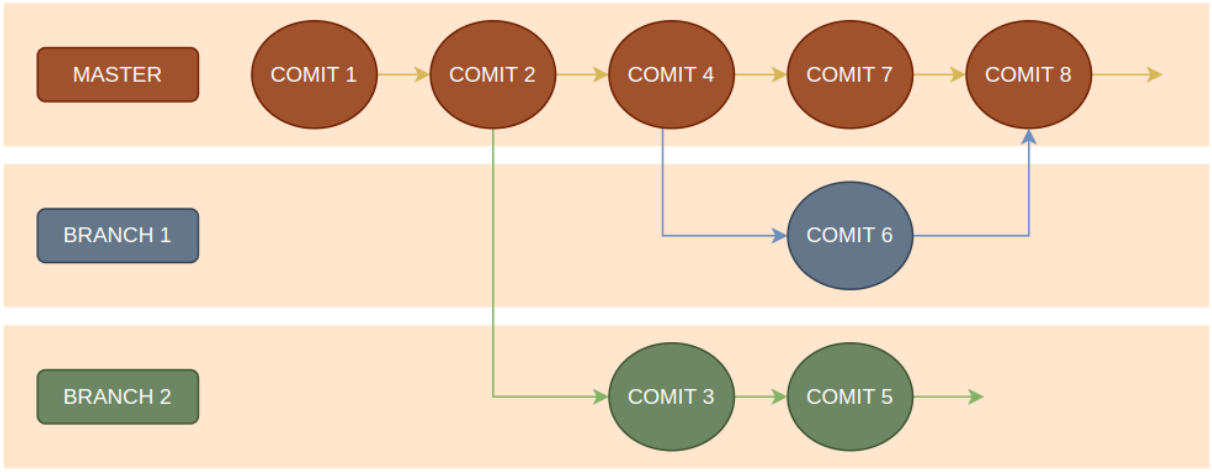
# Laboratório 1.11: Controle de Versões, Git - Parte II

## Introdução aos Branches

Digamos que você queira criar um novo recurso, mas esteja preocupado em fazer alterações no projeto principal. É aqui que entram os "*branches*" do Git. O Git trabalha como se fosse uma linha do tempo e esta linha seria o nosso branch principal, que é conhecido como *master*. Por exemplo, se você deseja adicionar uma nova página ao seu site, você pode criar uma nova ramificação apenas para essa página sem afetar a parte principal do projeto. Depois de terminar a página, você pode mesclar suas alterações no branch atual para o branch principal.




Vejam, na imagem acima, que temos três branchs, trabalhando de forma simultânea. O master e mais dois branchs, em paralelo.

Criar um branch é fazer um novo commit, contendo todos os commits anteriores a ele. Resumindo, trabalhar com branch é trabalhar em um mesmo projeto de forma assíncrona, de modo que, muitos desenvolvedores possam trabalhar em um mesmo projeto, sem um atrapalhar o outro.

## 1. Criando um Primeiro Branch

No repositório atual, temos os seguintes arquivos:

- `dev1.txt`
- `devtitans2.txt`
- `cod1.txt`
- `cod2.txt`

- cod3.txt

Suponham que este é o nosso projeto e que precisamos criar uma ramificação para o desenvolvimento de uma nova funcionalidade, ou seja, um novo branch. O comando responsável pela criação da branch é:

```
$ git checkout -b novafuncionalidade
```

```
Switched to a new branch 'novafuncionalidade'
```

O comando "checkout" diz ao Git que deve mudar para uma nova ramificação, "-b" diz ao Git que o nome do novo branch é "novafuncionalidade". É importante destacar que o branch é criado de acordo com o branch atual. Se o branch atual for o `master`, o novo branch é criado a partir do `master`. Para saber o branch atual, basta executar o comando `git branch`:

```
$ git branch
```

```
  master
* novafuncionalidade
```

Estando no branch "novafuncionalidade", criaremos um arquivo chamado funcionalidade1.txt. Após criar este arquivo, adicione-o e dê um commit.

```
$ touch funcionalidade1.txt
$ git add funcionalidade1.txt
$ git commit -m "Adicionando uma nova funcionalidade ao projeto."
```

```
 3 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 cod1.txt
 create mode 100644 cod3.txt
 create mode 100644 funcionalidade1.txt
```

Em seguida, volte para o branch master, com o seguinte comando:

```
$ git checkout master
```

```
Switched to branch 'master'
```

O comando para acessar e criar é o mesmo, mas para criação, passamos o parâmetro -b. Agora, no branch master, liste os arquivos, utilizando o comando ls:

```
$ ls
```

```
cod2.txt  dev1.txt  devtitans2.txt  esteArquivoSeraIgnorado.txt
```

Pronto, aqui podemos observar que o arquivo "`funcionalidade1.txt`" não faz parte deste branch master, mas somente do branch `funcionalidade1`. Não estamos falando, apenas, do arquivo estar ou não, fisicamente, na pasta. Se listar os logs, não verá o commit que foi feito no outro branch.

```
$ git log --pretty=oneline
```

```
3f5eb46d3afd39257ad73be899b906c669130d34 (HEAD -> master)
adicionando o git ignore
e5c1948c3f56703e5aa897cafa611ae19025b94d Comitando todos os
```

O mesmo ocorre se fizerem uma alteração em qualquer arquivo do branch master e comitar. Esta alteração não será enxergada no branch "novafuncionalidade" e nem fará parte do código. Cada branch será totalmente independente, até que seja feito um merge.

## 2. Fazendo Merge de Branchs

Neste seção, será mostrado como mesclar branchs diferentes, para que as funcionalidades se somem em um mesmo branch. No momento, temos o branch `novafuncionalidade` e o branch `master`. Agora podemos mesclar as alterações que fizemos no branch `funcionalidade` com o branch principal executando o comando `git merge novafuncionalidade`. Neste ponto, você verá todas as alterações feitas na ramificação `novafuncionalidade` refletidas na ramificação principal.

```
$ git merge novafuncionalidade
```

```
cod3.txt             | 0
funcionalidade1.txt | 0
3 files changed, 0 insertions(+), 0 deletions(-)
create mode 100644 cod1.txt
create mode 100644 cod3.txt
create mode 100644 funcionalidade1.txt
```

Depois de executar este comando teremos, tanto os commits existentes no branch novafuncionalidade, quanto os arquivos que lá foram criados. Para verificar isso, execute novamente o comando `ls`:

```
$ ls
```

```
cod1.txt  cod2.txt  cod3.txt  dev1.txt  devtitans2.txt
esteArquivoSeraIgnorado.txt  funcionalidade1.txt
```

## 3. Usando o GitHub


O GitHub é um serviço que armazena nossos projetos Git e nos auxilia a gerenciá-los. Se você tiver seu projeto hospedado no GitHub, poderá acessar e baixar esse projeto com um comando em qualquer computador ao qual tenha acesso. Em seguida, você pode fazer suas alterações e enviar a versão mais recente de volta ao GitHub. É uma excelente vitrine para expor nossos trabalhos e estudos.

### Criando uma Conta no GitHub

Neste laboratório, você precisará criar uma nova conta no GitHub. Para isso, acesse o site https://github.com/, clique no botão de "Sign In" e siga as instruções do site.

*Digite abaixo o **login** (não o e-mail) que você escolheu no GitHub:*

```
Marcosccp04
```

### Criando um Repositório Remoto

Para criar um novo repositório no GitHub, encontre a opção "New repository" sob o sinal "+" ao lado da sua foto de perfil, no canto superior direito do navegador. Depois de clicar no botão, o GitHub solicitará que você nomeie seu repositório e forneça uma breve descrição. Use o nome "**devtitans**" no nome do repositório. Quando terminar de preencher as informações, pressione o botão "Create repository".

Na próxima página, será mostrado o **link/URL** do repositório criado. Este link é usado para identificar o seu repositório e contém o seu login no GitHub e o nome do repositório criado, conforme o exemplo abaixo:

```
https://github.com/Marcosccp04/devtitans.git
```

Como já criamos um repositório localmente em nossa máquina, iremos basicamente enviá-lo para o GitHub.

### Configurando o Repositório Local

Entre no diretório do repositório local criado ("devtitans-git") em sua máquina.

Uma vez que já temos esse repositório, o próximo passo é configurá-lo para usar o servidor do GitHub e o repositório remoto que acabamos de criar. Para isso, execute o comando abaixo:

```
$ git remote add origin https://github.com/Marcosccp04/devtitans.git
```

### Primeiro Push

O comando `push` (do inglês, "empurrar") é utilizado quando temos um repositório local e queremos enviar os commits para um repositório online. Para isso, precisamos adicionar um repositório remoto.

Vamos agora enviar os arquivos locais para o repositório remoto através do comando `push`:

```
$ git push origin master
```

O comando acima basicamente diz: "empurre o meu branch master para o repositório origin". Isso significa que estamos subindo todos os arquivos e commits para o repositório online. Será pedido seu login/senha do GitHub. Entretanto, como você pode ver, um erro irá aparecer indicando que esse tipo de autenticação não é mais suportado pelo GitHub.

### Mensagem de Erro

O que acontece é que, mais recentemente, não é mais possível usar a sua senha para se autenticar na sua conta do Github para operações com o Git. É necessário fazê-lo por meio de um *token*. Para gerar um *token*,siga os seguintes passos:

- Acesse sua conta do GitHub e clique em *Settings*.
- Em seguida, vá em *Developer Settings* à esquerda.
- Clique em *Personal access tokens* e a seguir em *Generate new token*.
- Adicione uma pequena descrição (note) sobre o token a ser gerado.
- Na data de expiração, selecione "No expiration".
- Mais abaixo, na parte de **Select scopes**, selecione a primeira opção "**repo**".
- Por fim, clique em *Generate token* ao final da tela.
- Na página seguinte, copie o token gerado e salve em um arquivo texto. Você não terá mais como ver esse token no futuro. Por isso é importante salvá-lo. Caso você perca o token, é só gerar outro usando os passos anteriores.
- Agora execute o comando `push` novamente (abaixo) e, na autenticação, use seu login do GitHub e como senha use o token gerado acima.

```
$ git push origin master
```

```
Delta compression using up to 8 threads
Compressing objects: 100% (8/8), done.
Writing objects: 100% (16/16), 1.40 KiB | 477.00 KiB/s, done.
Total 16 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), done.
remote:
remote: Create a pull request for 'master' on GitHub by visiting:
remote:
https://github.com/Marcosccp04/devtitans/pull/new/master
remote:
To https://github.com/Marcosccp04/devtitans.git
```

Veja que no resultado do comando, o Git informa que foi criado um novo branch, chamado `master`, no repositório local. Em seguida, ele informa que está enviando do branch `master` local para o branch master remoto. Depois de enviar, acesse o repositório online, para verificarem os arquivos e informações usando o GitHub!

*Use o botão abaixo para submeter um screenshot da sua tela do GitHub mostrando o repositório criado. Selecione a tela inteira ou apenas a janela do navegador. Em qualquer caso, certifique-se que o repositório do GitHub esteja aparecendo em sua tela dentro de 3 segundos. Se não ficar bom da primeira vez, clique no botão novamente para substituir o screenshot.*

Tirar Screenshot [Screenshot Atual]

Observe que todas as informações de *commits* e *branchs* foram enviadas, juntamente, com o nosso `push`. Isso significa que temos uma cópia do nosso repositório local em um repositório remoto, centralizando nosso projeto.

Para que você não precise ficar digitando o token a cada operação, execute o comando abaixo para pedir que ele salve a senha.

```
$ git config --global credential.helper cache
```

Em seguida, execute o `git push` novamente para forçar uma operação com senha. Ele pedirá e salvará a senha. Por fim, execute o `git push` uma terceira vez para ver que a senha não foi mais requisitada.

### Dando Push em Outro Branch

Agora que já fizemos o push normal, enviando o branch master para o repositório remoto, iremos enviar um novo branch. Temos, em nosso exemplo, um branch chamado novafuncionalidade. O procedimento é acessar este branch e, em seguida, dar um push deste repositório, para o Github.

```
$ git checkout novafuncionalidade
```

```
Switched to branch 'novafuncionalidade'
```

Depois de acessar, execute o push:

```
$ git push origin novafuncionalidade
```

```
Total 0 (delta 0), reused 0 (delta 0)
remote:
remote: Create a pull request for 'novafuncionalidade' on GitHub
by visiting:
remote:
https://github.com/Marcosccp04/devtitans/pull/new/novafuncionalid
ade
remote:
To https://github.com/Marcosccp04/devtitans.git
 * [new branch]      novafuncionalidade -> novafuncionalidade
```

Vejam que ele informou que foi enviado de novafuncionalidade local para novafuncionalidade remoto. Quando não existe o branch, ele cria, automaticamente. Depois de terem enviado, podemos conferir no github que o branch foi realmente criado (O site exibirá o status: 2 branches).

## 4. Clonando um Repositório

Vamos supor que por engano, apagamos o repositório local, ou estamos em outro computador e precisamos desse repositório. Para exemplificar esta situação, apague a pasta devtitans-git que foi criada para esse laboratório.

```
$ cd ..
$ rm -rf devtitans-git
```

Dessa forma, o repositório não existe mais na máquina local. Para acessar novamente, é necessário baixar o repositório remotamente. "Baixar" um repositório do Git é conhecido como "clonar" um repositório.

Acessando o nosso repositório no GitHub, encontramos o botão "Code" (em verde). Através desse botão é possível copiar o link do repositório para a clonagem via https ou ssh, além de ser possível também baixar o arquivo `.zip`. A seguir, usaremos o link via HTTPS que, no seu caso, deve ser o seguinte:

```
https://github.com/Marcosccp04/devtitans.git
```

Para clonar um repositório, usamos o comando `git clone`:

```
$ git clone https://github.com/Marcosccp04/devtitans.git devtitans-
git
```

```
Cloning into 'devtitans-git'...
remote: Enumerating objects: 19, done.
remote: Counting objects: 100% (19/19), done.
remote: Compressing objects: 100% (8/8), done.
remote: Total 19 (delta 1), reused 16 (delta 1), pack-reused 0
Unpacking objects: 100% (19/19), 1.97 KiB | 336.00 KiB/s, done.
```

A última opção do comando acima indica o diretório de saída que será criado para conter o código.

Clonar um repositório é relativamente fácil, mas se executarmos o comando **git branch**, veremos que na clonagem só veio o branch *master*:

```
$ cd devtitans-git
$ git branch
```

```
* master
```

Quando executamos o comando `git branch`, visualizamos somente os branchs locais. Para vermos os branchs remotos, basta acrescentar o parâmetro `-a`:

```
$ git branch -a
```

```
  main
  remotes/origin/HEAD -> origin/main
  remotes/origin/main
  remotes/origin/master
  remotes/origin/novafuncionalidade
```

Dessa forma, podemos fazer a ligação de um branch local com um branch remoto. Primeiro, devemos criar o branch, normalmente, no repositório local, como vimos anteriormente, só que o comando deve receber um parâmetro a mais, que é o branch que desejamos ligar com o repositório remoto utilizando o comando:

```
$ git checkout -b novafuncionalidade origin/novafuncionalidade
```

```
Branch 'novafuncionalidade' set up to track remote branch
'novafuncionalidade' from 'origin'.
Switched to a new branch 'novafuncionalidade'
```

Veja que o comando retorna a mensagem informando que o branch foi criado de acordo com o branch remoto. Veja a listagem dos arquivos, para ver se o arquivo "`funcionalidade1.txt`" apareceu:

```
$ ls
```

```
cod1.txt  cod3.txt  dev1.txt  devtitans2.txt  funcionalidade1.txt
```

Para garantir que todos os arquivos do branch remoto estejam no branch local, utilizamos o comando git pull:

```
$ git pull
```

```
Already up to date.
```

## 5. Finalizando com o Push e Pull

Para finalizarmos o conceito de push e pull, faremos uma simulação para facilitar o entendimento.

Suponha que somos uma equipe de colaboradores e temos o repositório devtitans-git. Para isso, devemos ter, pelo menos, dois clones de nosso repositório para simular dois desenvolvedores. Como já temos um clone do repositório na nossa máquina local, iremos criar uma nova pasta com um novo clone:

```
$ cd ..
$ git clone https://github.com/Marcosccp04/devtitans.git clone2
```

```
Cloning into 'clone2'...
remote: Enumerating objects: 19, done.
remote: Counting objects: 100% (19/19), done.
remote: Compressing objects: 100% (8/8), done.
remote: Total 19 (delta 1), reused 16 (delta 1), pack-reused 0
Unpacking objects: 100% (19/19), 1.97 KiB | 336.00 KiB/s, done.
```

Agora que temos os dois ambientes, faremos uma alteração qualquer na pasta devtitans-git e enviaremos para o repositório remoto.

```
$ cd devtitans-git
$ git checkout master
```

```
Switched to branch 'master'
Your branch is up to date with 'origin/master'.
```

Modifique o arquivo "dev1.txt" alterando a frase "Olá mundo, eu sei fazer um commit" para "Olá mundo, aqui é uma nova alteração":

```
$ echo "Olá mundo, aqui é uma nova alteração" > dev1.txt
```

Em seguida, execute os comandos abaixo para enviar as alterações para o repositório remoto:

```
$ git add dev1.txt
$ git commit -m "Alterando o arquivo dev1.txt"
```

```
[master bbd1716] Alterando o arquivo dev1.txt
 1 file changed, 1 insertion(+), 2 deletions(-)
```

Por fim, envie as alterações para o servidor:

```
$ git push origin master
```

```
To https://github.com/Marcosccp04/devtitans.git
   423e7a0..bbd1716  master -> master
```

Em seguida, faça uma alteração em outro arquivo da outra pasta "clone2" e tente enviar para o repositório remoto também:

```
$ cd ..
$ cd clone2
$ echo "nova mensagem" > cod3.txt
$ git add cod3.txt
$ git commit -m "alterando o arquivo cod3.txt em clone2"
```

```
 1 file changed, 1 insertion(+)
 create mode 100644 cod3.txt
```

Envie as alterações para o servidor e **observe o erro** que irá aparecer:

```
$ git push origin master
```

```
 ! [rejected]        master -> master (fetch first)
error: failed to push some refs to
'https://github.com/Marcosccp04/devtitans.git'
hint: Updates were rejected because the remote contains work that
you do
hint: not have locally. This is usually caused by another
repository pushing
hint: to the same ref. You may want to first integrate the remote
```

Observe que o primeiro push foi feito com sucesso, mas o segundo no outro clone foi recusado. Isso acontece pois temos que primeiro atualizar o nosso repositório local com as últimas alterações feitas no repositório remoto utilizando o comando pull. Sempre que tiverem alguma alteração no repositório remoto, devemos atualizar, para depois subirmos

qualquer outra alteração local. Para fazerem a sincronização, devemos fazer um pull, que é o inverso do comando push:

```
$ git pull origin master
```

```
Unpacking objects: 100% (3/3), 298 bytes | 298.00 KiB/s, done.
From https://github.com/Marcosccp04/devtitans
 * branch            master     -> FETCH_HEAD
   423e7a0..bbd1716  master     -> origin/master
fatal: refusing to merge unrelated histories
```

Dessa forma, com o repositório local devidamente atualizado com a versão mais recente do repositório remoto, podemos enviar nossas alterações no arquivo cod3.txt utilizando o comando push:

```
$ git push origin master
```

```
Delta compression using up to 2 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (5/5), 656 bytes | 656.00 KiB/s, done.
Total 5 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), done.
To https://github.com/Marcosccp04/devtitans.git
   baa628c..7322141  master -> master
error: update_ref failed for ref 'refs/remotes/origin/master':
cannot lock ref 'refs/remotes/origin/master': Unable to create
'/home/clone2/.git/refs/remotes/origin/master.lock': Permission
```

Crie crie um novo branch no diretório clone

```
$ git checkout -b novobranch
```

```
Switched to a new branch 'novobranch'
```

Crie agora um novo arquivo e faça um commit dele:

```
$ touch arquivonovo.html
$ git add arquivonovo.html
$ git commit -m "Novo arquivo em novo branch"
```

```
[novobranch 3bd8eda] Novo arquivo em novo branch
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 arquivonovo.html
```

Em seguida, envie para o repositório remoto:

```
$ git push origin novobranch
```

```
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done
```

Em seguida, entre na pasta devtitans-git e faça um pull para pegar este novo arquivo com um novo branch:

```
$ cd ..
$ cd devtitans-git
$ git pull
```

```
remote: Enumerating objects: 7, done.
remote: Counting objects: 100% (7/7), done
```

A sequência de comandos acima foi executada para exemplificar o processo de criação de um arquivo, em um novo branch. Primeiro, criamos o branch, depois o arquivo, adicionamos, comitamos e, por último, subimos para o repositório remoto, criando um novo branch.

Agora, dentro do diretório "devtitans-git" e observe que não existe o arquivo arquivonovo.html, nem o novo branch.

```
$ ls
```

```
cod1.txt  cod3.txt  dev1.txt  devtitans2.txt  funcionalidade1.txt
```

Veja que o novo branch também não existe:

```
$ git branch
```

```
main
* master
```

Como já falamos, quando queremos sincronizar com o repositório remoto, temos que rodar o comando pull. Portanto, execute os comandos abaixo:

```
$ git pull
```

Veja agora a lista de branch, incluindo as remotas:

```
$ git branch -a
```

```
main
* master
```

Para usar o novo branch, criamos o branch local, com o mesmo nome e linkamos com o branch online:

```
$ git checkout -b novobranch origin/novobranch
```

```
Branch 'novobranch' set up to track remote branch 'novobranch'
from 'origin'
```

Desta forma temos tudo sincronizado, tanto repositório, quanto branchs.

*You're the best! Around! Nothing's gonna ever keep you down!* — Joe Esposito

π