

TÓPICOS AVANÇADOS DE
DESENVOLVIMENTO ANDROID

INTRODUÇÃO A **VERSIONAMENTO** **DE CÓDIGO** E REPOSITÓRIOS

GUSTAVO TORRENTE



1

LISTA DE FIGURAS

Figura 1.1 - Site oficial do GIT	5
Figura 1.2 - Instalando o GIT	6
Figura 1.3 - Criando uma pasta	7
Figura 1.4 - Comando git init no terminal	7
Figura 1.5 - Comando git status	8
Figura 1.6 - Commitando um arquivo	9
Figura 1.7 - Comando git log	10
Figura 1.8 - Alteração no arquivo	10
Figura 1.9 - Comando git status	11
Figura 1.10 - Comando git diff	11
Figura 1.11 - Comando git add index	12
Figura 1.12 - Microsoft compra GitHub	14

SUMÁRIO

1 INTRODUÇÃO A VERSIONAMENTO DE CÓDIGO E REPOSITÓRIOS.....	4
1.1 Sistema de controle de versão	4
1.1.2 Instalação e configuração do Git	5
1.1.3 Estados dos arquivos e commits	8
1.1.4 Navegação entre as versões	9
1.1.5 Desfazendo alterações	12
1.2 Organizando o trabalho	12
1.2.1 Trabalhando com Branches e Tags	13
1.2.2 Merge	13
1.2.3 Resolução de conflitos	13
1.2.4 Ignorando arquivos do repositório	13
1.3 Conhecendo o GitHub	13
1.3.1 Trabalhando com repositórios remotos	14
1.3.2 Git Pull	14
1.3.3 Git Push	14
1.4 Boas práticas	15
REFERÊNCIAS	16

1 INTRODUÇÃO A VERSIONAMENTO DE CÓDIGO E REPOSITÓRIOS

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 "quebrar" o que estava funcionando.

Num ambiente em que há bastante colaboração, também é necessário manter a equipe inteira trabalhando em um mesmo projeto ou em uma mesma base de código para lidar com alterações frequentes nas aplicações.

Não importa qual a linguagem (Kotlin, Swift, Javascript etc.) ou a base de código utilizada. Versionar aplicações é o ponto-chave e inicial para a organização das aplicações, como também sua própria evolução.

1.1 Sistema de controle de versão

Para entendermos todas as mudanças em nosso projeto, não basta apenas comentar o código – temos, portanto, que criar um histórico de alterações. Mas imagine 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; sendo assim, será nesse 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, o criador do Linux, desenvolveu um novo sistema chamado Git, incorporando uma série de melhorias nos antigos sistemas de controle de versão.

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

Seriam, então, Git e GitHub 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.

1.1.2 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

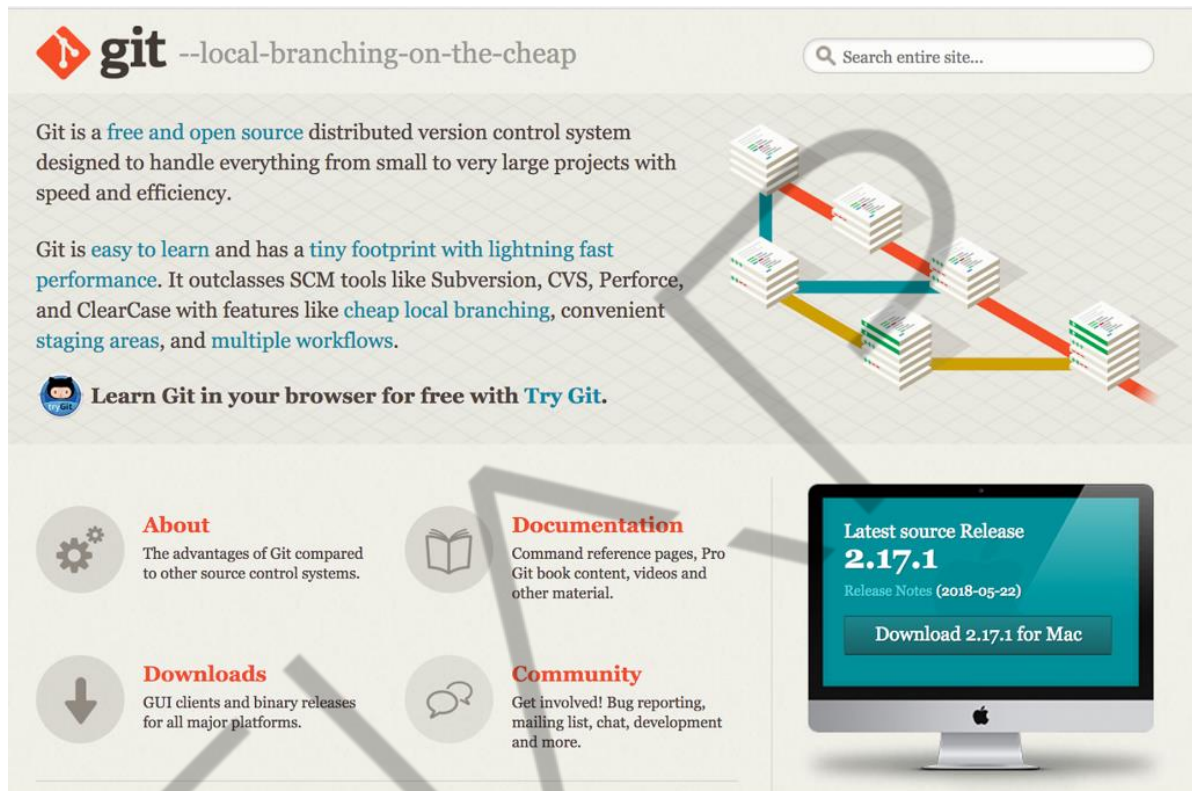


Figura 1.1 - Site oficial do GIT
Fonte: GIT (2020)

O site identificará o seu sistema operacional e então é só clicar no botão download. Após baixar o arquivo executável, inicia-se o processo de instalação, que é bem simples: *easy-to-install*:

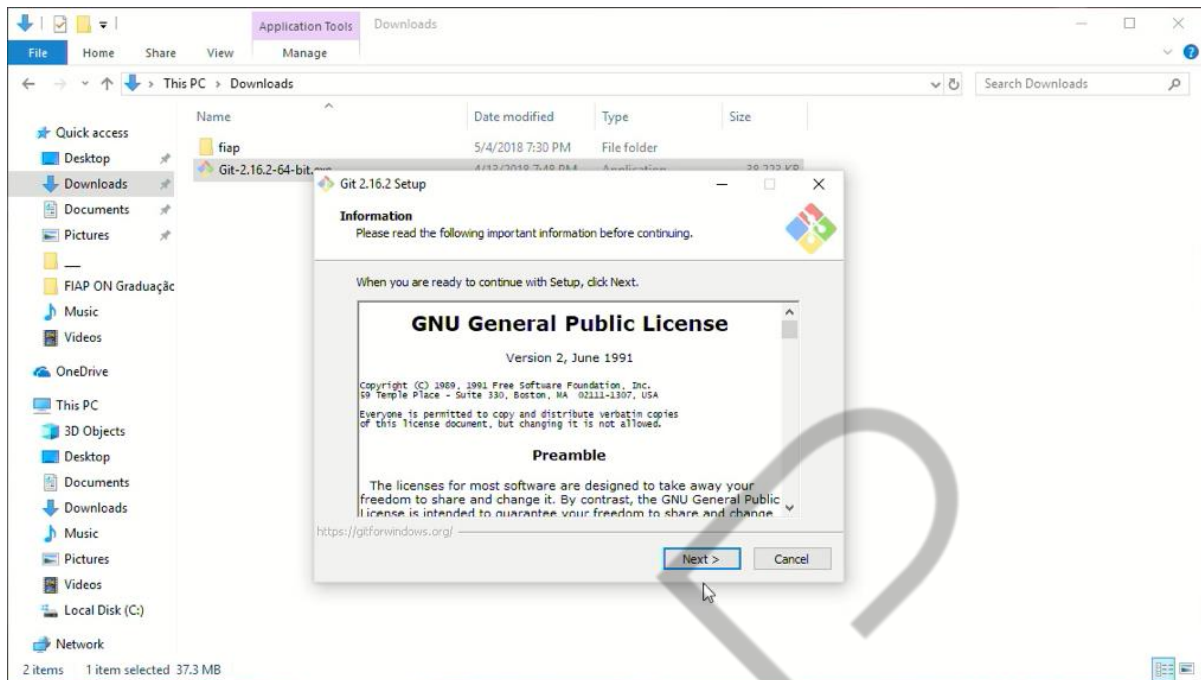


Figura 1.2 - Instalando o GIT
Fonte: Elaborado pelo autor (2018)

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, devemos primeiro criar uma pasta (pode ser no desktop mesmo) e, com o botão direito do mouse, clicar na opção Git Bash Here:

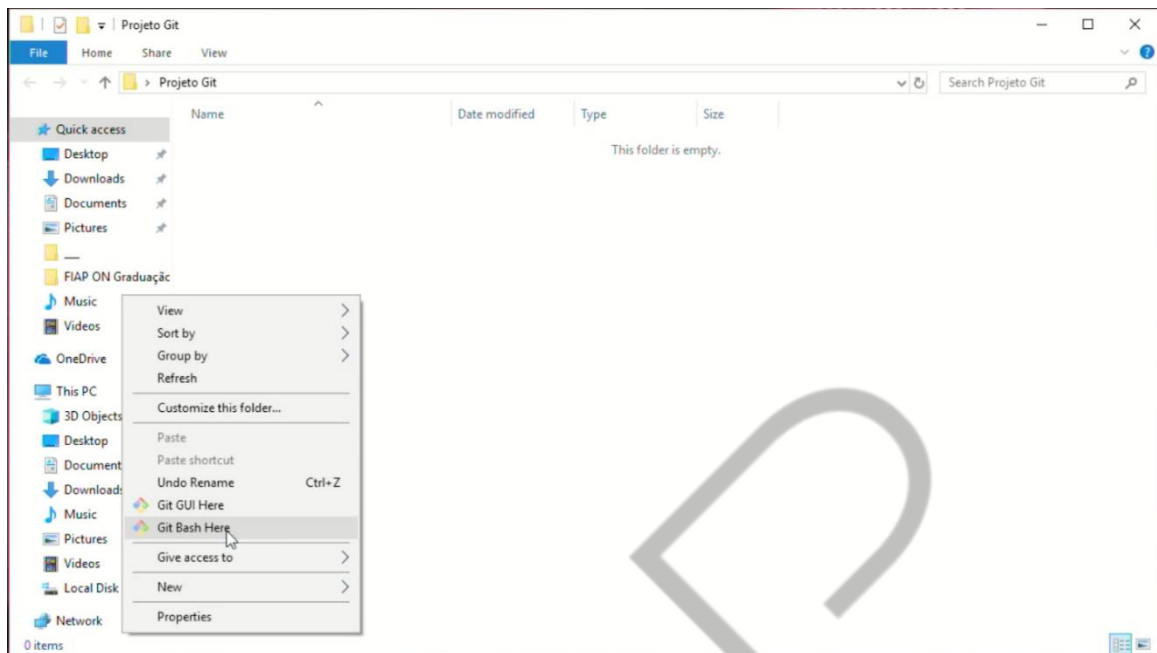


Figura 1.3 - Criando uma pasta
Fonte: Elaborado pelo autor (2018)

No exemplo da figura “Criando uma pasta”, nomeie a pasta de: Projeto Git. Após nomeá-la, clique em Git Bash Here e o nosso terminal se abrirá para você digitar o primeiro comando: **git init**.

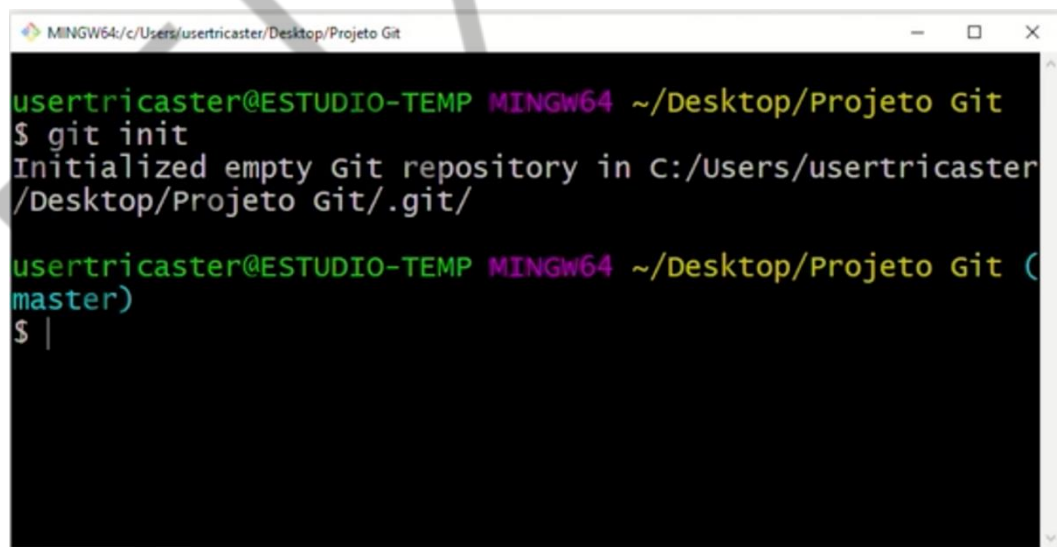


Figura 1.4 - Comando git init no terminal
Fonte: Elaborado pelo autor (2018)

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

Para isso, basta digitar no terminal os comandos:


```
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.

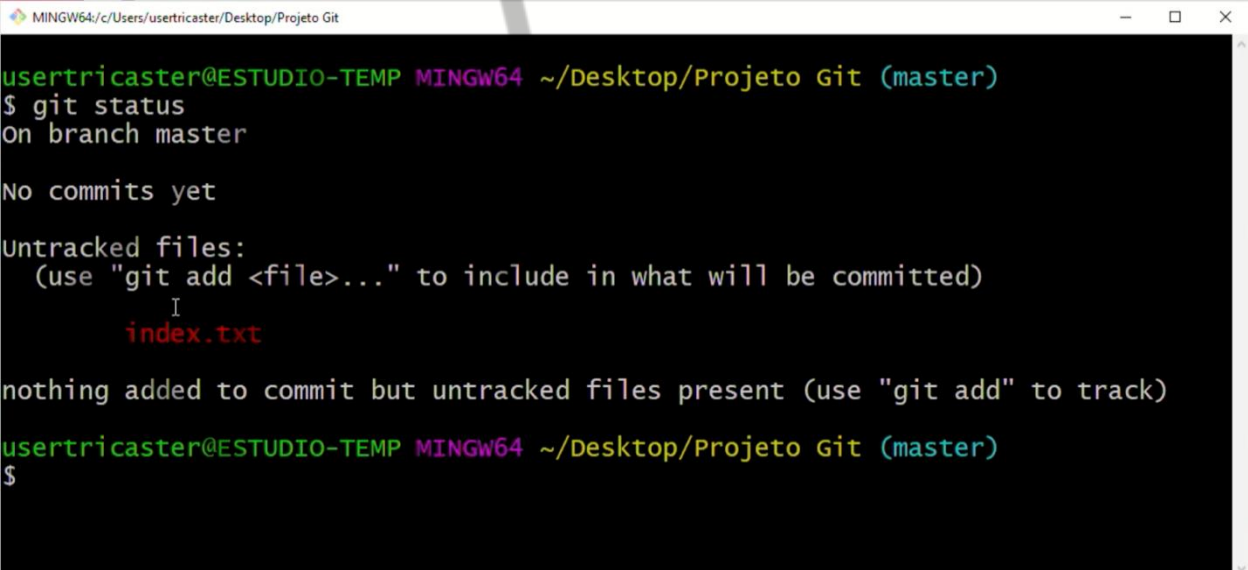
Para verificar se o nome e e-mail foram setados corretamente, basta entrar com os comandos:

```
git config user.name
```

```
git config user.email
```

1.1.3 Estados dos arquivos e commits

Dentro da sua pasta, Projeto Git, crie um arquivo index.txt; e, em seu terminal, execute o comando “git status”. Esse 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 returns to '\$ usertricaster@ESTUDIO-TEMP MINGW64 ~/Desktop/Projeto Git (master)'.

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

No commits yet

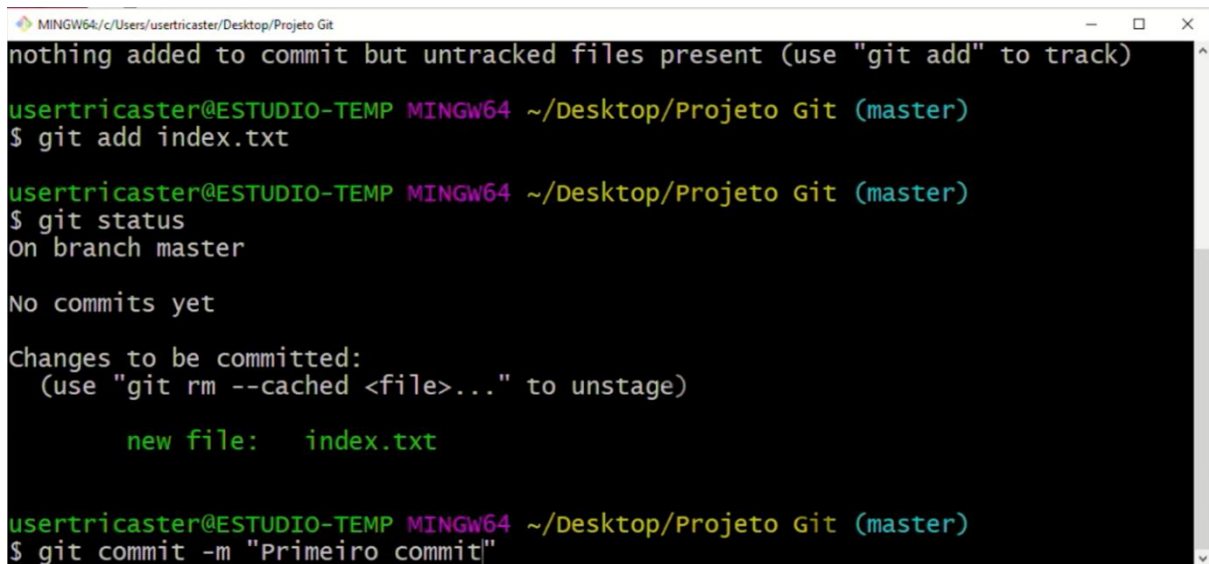
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        index.txt

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

Figura 1.5 - Comando git status
Fonte: Elaborado pelo autor (2018)

Neste caso, a mensagem retornada é: untracked; ou seja, um arquivo desconhecido para o Git, pois ele entende que existe um arquivo, mas não identifica nenhum estado. Precisamos informar para o Git que o arquivo deve ser versionado e, para isso, executaremos, então, dois comandos.

O primeiro comando é seguido do nome do arquivo: `git add index.txt`. O segundo comando é responsável pela gravação no repositório: `git commit -m "Primeiro commit"`

A terminal window titled 'MINGW64/c/Users/usertricaster/Desktop/Projeto Git' with standard window controls. The terminal shows the following sequence of commands and output:

```
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)

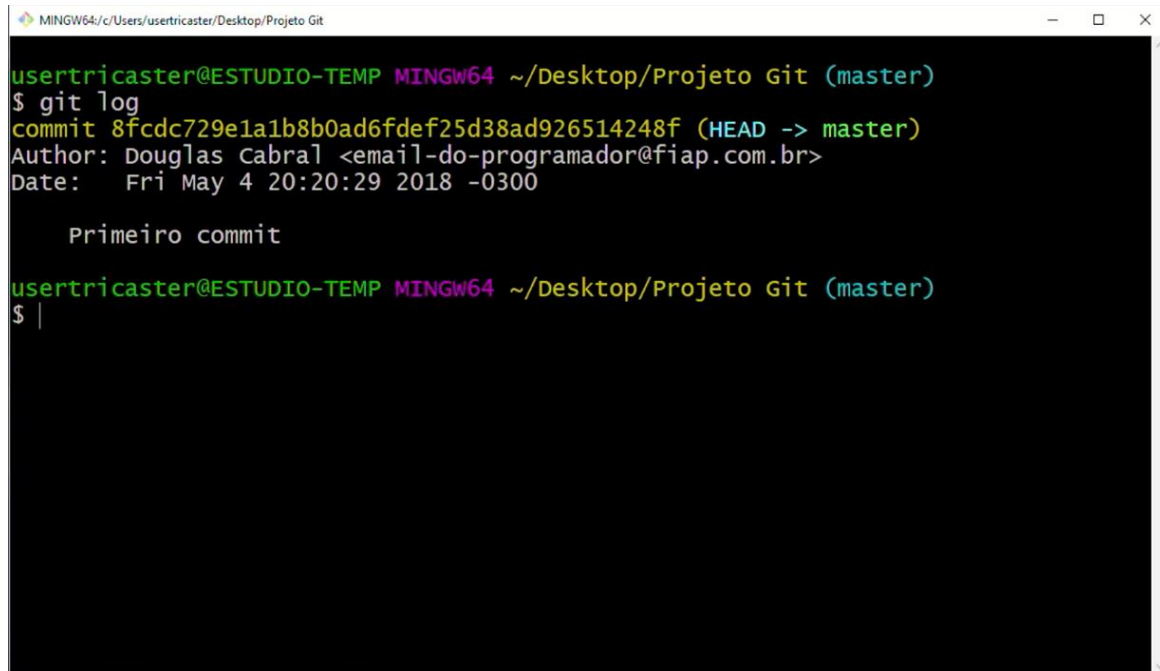
        new file:   index.txt
usertricaster@ESTUDIO-TEMP MINGW64 ~/Desktop/Projeto Git (master)
$ git commit -m "Primeiro commit"
```

Figura 1.6 - Commitando um arquivo
Fonte: Elaborado pelo autor (2018)

Pronto! Agora temos o primeiro versionamento do nosso arquivo.

1.1.4 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`.



```
MINGW64/c:/Users/usertricaster/Desktop/Projeto Git
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 1.7 - Comando git log
Fonte: Elaborado pelo autor (2018)

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

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

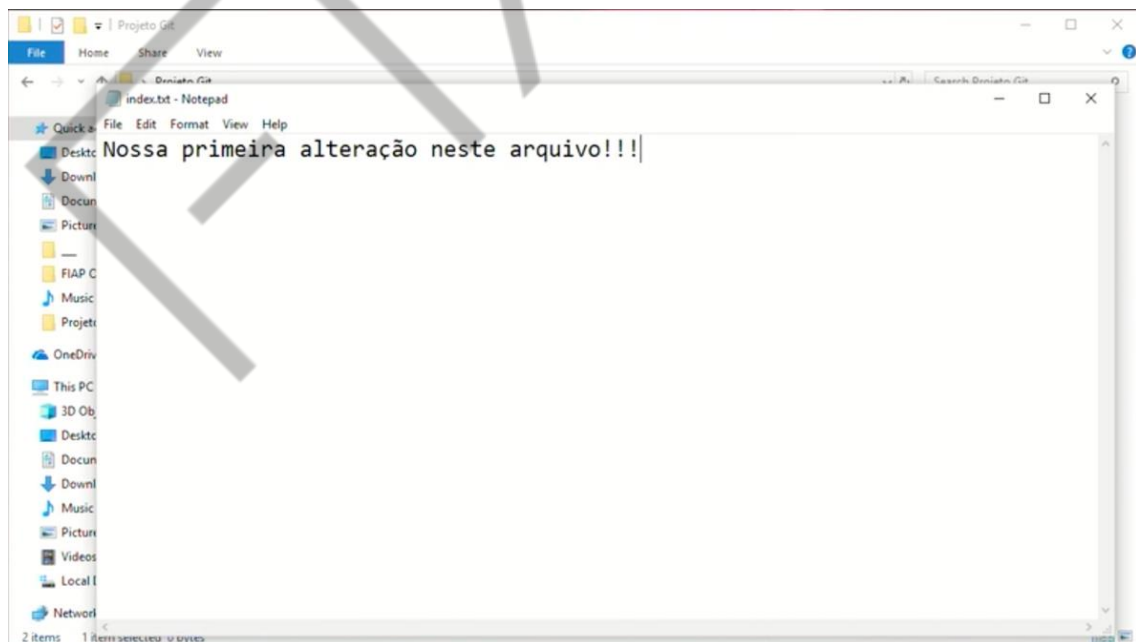
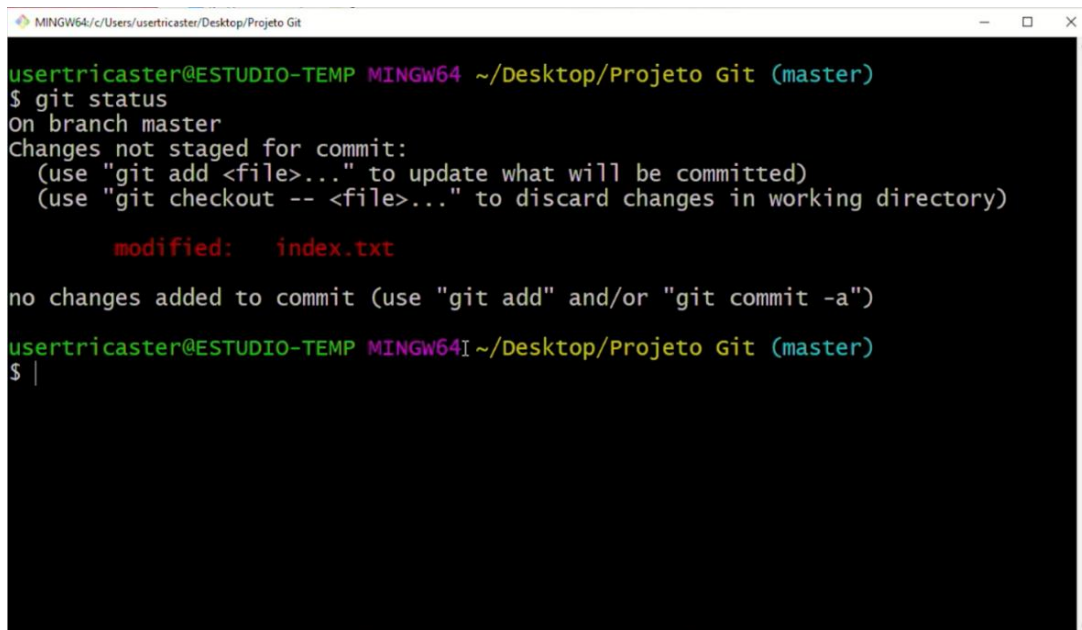


Figura 1.8 - Alteração no arquivo
Fonte: Elaborado pelo autor (2018)

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



```
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)

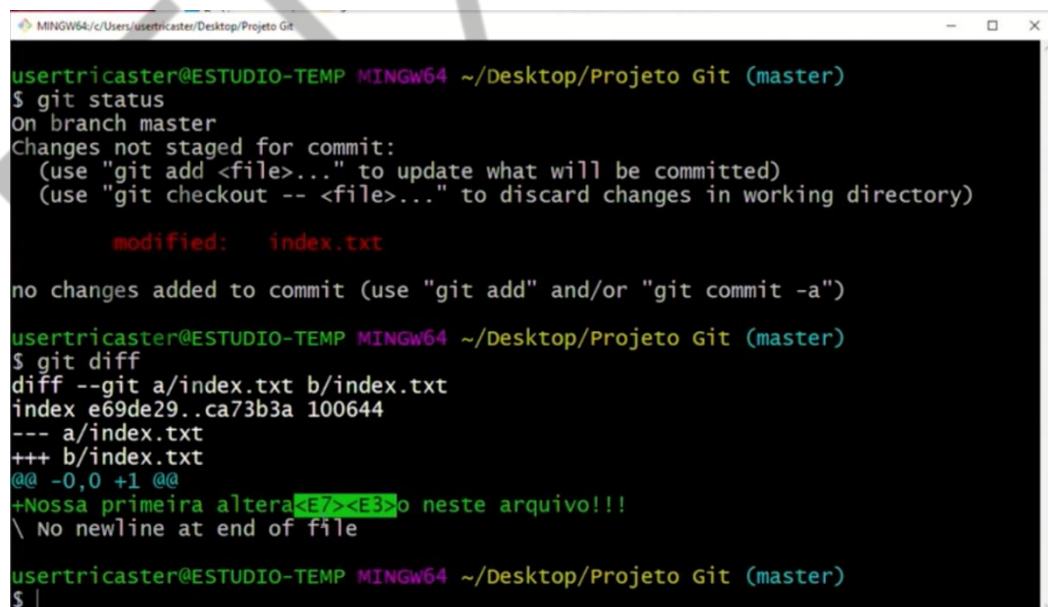
        modified:   index.txt

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

Figura 1.9 - Comando git status
Fonte: Elaborado pelo autor (2018)

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

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



```
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)

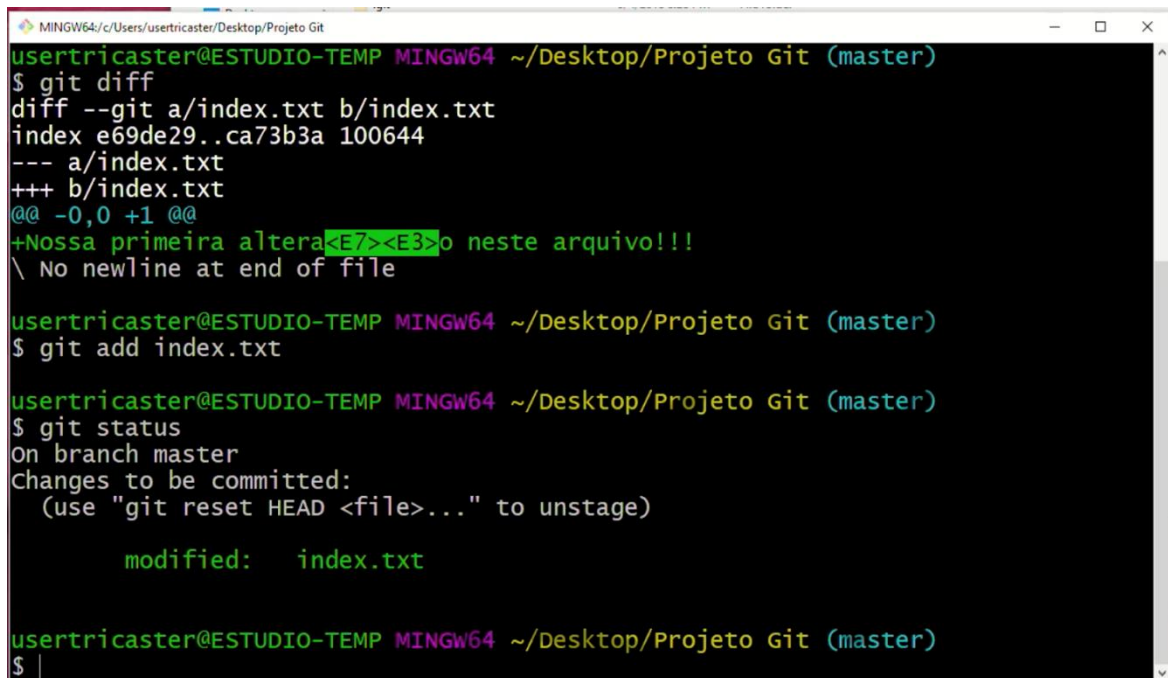
        modified:   index.txt

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 1.10 - Comando 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 o arquivo, para isso, utilize o comando que você já aprendeu: `git add index.html`.



```
MINGW64/c:/Users/usertricaster/Desktop/Projeto Git
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 1.11 - Comando `git add index`
Fonte: Elaborado pelo autor (2018)

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

1.1.5 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.

1.2 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.

1.2.1 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**

1.2.2 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 essas mudanças e alterações? Assista ao vídeo: **Merge** e aprenda a mesclar nossas alterações.

1.2.3 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 esses erros no vídeo: **Resolução de conflitos**.

1.2.4 Ignorando arquivos do repositório

E quando surgirem situações nas quais temos arquivos em nosso repositório, mas não queremos que eles sejam rastreados? Existem diversos motivos para isso acontecer, e para não atrapalhar a sua produtividade, aprenda a ignorá-los no vídeo: **Ignorando arquivos do repositório**.

1.3 Conhecendo o GitHub

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.

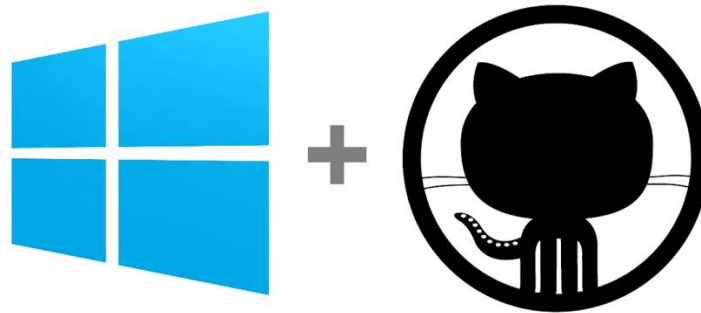


Figura 1.12 - Microsoft compra GitHub
Fonte: MSDN Blogs

Aprenda no vídeo: **Conhecendo o GitHub** a criar sua conta no GitHub e comece a versionar o seu código através dessa plataforma.

1.3.1 Trabalhando com repositórios remotos

Você já aprendeu a criar repositórios locais, mas com o 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

1.3.2 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 esse comando?

Então assista ao vídeo **Git pull**

1.3.3 Git Push

O comando git push é o inverso do git pull. É por meio dele que você envia seus arquivos atualizados para o seu repositório remoto, o GitHub.

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

1.4 Boas práticas

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

REFERÊNCIAS

AQUILES, A. **Controlando versões com Git e GitHub**. São Paulo: Casa do Código, 2016.

GIT. Disponível em: <<https://git-scm.com/>>. Acesso em: 17 set. 2020.

EMANIP