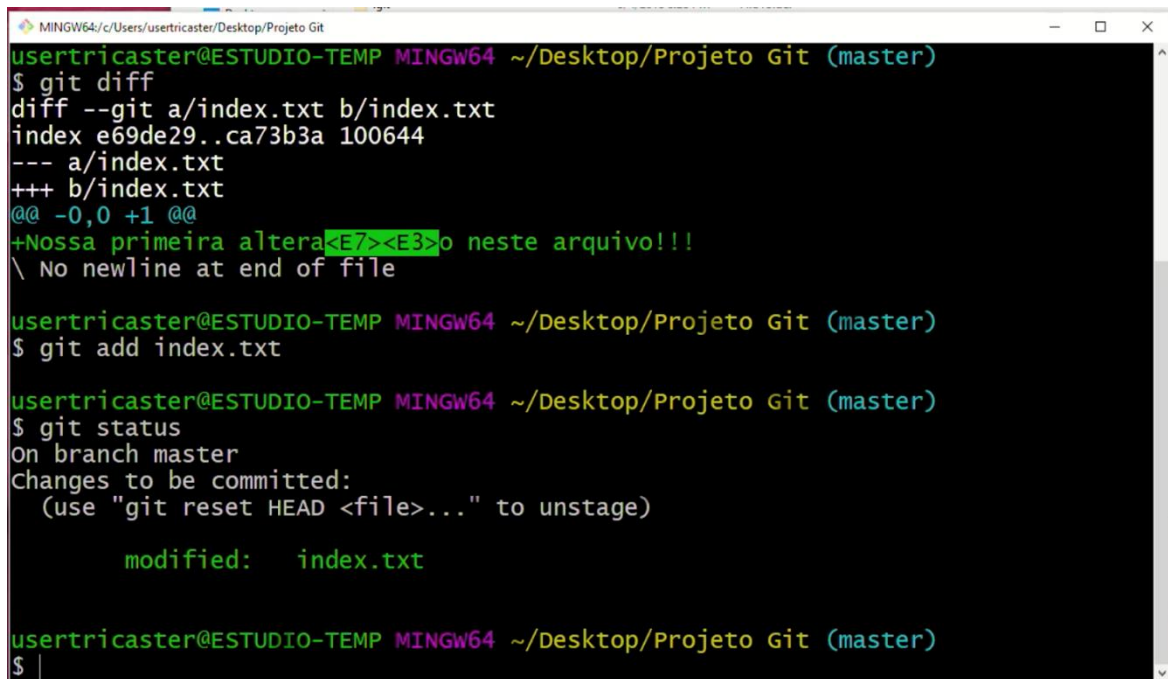
        modified:   index.txt

no changes added to commit (use "git add" and/or "git commit -a")
usertricaster@ESTUDIO-TEMP MINGW64 ~/Desktop/Projeto Git (master)
$ git diff
diff --git a/index.txt b/index.txt
index e69de29..ca73b3a 100644
--- a/index.txt
+++ b/index.txt
@@ -0,0 +1 @@
+Nossa primeira altera<E7><E3>o neste arquivo!!!
\ No newline at end of file
usertricaster@ESTUDIO-TEMP MINGW64 ~/Desktop/Projeto Git (master)
$ |
```

Figura 6.10 - git diff  
Fonte: Elaborado pelo autor (2018)

Repare que após executarmos este comando o GIT nos informa qual foi a alteração (texto em verde) e quantas alterações foram realizadas (texto em azul).

O próximo passo é versionar este arquivo e para isso utilize o comando que você já aprendeu: `git add index.html`.



```
usertricaster@ESTUDIO-TEMP MINGW64 ~/Desktop/Projeto Git (master)
$ git diff
diff --git a/index.txt b/index.txt
index e69de29..ca73b3a 100644
--- a/index.txt
+++ b/index.txt
@@ -0,0 +1 @@
+Nossa primeira altera<E7><E3>o neste arquivo!!!
\ No newline at end of file

usertricaster@ESTUDIO-TEMP MINGW64 ~/Desktop/Projeto Git (master)
$ git add index.txt

usertricaster@ESTUDIO-TEMP MINGW64 ~/Desktop/Projeto Git (master)
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   index.txt

usertricaster@ESTUDIO-TEMP MINGW64 ~/Desktop/Projeto Git (master)
$
```

Figura 6.11 - git add index  
Fonte: Elaborado pelo autor (2018)

Veja mais no vídeo: **Navegando entre versões**

### 6.1.5 Desfazendo alterações

Durante o desenvolvimento de um projeto, erros podem acontecer e a necessidade de voltar para uma versão anterior do seu código se torna necessária.

Veja no vídeo: **Desfazendo alterações**, como desfazemos alterações.

## 6.2 Organizando o trabalho

Para aumentarmos a produtividade em sistemas de controle de versão, temos a possibilidade de nomear as hashes, criando tags e versões paralelas do nosso código.

### 6.2.1 Trabalhando com Branches e Tags

Aprenda a organizar o seu trabalho durante o desenvolvimento e cada comando executado com o git no vídeo:

Trabalhando com Branches e Tags

### 6.2.2 Merge

Uma vez que trabalhamos com branches temos várias linhas de desenvolvimento. Quando trabalhamos em equipe cada um pode seguir a sua branch, mas o que acontece na hora de integrar todas estas mudanças e alterações? Aprenda no vídeo **Merge** como mesclar nossas alterações.

### 6.2.3 Resolução de conflitos

Conflitos são inevitáveis, principalmente quando estamos trabalhando em equipe. Imagine a seguinte situação: Trabalho remoto com 2 programadores alterando partes diferentes do código.

O GIT fica confuso e não sabe como proceder!

Veja como resolver estes erros no vídeo:

Resolução de conflitos.

### 6.2.4 Ignorando arquivos do repositório

E quando surgirem situações onde temos arquivos em nosso repositório, mas não queremos que eles sejam rastreados? Existem diversos motivos para isso acontecer e para que não atrapalhe a sua produtividade, aprenda no vídeo:

**Ignorando arquivos do repositório** a como ignorá-los.

### 6.3 Conhecendo o Git Hub

O Github é a maior rede social de código aberto utilizando o sistema de controle de versão GIT. Em 2018 a Microsoft comprou o site Github por US\$ 7,5 bilhões.

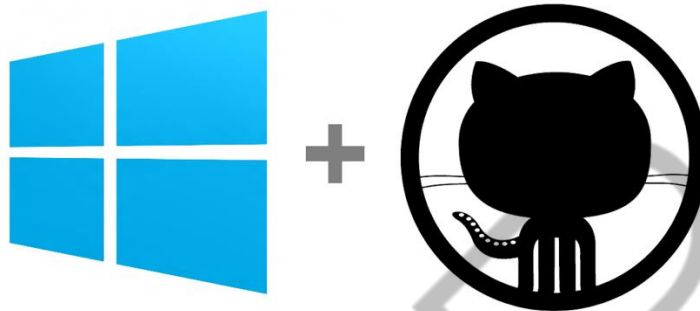


Figura 6.12 – Microsoft compra Github  
Fonte: MSDN Blogs

Aprenda no vídeo **Conhecendo o Github** a como criar sua conta no Github e começar a versionar o seu código através desta plataforma.

#### 6.3.1 Trabalhando com repositórios remotos

Você já aprendeu a criar repositórios locais, mas através do Github você tem a possibilidade de criar repositórios remotos. Quer saber como enviar seu repositório local para o Github? Veja o passo a passo no vídeo:

Trabalhando com repositórios remotos

#### 6.3.2 GIT PULL

Existem situações em que temos arquivos atualizados em nosso repositório remoto e arquivos que não estão atualizados no repositório local.

O git pull é responsável pela atualização de todas as versões, ideal para quando você está trabalhando off-line e seus arquivos locais ainda não estão sincronizados com o repositório remoto. Ficou curioso para conhecer este comando?

Assista o vídeo **Git pull**

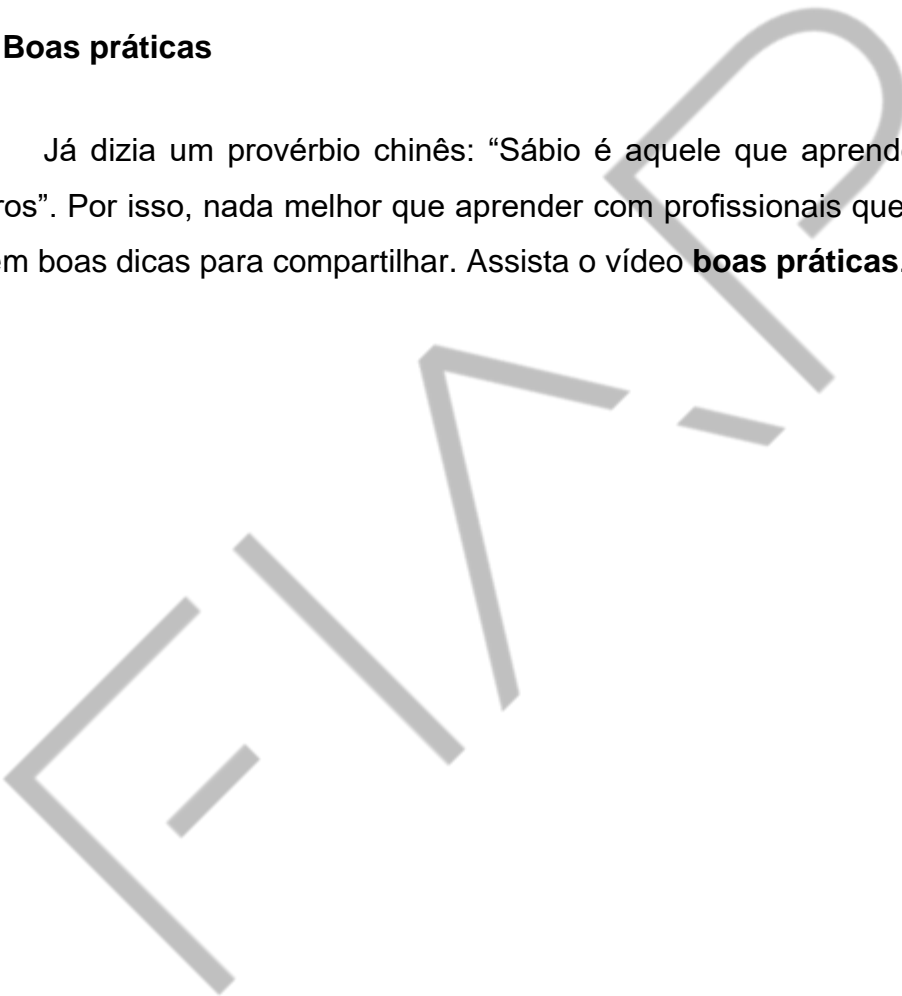
### 6.3.3 GIT PUSH

O comando git push é o inverso do git pull, é através dele que você envia seus arquivos atualizados para o seu repositório remoto, no caso o Git hub.

Veja no vídeo **git push** como realizar o passo a passo para enviar os seus arquivos.

### 6.4 Boas práticas

Já dizia um provérbio chinês: “Sábio é aquele que aprende com o erro dos outros”. Por isso, nada melhor que aprender com profissionais que já atuam na área e têm boas dicas para compartilhar. Assista o vídeo **boas práticas**.



## REFERÊNCIAS

AQUILES, Alexandre. **Controlando versões com git e github**. São Paulo: Editora Casa do Código, 2016.

GIT. Disponível em: <<https://git-scm.com/>>. Acesso em: 27 jun. 2018.

AVANADE