

# Git e Versionamento de Código



## Sumário

<b>1. Introdução e Definição de Git.....</b>	<b>3</b>
1.1 O que é Git?	
1.2 Qual a importância do Git?	
1.3 O que é Versionamento de Código?	
1.4 O que é GitHub?	
- Diferença entre Git e GitHub	
- O que são repositórios remotos?	
<b>2. Criando uma conta no GitHub.....</b>	<b>5</b>
2.1 Instalando o Git (Windows, macOS, Linux)	
<b>3. Configuração inicial do Git.....</b>	<b>8</b>
3.1 Configurando o usuário global	
3.2 Criando um repositório	
3.3 Clonando um repositório	
<b>4. Fluxo de Trabalho no Git.....</b>	<b>10</b>
4.1 Conceitos Fundamentais	
<b>5. Comandos Básicos.....</b>	<b>11</b>
5.1 Verificar o status do repositório (git status)	
5.2 Adicionar arquivos ao stage (git add)	
5.3 Confirmar alterações (git commit)	
5.4 Enviar e Baixar as alterações	
<b>6. Boas Práticas de Mensagens de Commit.....</b>	<b>14</b>
<b>7. Visualizar o histórico de commits com git log.....</b>	<b>15</b>
7.1 Ver resumo por autor com git shortlog	
<b>8. Branches: Trabalhando com Ramificações.....</b>	<b>17</b>
8.1 O que é uma Branch	
8.2 Por que usar Branches	
8.3 Como usar Branches	
8.4 O que é um Pull Request	
<b>9. Conclusão e Referências.....</b>	<b>19</b>



## 1. Introdução e Definição de Git



### 1.1 O que é Git?

O Git é um sistema de controle de versões distribuídas, o qual registra quaisquer alterações feitas em cima de um código, armazenando essas informações e, permitindo que, caso seja necessário, um programador possa regredir a versões anteriores de uma aplicação de modo simples e rápido. Se algo não funcionar como esperado no desenvolvimento, com o Git é simples reverter para uma versão anterior do código.

### 1.2 Qual a importância do Git?

Desempenha um papel fundamental no desenvolvimento de software, permitindo que desenvolvedores rastreiem, gerenciem e colaborem em projetos de forma eficiente.

### 1.3 O que é Versionamento de Código?

O versionamento de código é usado durante a construção de uma aplicação tecnológica, onde novas correções e implementações são criadas diariamente. Por isso é necessário trabalhar com o auxílio de ferramentas que ajudam no registro dessas modificações, indicando quem fez a mudança, quando e o que foi alterado.

- Qual a importância do Versionamento de Código?

Com o uso de sistemas de controle de versão, os desenvolvedores podem garantir a qualidade, segurança e estabilidade do código-fonte, facilitando a evolução dos projetos e o trabalho colaborativo.

Principais situações em que o versionamento de código é benéfico:

- Colaboração em equipe - Permite que vários desenvolvedores trabalhem no mesmo projeto simultaneamente, sem conflitos, graças à gestão de branches e mesclagem de código.

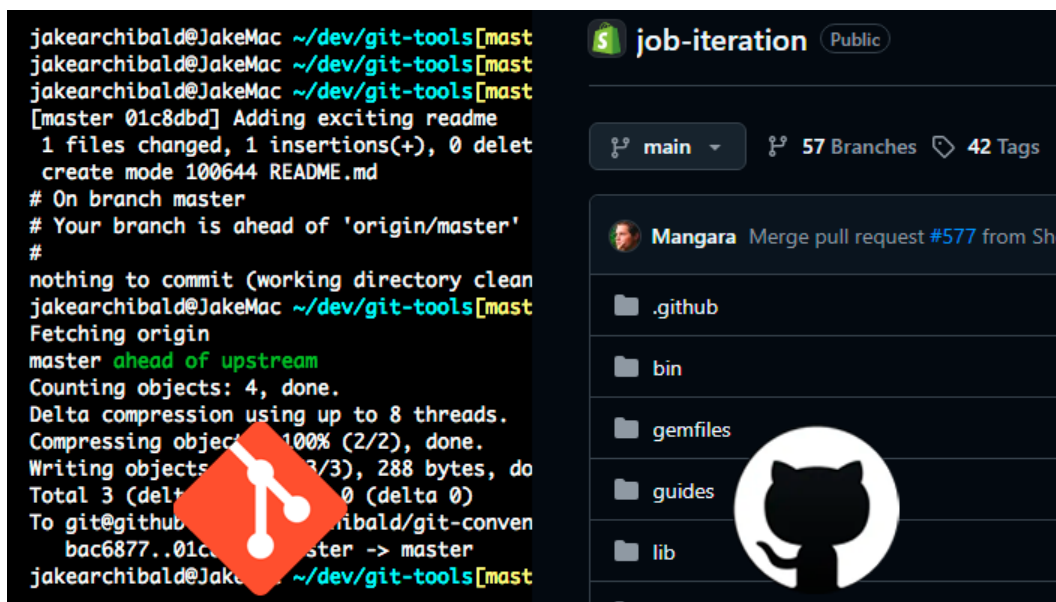
- Rastreabilidade e histórico - Registra todas as alterações no código, incluindo quem as fez e quando, facilitando a auditoria e o entendimento da evolução do projeto.
- Recuperação de versões - Possibilita reverter o código a estados anteriores em caso de bugs ou falhas, ou para explorar novas funcionalidades.

#### 1.4 O que é Github?

O Github é uma espécie de rede social voltada a profissionais de TI, cuja tecnologia de suporte é o GIT. Além disso, é uma plataforma totalmente online onde se pode criar repositórios e hospedar neles seus projetos, colaborar com softwares open source, seguir outros(as) programadores(as) e interagir com códigos de terceiros. Ele também armazena todos estes dados em uma nuvem, podendo ser acessados de onde estiver, basta logar-se no site em qualquer navegador.

- Qual a diferença de Git para Github?

O Git é uma ferramenta de controle de versão local, enquanto o GitHub é uma plataforma online que permite hospedar projetos Git e colaborar com outras pessoas.



- O que são repositórios remotos?

Repositórios remotos, no contexto do Git, são versões de um projeto (código, documentos etc.) que são armazenadas em um servidor remoto, como o GitHub ou Bitbucket, em vez de estarem apenas no seu computador local. São essenciais para a colaboração em projetos, o compartilhamento de código e o rastreamento de mudanças.

Exemplos de Repositórios Remotos:

- **GitHub** - Uma plataforma popular para hospedar repositórios Git, permitindo que desenvolvedores compartilhem e colaborem em projetos de código aberto e privados.
- **Bitbucket** - Outra plataforma de hospedagem de repositórios Git, especialmente forte na gestão de projetos de software e na integração com ferramentas de acompanhamento de projetos, como Jira.
- **GitLab** - Uma plataforma que oferece hospedagem de repositórios Git, juntamente com um sistema de acompanhamento de projetos, CI/CD e outras ferramentas de desenvolvimento de software.

Portanto entendidas as definições e principais diferenças, aprenderemos a como instalar e utilizar os programas na máquina.

## 2. Criando uma conta no Github



A fim de utilizar o GitHub, é necessário criar uma conta no site oficial do GitHub para depois instalar o Github Desktop:

1. Acesse o site do [GitHub](https://github.com).
2. Digite seu e-mail em "Enter your email" e depois clique em "Sign up for Github" (Inscreva-se no Github).
3. Preencha as informações solicitadas, como nome de usuário, endereço de e-mail e senha e clique em "Continue".

### Sign up to GitHub

Email\*

exemplo@gmail.com

Password\*

\*\*\*\*\*

Password should be at least 15 characters OR at least 8 characters including a number and a lowercase letter.

Username\*

nome-de-usuario

Username may only contain alphanumeric characters or single hyphens, and cannot begin or end with a hyphen.

Your Country/Region\*

Brazil

For compliance reasons, we're required to collect country information to send you occasional updates and announcements.

Email preferences

☐ Receive occasional product updates and announcements

Continue >

4. Siga as instruções para verificar sua conta.

Seguindo essas instruções, instalaremos o **GitHub Desktop**:

5. Entre no site do [Github Desktop](#).
6. Para Windows, clique em "Download for Windows (64bit)"

## Download GitHub Desktop

Focus on what matters instead of fighting with Git. Whether you're new to Git or a seasoned user, GitHub Desktop simplifies your development workflow.

Download for Windows (64bit)

7. Para macOS, baixe o arquivo ZIP da página de download do GitHub Desktop e, em seguida, descompacte o arquivo e mova o aplicativo para a pasta Aplicativos
8. Depois de instalar, abra o GitHub Desktop.
9. Na tela inicial, clique em "Sign in to GitHub.com" que irá abrir o site para vincular sua conta.

Após seguir esse conjunto de passos, é possível implementar o sistema de versionamento de código com o Git.

## 2.1 Instalando Git

Para Windows:

1. Acesse o site: <https://git-scm.com/downloads/win>
2. Clique em "Click here to download".



3. Abra o arquivo baixado para instalar.

Para macOS:

- 1.1 Acesse o site oficial: <https://brew.sh/>
- 1.2 Copie o comando de instalação exibido na página.
- 1.3 Pressione CMD + Espaço para abrir a busca do Spotlight e digite "Terminal", depois pressione Enter.
- 1.4 No Terminal, cole o comando copiado com CMD + V e pressione Return.
- 1.5 Digite sua senha de administrador (a digitação não será exibida) e pressione Return novamente.
- 1.6 Siga as instruções do terminal e pressione Return sempre que solicitado.
- 1.7 Se receber uma mensagem de que o diretório bin não está no PATH, siga as instruções exibidas em "Next steps" para corrigir isso.
- 1.8 Após a instalação, feche o Terminal.
- 1.9 Abra novamente o Terminal e execute o comando ***brew install git***.

E Para Linux:

1.1 Utilize o gerenciador de pacotes do seu sistema (ex: apt, yum, etc.) para instalar o Git. Por exemplo, em sistemas Debian/Ubuntu.

1.2 Use o comando ***sudo apt-get install git***.

### 3. Configuração inicial do Git

Com Git em seu sistema, é crucial personalizar o ambiente Git. Isso será feito apenas uma vez por computador e o efeito se manterá após atualizações. Além disso, é possível mudá-las a qualquer momento, executando esses comandos novamente.

O Git vem com uma ferramenta chamada git config que permite visualizar e atribuir variáveis de configuração que controlam todos os aspectos de como o Git aparece e opera.

Estas variáveis podem ser armazenadas em três lugares diferentes:

- ***/etc/gitconfig*** - Válido para todos os usuários no sistema e todos os seus repositórios. Se alterar a opção *--system para git config*, ele lê e escreve neste arquivo.
- ***~/.gitconfig* ou *~/.config/git/config*** - Somente para o seu usuário. O Git Lê e escreve neste arquivo passando a opção *--global. config* no diretório Git (ou seja, *.git/config*) de qualquer repositório em uso. Cada nível sobrescreve os valores no nível anterior, ou seja, valores em *.git/config* prevalecem sobre */etc/gitconfig*.

No Windows, Git busca pelo arquivo .gitconfig no diretório \$HOME (C:\Users\%USER para a maioria) e por */etc/gitconfig*, mesmo sendo relativo à raiz do sistema, a qual é onde o Git se encontra instalado no sistema.

#### 3.1 Configurando o usuário global

Um usuário global no Git é uma configuração que influencia todas as operações do Git (inclusive repositórios locais), em que se define o nome do autor e seu email para os commits (operação de enviar o código atualizado para o repositório):

1. Abra o Terminal (no macOS e Linux) ou o Git Bash (no Windows, que é instalado junto com o Git).
2. Configure seu nome de usuário com o seguinte comando, substituindo "Seu Nome" pelo seu nome real: ***Bash (git config --global user.name "Seu Nome")***



3. Configure seu endereço de e-mail com o comando a seguir, substituindo "seu.email@exemplo.com" pelo seu endereço de e-mail usado no GitHub: ***Bash (git config --global user.email seu.email@exemplo.com).***
4. Verifique se a sua configuração está correta utilizando: ***Bash (git config --global user.name) – (git config --global user.email).*** Essa linha de código deve exibir o nome de usuário e o e-mail que foi configurado anteriormente.

### 3.2 Criando um repositório

Para criar um repositório, utiliza-se o comando único ***git init***, pois cria um subdiretório ***.git*** no diretório de trabalho atual, e uma ramificação principal.

### 3.3 Clonando um repositório

É possível utilizar o Sourcetree, o Git pela linha de comando, ou qualquer cliente que desejar para clonar seu repositório Git.

Siga os passos abaixo de como clonar seu repositório usando o Git pelo terminal:

1. No repositório, selecione o botão Clonar.
2. Copie o comando clone (no formato SSH ou HTTPS).
3. Se estiver usando o protocolo SSH, certifique-se de que a chave pública esteja no Bitbucket e carregada no sistema local para onde deseja-se clonar.
4. Em uma janela de terminal, vá para o diretório local onde você deseja clonar seu repositório.
5. Cole o comando que você copiou do Bitbucket:

5.1 ***\$git clone*** (Clonar por HTTPS): <https://username@bitbucket.org/teamsinspace/documentation-tests.git>. Uma nova janela será aberta e solicitará que você faça login em bitbucket.org e autorize a conexão.

5.2 ***\$git clone*** (Clonar via SSH): <git@bitbucket.org:teamsinspace/documentation-tests.git>.

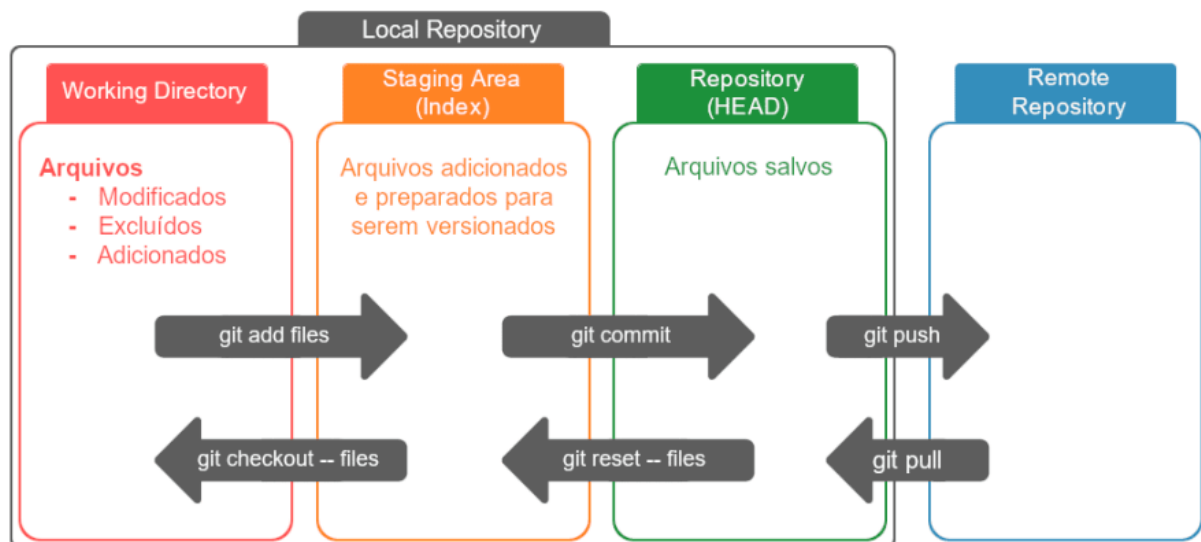
Se a clonagem for bem-sucedida, um novo subdiretório aparecerá na sua unidade local, no diretório em que o repositório foi clonado. Este diretório tem o mesmo nome do repositório do Bitbucket que foi clonado. O clone contém os arquivos e metadados que o Git precisa para manter as alterações feitas nos arquivos de origem.

## 4. Fluxo de Trabalho no Git

### 4.1 Conceitos Fundamentais

O fluxo de trabalho do Git se resume a esses três conceitos:

1. **Working Directory** - É o diretório principal onde alterações serão realizadas nos arquivos fonte dos projetos, ou seja, é a cópia do projeto e seus arquivos, que podem estar modificados, não rastreados ou prontos para serem versionados.
2. **Staging Area (Index)** - Serão preparados e selecionados arquivos para serem incluídos no próximo commit. O usuário pode controlar e selecionar quais mudanças permanecem ou não.
3. **HEAD** - Reflete a versão atual do repositório. Servindo como uma espécie de ponteiro, ele aponta para o commit mais recente da branch e atualiza quando um novo é feito, mudando para representá-lo.



## 5. Comandos Básicos

Conhecer os comandos básicos do Git garantirá um gerenciamento eficiente das versões do seu projeto. Aqui estão alguns deles:

### 5.1 Verificar o Status do Repositório

No intuito de verificar o estado do Diretório de Trabalho e Stage, utilizamos o comando *"git status"*. Ele exibe o que foi modificado, o que está pronto para ser salvo no próximo commit e o que ainda está fora do controle do Git.

```
D:\projects\repo>git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")

Changes to be committed:
  new file:   added.txt
  modified:   updated-staged.txt

Unmerged paths:
  (use "git add <file>..." to mark resolution)
    both modified:   unmerged.txt

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:   updated-unstaged.txt

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

D:\projects\repo>
```

## 5.2 Adicionar arquivos ao Stage

Para adicionar arquivos ao Stage, escreve-se *"git add nomedoarquivo"*. "Nomedoarquivo" representa que, depois do comando, é necessário inserir o nome do arquivo existente o qual será enviado, pois, dessa forma, será adicionado ao Stage.

Se quiser adicionar todas as alterações do Working Directory pro Stage, escreve-se *"git ."* com um "." (ponto) no lugar de *"add nomedoarquivo"* e são inseridas todas as mudanças, incluindo as não selecionadas. Dessa maneira, é útil, porém é recomendado usar corretamente.

```

Akash Jha@LAPTOP-LJJ1U61G MINGW64 ~/Desktop/Git/SetUp (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        file1.txt
        file2.txt

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

Akash Jha@LAPTOP-LJJ1U61G MINGW64 ~/Desktop/Git/SetUp (master)
$ git add .

Akash Jha@LAPTOP-LJJ1U61G MINGW64 ~/Desktop/Git/SetUp (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   file1.txt
        new file:   file2.txt

```

### 5.3 Confirmar alterações

Escrevendo a linha de código: *"git commit -m "mensagem do commit"*, permite que alterações sejam enviadas com um comentário para descrever claramente o estado da nova versão, nesse caso, o que mudou. Essas mudanças enviadas efetivamente ficam salvas no histórico que demonstra cada versão do seu projeto

```

Akash Jha@LAPTOP-LJJ1U61G MINGW64 ~/Desktop/Git (master)
$ git status
On branch master

No commits yet

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

Akash Jha@LAPTOP-LJJ1U61G MINGW64 ~/Desktop/Git (master)
$ git commit -m "first commit"
[master (root-commit) 5a457ea] first commit
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 file.txt

Akash Jha@LAPTOP-LJJ1U61G MINGW64 ~/Desktop/Git (master)
$ git status
On branch master
nothing to commit, working tree clean

```

## 5.4 Enviar e Baixar as alterações

Após confirmar as alterações com o comando **git commit**, é necessário sincronizar o repositório local com o repositório remoto. Para isso, utilizamos os comandos **git push** e **git pull**.

O comando **git push** é utilizado para enviar as alterações confirmadas (commits) do repositório local para o repositório remoto, garantindo que o trabalho feito fique salvo na nuvem e acessível a outros colaboradores.

```
justi@JM-DESKTOP MINGW64 ~/OneDrive/Documents/dev_code/new-test-repo (main)
$ git push origin main
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 20 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 351 bytes | 351.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/jumuir/new-test-repo.git
   fa232f0..6bfdc5e  main -> main

justi@JM-DESKTOP MINGW64 ~/OneDrive/Documents/dev_code/new-test-repo (main)
$ |
```

Já o comando **git pull** serve para baixar e integrar as alterações feitas por outras pessoas que já estejam no repositório remoto.

```
$ git remote -v
origin https://github.com/frankguitar11/Space_Shooter_Pro.git (fetch)
origin https://github.com/frankguitar11/Space_Shooter_Pro.git (push)

Frankie Four@DESKTOP-83D9DLR MINGW64 /f/Unity/Unity Repos/GameDevHQ Course/Space
Shooter Pro (master)
$ git pull origin master
fatal: couldn't find remote ref master

Frankie Four@DESKTOP-83D9DLR MINGW64 /f/Unity/Unity Repos/GameDevHQ Course/Space
Shooter Pro (master)
$ git pull origin main
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), 1.09 KiB | 185.00 KiB/s, done.
From https://github.com/frankguitar11/Space_Shooter_Pro
 * branch          main      -> FETCH_HEAD
 * [new branch]    main      -> origin/main

Frankie Four@DESKTOP-83D9DLR MINGW64 /f/Unity/Unity Repos/GameDevHQ Course/Space
Shooter Pro (master)
$
```

## 6. Boas Práticas de Mensagens de Commit

### 6.1 Explicação curta sobre commits semânticos

Assim como na programação, em que escrever um código legível e bem estruturado facilita a compreensão da leitura por outros desenvolvedores, o git é semelhante. Recomenda-se um padrão legível nos comentários de cada commit, auxiliando melhor o entendimento das versões e suas mudanças, independente se alguém estiver visualizando dentro ou fora do projeto, ou se for necessário utilizar em certo tipo de automação.

Há mais de um tipo de estrutura útil nas mensagens, sendo informadas no comando:

- ***feat***: Nova funcionalidade
- ***fix***: Correção de bug
- ***docs***: Alterações na documentação
- ***style***: Mudança na formação
- ***refactor***: Refatoração de código sem alterar comportamento
- ***test***: Realização de testes
- ***chore***: Tarefas de manutenção do projeto

Exemplo: ***git commit -m "fix: corrigir botão de login flutuando nos cantos da tela"***.

Mensagens objetivas serão sempre prioridade, uma vez que são mais informativas. Isso é uma ideia aplicável na maioria das vezes e nesse caso não é diferente.

## 7. Visualizar o histórico de commits com *"git log"*

Use ***"git log"*** para ver o histórico de commits feito no projeto. Para cada commit, é apresentado o autor, a data e hora e a mensagem escrita no momento da alteração.

Comandos úteis:

- *git log*. Mostra todos os commits com detalhes.
- *git log --oneline*. Mostra os commits em uma linha.
- *git log --graph*. Mostra um gráfico das branches.
- *git log --author="Nome"*. Mostra commits de um autor específico.

```
Lenovo@DESKTOP-LU5US00 MINGW64 /e/Git (master)
$ git log
commit 110f334a69a9fcb5320cb92e045a988aaa633f92 (HEAD -> master)
Author: Ankit Mahajan <ankitmahajan852@gmail.com>
Date:   Sun Dec 26 16:48:59 2021 +0530

    Second Commit


commit a38502e3b656b84f47b58eb1ada5490703ab269b (origin/master)
Author: Ankit Mahajan <ankitmahajan852@gmail.com>
Date:   Sat Dec 25 20:26:20 2021 +0530

    Initial Commit
```



exemplo do *git log*

```
Lenovo@DESKTOP-LU5US00 MINGW64 /e/Git (master)
$ git log --oneline
110f334 (HEAD -> master) Second Commit
a38502e (origin/master) Initial Commit
```



exemplo do *git log --oneline*

```

$ git log --graph
* commit 7fb9626a9bb7fd176064478e5f61a17bd596b217
  Merge: eb2ba0d f62114a
  Author: Rob Dolin (MSFT) <robdolin@microsoft.com>
  Date: Tue Aug 30 11:00:26 2016 -0700

    Merge pull request #133 from rhysgodfrey/update-telligent-provider

    Update Telligent Community Provider

* commit f62114a8ac380986bf8aa2c56d9189bc83785d72
  Author: Rhys Godfrey <rhys.godfrey@gmail.com>
  Date: Sun Dec 13 22:26:50 2015 +0000

    Update Telligent Community Provider

    Telligent Community hasn't been called "Community Server" since version
    4.x. The metablog URL was updated in 8.x so the suggested URL when
    picking "Community Server" no longer works.

    - Updating the naming of "Community Server" to "Telligent Community"
    - Adding new provider for "Telligent Community 8.0+" with updated URL
    pattern and supporting in all markets

```

exemplo do *git log --graph*

```

Lenovo@DESKTOP-LU5US00 MINGW64 /e/Git (master)
$ git log --author="Ankit Mahajan"
commit 110f334a69a9fcb5320cb92e045a988aaa633f92 (HEAD -> master)
Author: Ankit Mahajan <ankitmahajan852@gmail.com>
Date: Sun Dec 26 16:48:59 2021 +0530

    Second Commit

commit a38502e3b656b84f47b58eb1ada5490703ab269b (origin/master)
Author: Ankit Mahajan <ankitmahajan852@gmail.com>
Date: Sat Dec 25 20:26:20 2021 +0530

    Initial Commit

```

exemplo do *git log --author="Nome"*

## 7.1 Ver resumo por autor (*"git shortlog"*)

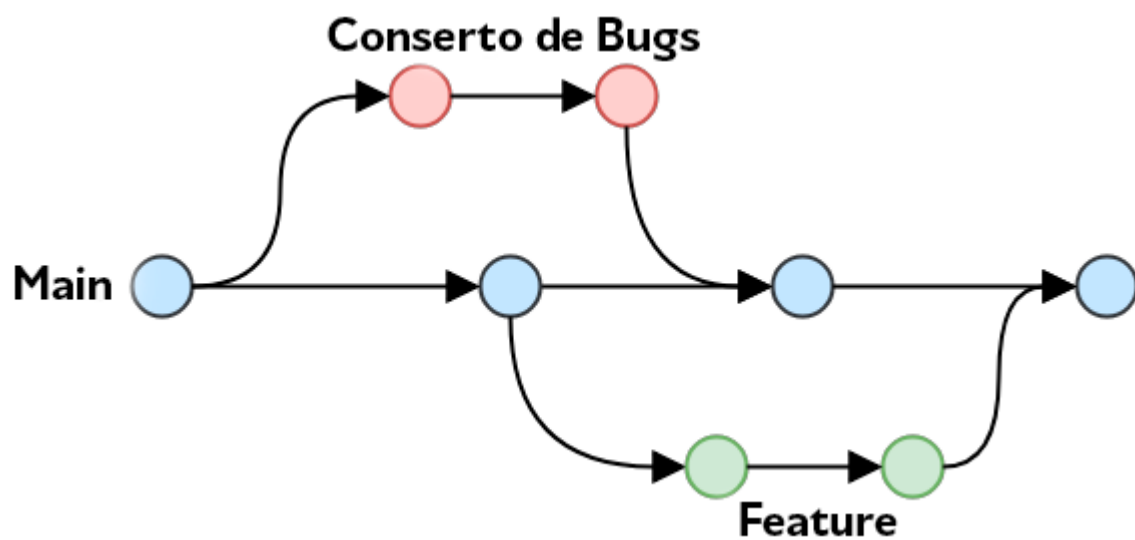
Esse comando exibe um resumo dos commits agrupados por autor, indicando quantos commits cada pessoa realizou, e podendo visualizar apenas a contagem de commits por autor usando *"git shortlog -s"*.



## 8. Branches: trabalhando com ramificações

### 8.1 O que é uma Branch?

Branch é uma cópia do seu projeto onde é possível realizar testes ou criar algo novo sem afetar o código principal.



exemplo de branches

### 8.2 Por que usar Branches?

- Trabalhar em funcionalidades separadas.
- Evitar conflitos no código.
- Facilitar o trabalho em equipe.

### 8.3 Como usar Branches?

1. **Ver branches disponíveis:** `git branch` (`git branch -r` para remotas)
2. **Criar uma nova branch:** `git branch nomedabbranch`
3. **Mudar de branch:** `git checkout nomedabbranch`
4. **Renomear uma branch:** `git branch -m novo-nome`
5. **Excluir uma branch:** `git branch -d nomedabbranch`

Observação: Você pode criar uma branch e já mudar para ela usando o comando *"git checkout -b nomedabbranch"*.

```
MINGW64 /d/exemplo (main)
$ git branch
* main
  outrabbranch

MINGW64 /d/exemplo (main)
$ git branch maisumabbranch

MINGW64 /d/exemplo (main)
$ git checkout maisumabbranch
Switched to branch 'maisumabbranch'

MINGW64 /d/exemplo (maisumabbranch)
$ git branch -m branchrenomeada

MINGW64 /d/exemplo (branchrenomeada)
$ git branch -d outrabbranch
Deleted branch outrabbranch (was 54d32ee)

MINGW64 /d/exemplo (branchrenomeada)
$ git branch
* branchrenomeada
  main

MINGW64 /d/exemplo (branchrenomeada)
$ git checkout -b outrabbranch
Switched to a new branch 'outrabbranch'

MINGW64 /d/exemplo (outrabbranch)
$ |
```

exemplos no terminal

#### 8.4 O que é Pull Request?

Pull Request (PR) é quando o usuário pede para juntar a sua branch com a principal (geralmente main). Isso é realizado em plataformas como GitHub e GitLab, permitindo que alguém revise o tal código e possibilita que as melhorias sejam discutidas, tornando mudanças efetivas apenas quando aprovadas.

## 9. Conclusão

Dominar o Git é uma das habilidades mais valiosas para qualquer desenvolvedor independente do seu nível. Nesse ebook foram transmitidos fundamentos do controle de versão, fluxo de trabalho com branches, comandos essenciais e boas práticas para manter um histórico de commits limpo e compreensível.

O Git é uma ferramenta poderosa, mas seu verdadeiro valor está na maneira como utilizá-lo para colaborar de forma eficiente e organizada com sua equipe. Pratique, explore novos comandos e aprofunde seus conhecimentos aos poucos de acordo com o que entendeu aqui. Boa parte do conteúdo que irá precisar está todo documentado e é de fácil acesso. Com o tempo, essa ferramenta se tornará parte do seu dia a dia como desenvolvedor.

### 9.1 Resumo e boas práticas

- Faça commits **pequenos e frequentes**.
- Escreva mensagens de **commit claras**.
- Crie uma **branch** por **funcionalidade** ou **tarefa**.
- Use **Pull Requests** para **revisão** de código.
- Mantenha sua **branch atualizada** com *git pull*.
- Evite muitos **commits grandes** de uma vez.

### REFERÊNCIAS:

JESUS, T. Versionamento com Git e Github - Breve resumo. Disponível em: <<https://www.dio.me/articles/versionamento-com-git-e-github-breve-resumo>>. Acesso em: 16 maio. 2025.

DESENVOLVEDOR, C. DO. Versionamento de Código: Práticas e Ferramentas para Controle de Versão. Disponível em: <<https://blog.casadodesenvolvedor.com.br/versionamento-de-codigo/>>.

GIT. Git. Disponível em: <<https://git-scm.com/>>.

GIT. Pro Git Book. Disponível em: <<https://git-scm.com/book/>>.

GIT. Git - Documentation. Disponível em: <<https://git-scm.com/doc>>.

GitHub.com Help Documentation. Disponível em: <<https://docs.github.com>>.

**ATLASSIAN. Git Tutorials and Training | Atlassian Git Tutorial. Disponível em:**  
**<<https://www.atlassian.com/git/tutorials>>.**

**git - the simple guide - no deep shit! Disponível em:**  
**<<https://rogerdudler.github.io/git-guide>>. Acesso em: 16 maio. 2025.**

