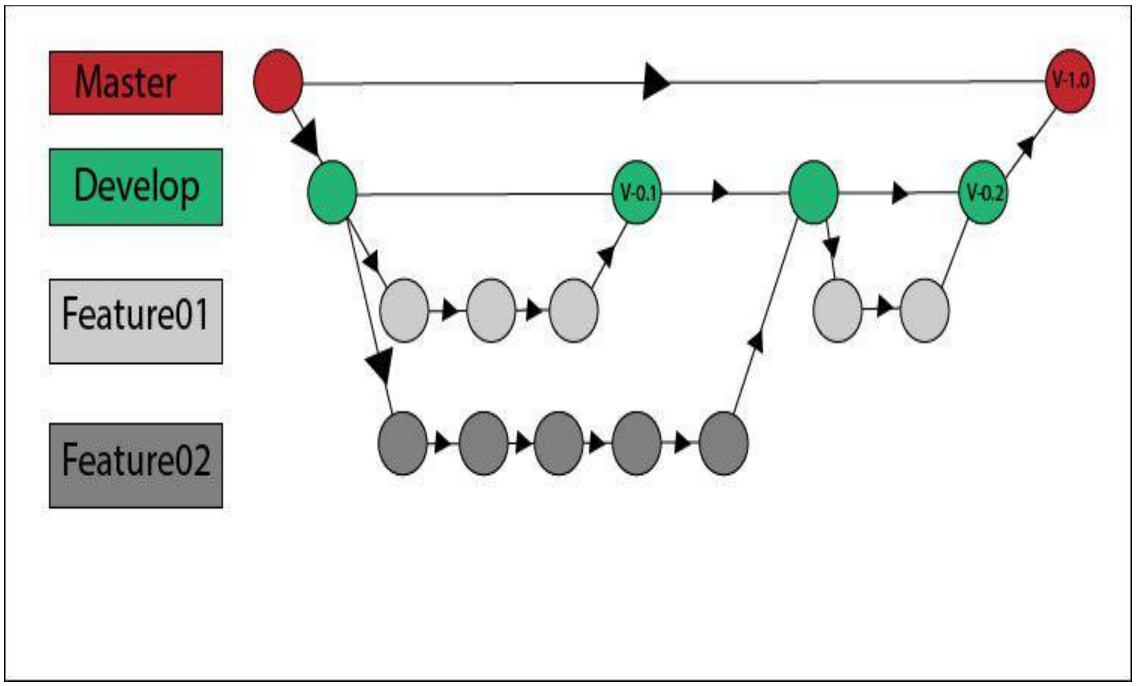

git


GitHub

Sumário

1. BRANCHS (ramos)



Os branches no Git são uma parte fundamental da metodologia de desenvolvimento, permitindo que você organize e gerencie alterações de forma eficaz, colaborando de maneira mais eficiente e mantendo o histórico de desenvolvimento organizado e rastreável.

Um "branch" no Git é como uma linha de desenvolvimento separada em um projeto de software, permitindo que diferentes partes do trabalho sejam gerenciadas de forma independente. É como se você pudesse criar cópias do seu projeto, fazer alterações em uma cópia sem afetar as outras e, quando estiver satisfeito, combinar suas alterações de volta à versão principal. Isso é útil para trabalhar em novos recursos, correções de bugs e experimentações sem causar confusão no código principal e para facilitar a colaboração entre equipes de desenvolvimento. As branches ajudam a manter o projeto organizado e o tornam mais flexível e eficiente.

1.1 Listando Branchs

O "main" ou "master" é o branch principal do Git, onde a versão mais estável do seu código é mantida, enquanto o "HEAD" é um ponteiro especial que indica o branch em que você está trabalhando atualmente, controlando onde seus commits são registrados.

Execute o comando "git branch" no gitbash. Isso listará todos os branches disponíveis no seu repositório local e destacará o branch atual com um asterisco (*).

```
usuário@DESKTOP-RVF1GLO MINGW64 ~/Desktop/curso-git (main)
$ git branch
* main
```

1.2 Manipulando Branch local

1.2.1 Criando branch

Para **criar** uma Branch, executamos o comando "git branch <nome da branch>"

```
usuário@DESKTOP-RVF1GLO MINGW64 ~/Desktop/curso-git (main)
$ git branch develop

usuário@DESKTOP-RVF1GLO MINGW64 ~/Desktop/curso-git (main)
$ git branch
  develop
* main
```

OU

```
usuário@DESKTOP-RVF1GLO MINGW64 ~/Desktop/curso-git (main)
$ git checkout -b develop
Switched to a new branch 'develop'

usuário@DESKTOP-RVF1GLO MINGW64 ~/Desktop/curso-git (develop)
$
```

1.2.2 Trocando branch

Para **trocar** de Branch, utilizamos o comando "checkout".

```
usuário@DESKTOP-RVF1GLO MINGW64 ~/Desktop/curso-git (main)
$ git checkout develop
Switched to branch 'develop'

usuário@DESKTOP-RVF1GLO MINGW64 ~/Desktop/curso-git (develop)
$ git branch
* develop
  main
```

1.2.3 Deletando branch

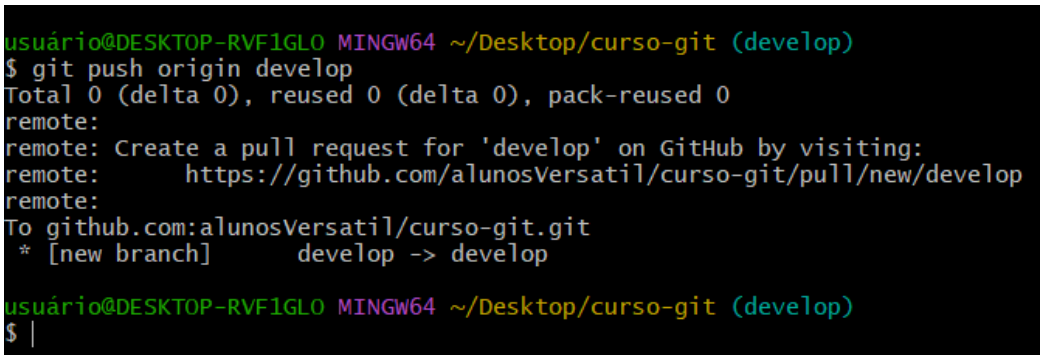
Para **deletar** uma Branch, saímos com "checkout" e utilizamos o comando "git branch -D <nome da branch>".

```
usuário@DESKTOP-RVF1GLO MINGW64 ~/Desktop/curso-git (develop)
$ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.

usuário@DESKTOP-RVF1GLO MINGW64 ~/Desktop/curso-git (main)
$ git branch -D develop
Deleted branch develop (was f05313b).

usuário@DESKTOP-RVF1GLO MINGW64 ~/Desktop/curso-git (main)
$ git branch
* main
```

1.3 Branchs no repositório remoto

1.3.1 Enviando Branch para o repositório remoto

Para um branch ficar disponível para outros, deve envia-lo ao repositório remoto.

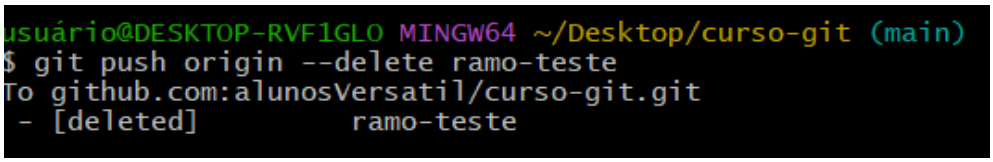


1.3.2 Baixando um Branch remoto

Se precisarmos baixar um branch remoto para edição, devemos utilizar o comando a seguir:

1.3.3 Deletando Branch remota

Para deletar uma Branch que está no repositório remoto, utilizamos o comando a seguir:



2. Logs

O comando "log" no Git é usado para visualizar o histórico de commits de um repositório. Ele exibe informações detalhadas sobre cada commit, como o autor, a data e hora da criação, a mensagem de commit e um identificador exclusivo (hash) que pode ser usado para referenciar um commit específico. O "log" é uma ferramenta poderosa para rastrear o progresso do desenvolvimento em um projeto, identificar alterações específicas e solucionar problemas.

2.1 Obtendo logs

Para visualizar o histórico de commits do seu repositório, use o seguinte comando:

```
usuário@DESKTOP-RVF1GLO MINGW64 ~/Desktop/curso-git (main)
$ git log
commit 4ced729ebee931e09eeb5ad1022631f8deaf7d22 (HEAD -> main)
Author: alunosVersatil <alunos@versatilti.com.br>
Date:   Sat Sep 16 09:24:18 2023 -0300

    testando...123

commit f05313b40c019a3da8b7c2fca08b5fdb93b66c39 (origin/main, origin/develop, or
igin/HEAD, develop)
Merge: 26efe85 c8f8451
Author: alunosVersatil <alunos@versatilti.com.br>
Date:   Mon Sep 11 17:34:45 2023 -0300

    Merge branch 'main' of github.com:alunosVersatil/curso-git

commit 26efe85a76dea5a18b0c575b6fee6442ffb1e6d6
Author: alunosVersatil <alunos@versatilti.com.br>
Date:   Mon Sep 11 17:33:59 2023 -0300

    criando aquivo
```

Isso exibirá uma lista de commits em ordem cronológica reversa (do mais recente para o mais antigo).

2.2 Manipulando logs

Você pode personalizar a saída do comando "log" usando várias opções. Alguns exemplos comuns incluem:

- Limitar o número de commits exibidos (útil para logs longos):

```
git log -n 5
```

- Exibir um resumo compacto de commits:

```
git log --oneline
```

- Exibir um gráfico simples das ramificações:

```
git log --graph
```

- Filtrar por autor ou mensagem de commit:

```
git log --author="Nome do Autor"
git log --grep="Texto na Mensagem"
```

- Exibir apenas commits de uma branch específica:

```
git log nome-da-branch
```

- Exibir o log com um formato personalizado:

```
git log --pretty=format:"%h - %an, %ar : %s"
```

3. Rótulos

Rótulos, ou "tags" em inglês, no Git são referências estáticas a commits específicos em um repositório. Eles são usados para marcar versões importantes do seu projeto, como lançamentos de software ou pontos de referência significativos. Ao contrário das branches, que representam linhas de desenvolvimento, as tags fornecem uma maneira fixa de identificar um commit em um momento específico, facilitando a recuperação de versões específicas do seu código no futuro.

3.1 Rotulando

É recomendado criar rótulos para releases de software. Para criar uma tag, execute o seguinte comando:

```
usuário@DESKTOP-RVF1GLO MINGW64 ~/Desktop/curso-git (develop)
$ git log -n 1
commit f05313b40c019a3da8b7c2fca08b5fdb93b66c39 (HEAD -> develop, origin/main, origin/develop, origin/H
EAD)
Merge: 26efe85 c8f8451
Author: alunosVersatil <alunos@versatilti.com.br>
Date:   Mon Sep 11 17:34:45 2023 -0300

    Merge branch 'main' of github.com:alunosVersatil/curso-git

usuário@DESKTOP-RVF1GLO MINGW64 ~/Desktop/curso-git (develop)
$ git tag 1.0.0 f05313b40c019a3da8b7c2fca08b5fdb93b66c39

usuário@DESKTOP-RVF1GLO MINGW64 ~/Desktop/curso-git (develop)
$ git tag
1.0.0
```

Aqui utilizamos o "git log" para buscar o último commit, em seguida atribuímos a tag "1.0.0" para o código hash do commit. Com o comando "git tag" é possível verificar as tags existentes.

3.2 Exibindo detalhes de uma tag

Para exibir detalhes de uma tag específica, incluindo o commit associado, use o seguinte comando:

```
usuário@DESKTOP-RVF1GLO MINGW64 ~/Desktop/curso-git (develop)
$ git show 1.0.0
commit f05313b40c019a3da8b7c2fca08b5fdb93b66c39 (HEAD -> develop, tag: 1.0.0, origin/main, origin/devel
op, origin/HEAD)
Merge: 26efe85 c8f8451
Author: alunosVersatil <alunos@versatilti.com.br>
Date:   Mon Sep 11 17:34:45 2023 -0300

    Merge branch 'main' of github.com:alunosVersatil/curso-git
```

3.3 Compartilhando tags

Tags criadas localmente não são automaticamente compartilhadas com repositórios remotos. Para compartilhar tags com um repositório remoto, você pode usar o seguinte comando:

```
usuário@DESKTOP-RVF1GLO MINGW64 ~/Desktop/curso-git (develop)
$ git push origin 1.0.0
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:alunosVersatil/curso-git.git
 * [new tag]         1.0.0 -> 1.0.0

usuário@DESKTOP-RVF1GLO MINGW64 ~/Desktop/curso-git (develop)
$
```

4. Atualizando Repositórios

4.1 Comando fetch

O comando "fetch" é usado para obter as atualizações mais recentes de um repositório remoto, trazendo as informações sobre novos branches, commits e tags, sem aplicar automaticamente essas alterações ao seu diretório de trabalho local. Isso permite que você verifique o que mudou no repositório remoto antes de decidir incorporar essas alterações ao seu projeto local. O "fetch" é uma operação segura e não afeta seu trabalho local.

```
usuário@DESKTOP-RVF1GLO MINGW64 ~/Desktop/curso-git (develop)
$ git fetch
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), 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), 674 bytes | 33.00 KiB/s, done.
From github.com:alunosVersatil/curso-git
   f05313b..25a4561  develop    -> origin/develop
```

4.2 Comando pull

O comando "pull" é usado para atualizar seu diretório de trabalho local com as alterações do repositório remoto. Em essência, ele combina dois passos: primeiro, ele executa um "fetch" para trazer as atualizações do repositório remoto para o seu repositório local, e depois realiza um "merge" dessas atualizações no seu branch atual. O "pull" é uma maneira conveniente de manter seu trabalho local atualizado com o projeto principal ou com outras branchs remotas.

```
usuário@DESKTOP-RVF1GLO MINGW64 ~/Desktop/curso-git (develop)
$ git pull origin develop
From github.com:alunosVersatil/curso-git
 * branch            develop    -> FETCH_HEAD
Updating cecf904..8531cfc
Fast-forward
 develop/arquivo.txt | 1 +
 1 file changed, 1 insertion(+)

usuário@DESKTOP-RVF1GLO MINGW64 ~/Desktop/curso-git (develop)
$ git log -n 1
commit 8531cfcf45b429b4efab051d00d363015a979c4a (HEAD -> develop, origin/develop)
Author: alunosVersatil <alunos@versatilti.com.br>
Date:   Sat Sep 16 12:09:15 2023 -0300

    testando pull

usuário@DESKTOP-RVF1GLO MINGW64 ~/Desktop/curso-git (develop)
$
```

5. Merge e Rebase

5.1 Merge

O comando "merge" é uma maneira fundamental de unir diferentes linhas de desenvolvimento em um projeto Git e garantir que as alterações feitas em várias branchs sejam consolidadas em uma única versão. Certifique-se de estar na branch de destino correta antes de realizar o merge e esteja preparado para resolver conflitos, se necessário. **OBS**: foram utilizados dois bash diferentes para realização do merge.

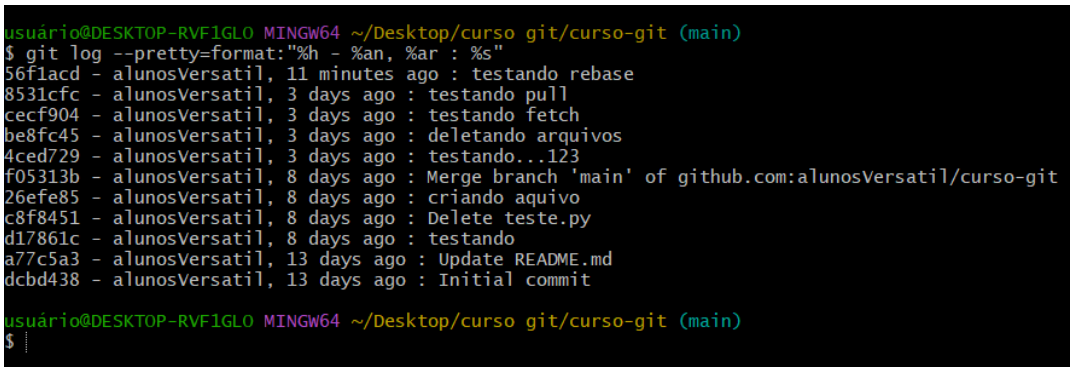
```
usuário@DESKTOP-RVF1GLO MINGW64 ~/Desktop/curso git/develop/curso-git (develop)
$ ls
README.md  develop/

usuário@DESKTOP-RVF1GLO MINGW64 ~/Desktop/curso git/develop/curso-git (develop)
$
```

```
usuário@DESKTOP-RVF1GLO MINGW64 ~/Desktop/curso git/curso-git (main)
$ ls
README.md

usuário@DESKTOP-RVF1GLO MINGW64 ~/Desktop/curso git/curso-git (main)
$ git merge origin/develop
Updating be8fc45..8531cfc
Fast-forward
 develop/arquivo.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 develop/arquivo.txt

usuário@DESKTOP-RVF1GLO MINGW64 ~/Desktop/curso git/curso-git (main)
$ ls
README.md  develop/
```

5.2 Rebase

O comando "rebase" no Git é usado para reorganizar ou "recolar" commits em uma branch, geralmente para manter um histórico de commits mais limpo e linear. Em vez de criar novos commits, como ocorre com o "merge", o "rebase" move seus commits existentes para outra branch. Usamos quando for necessário atualizar nossa branch com atualizações de uma segunda ou a main.

Verificando commits que serão enviados.



Usando git rebase para obter commits vindos de outra branch.