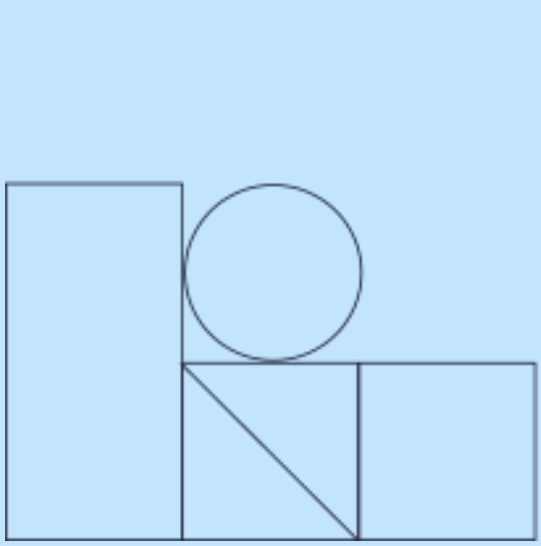
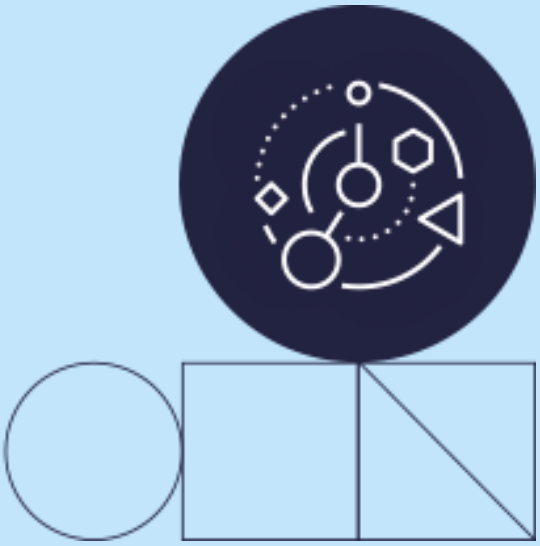
IBM SkillsBuild

IBM

Git y Github

Branch, Merge e clonflitos

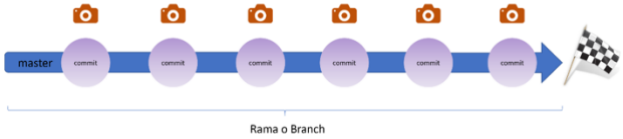




Índice

Branch

As filiais ou *Branch* são linhas do tempo que serão formadas por todos os *"commits"* que fizemos ao longo da evolução de nosso projeto.



A ramificação é uma característica disponível na maioria dos sistemas de controle de versões modernas.

A criação de filiais em outros sistemas de controle de versões pode ser demorada e ocupar muito espaço de armazenamento.

Em Git, as filiais fazem parte do processo de desenvolvimento do dia-a-dia. Os galhos de Git são um ponteiro eficiente para tirar instantâneos de nossas mudanças. Quando queremos adicionar um novo recurso ou consertar um bug, independentemente de seu tamanho, geramos uma nova filial para realizar essas mudanças. Isto torna mais difícil a fusão de códigos instáveis na base de códigos principal, e nos dá a oportunidade de limpar nossa história futura antes de fundi-la no ramo principal.

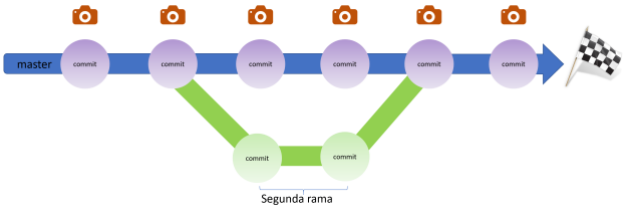
O ramo principal de um projeto é o ramo *master*. Por padrão, é o ramo no qual você começa a trabalhar.

Até agora, temos trabalhado neste ramo master.

Mas *GitHub* nos dá a possibilidade de trabalhar em vários ramos simultaneamente. Isto é muito útil para que vários desenvolvedores trabalhem simultaneamente sem pisar nos dedos dos pés uns dos outros.

Cada programador criará uma filial a partir da filial principal, trabalhará nela fazendo as mudanças ou implementações de que necessita e, quando terminar, fundirá sua filial com a filial principal.

Portanto, no final, o ramo principal refletirá todas as implementações de toda a equipe de programadores.



Cada programador será capaz de criar uma filial (uma "cópia") do projeto. Assim, podemos trabalhar tanto no original quanto na cópia.

O ideal é trabalhar na cópia e quando tivermos certeza de que nosso trabalho é perfeito e que não há insetos, o "injetamos" no ramo mestre.

Para fins práticos, é como se nosso projeto estivesse dividido em vários projetos e cada um deles funcionasse de forma completamente independente. Uma vez terminado cada "sub-projeto", fundimo-lo com o ramo principal.

Temos também a vantagem de, se cometermos um erro no código do ramo, podermos detectá-lo e corrigi-lo antes da fusão com o ramo principal. Desta forma, o ramo master estará sempre livre de erros.

Quando arquivos diferentes estão sendo modificados em cada filial, geralmente não há problemas, mas se vários programadores estão modificando o mesmo arquivo em filiais diferentes, pode haver problemas de simultaneidade.

Imaginemos que no ramo master modificamos o arquivo *index.html* na linha 70 e no ramo secundário modificamos o mesmo arquivo na mesma linha. Quando se trata de fundir as duas mudanças, qual delas prevalece? Neste caso, *Git* nos dirá que existe um conflito de concorrência, uma vez que estamos modificando exatamente a mesma coisa (mesmo arquivo e mesma linha) em ramos diferentes e nos permitirá escolher qual deles manter.

Criar um Branch

Para criar uma filial, devemos usar o comando da *Branch* e especificar o nome que queremos dar à filial:

```
miPC@miPC MINGW64 /g/WORKSPACE/011
GitHub/Proyecto GIT (master)

$ git branch segundaRama
```

Criamos agora nossa filial. Se fizermos um git log -online para ver o status de nosso projeto:

```
miPC@miPC MINGW64 /g/WORKSPACE/011
GitHub/Proyecto GIT (master)

$ git log –oneline

28746f5 (HEAD -> main, tag: v1.0.3,
origin/main, origin/HEAD, segundaRama)
Update index.html
75d4418 Version 1.0.3
fd8ada9 Version 1.0.2
093f1c5 Version 1.0.0
00350f8 Version 1.0.0
```

Podemos ver que já existe uma referência à nossa nova filial (chamada secondBranch).

Com o comando do git branch, veremos os ramos do projeto e em que ramo estamos:

```
miPC@miPC MINGW64 /g/WORKSPACE/011
GitHub/Proyecto GIT (master)

$ git branch

* main
  segundaRama
```

Para nos mudarmos para o ramo infantil, usaremos o comando:

git checkout nombreRama

```
miPC@miPC MINGW64 /g/WORKSPACE/011
GitHub/Proyecto GIT (master)

$ git checkout segundaRama

Switched to branch 'segundaRama'
```

Se refizermos um git branch:

```
miPC@miPC MINGW64 /g/WORKSPACE/011
GitHub/Proyecto GIT (master)

$ git Branch

  main
* segundaRama
```

Podemos ver que já estamos na secondBranch.

De agora em diante, quaisquer mudanças que eu fizer em meu código serão refletidas apenas no ramo secundário. Os arquivos no ramo principal permanecerão inalterados.

Se tivéssemos que fazer qualquer mudança no conteúdo do arquivo index.html, por exemplo, teríamos que executar uma add e um commit. Como vimos anteriormente, podemos fazer as duas operações ao mesmo tempo com o comando:

git commit -am “descripción”

```
miPC@miPC MINGW64 /g/WORKSPACE/011
GitHub/Proyecto GIT (master)

$ git commit -am "Versión 1.0.5"


[segundaRama a1bd1cc] Versión 1.0.5
 1 file changed, 2 insertions(+)
```

Já temos uma primeira mudança feita no ramo secundário.

Mas o ramo mestre não vai refletir essas mudanças. De fato, se mudássemos agora para o ramo principal, as mudanças que acabamos de fazer não apareceriam no arquivo index.html. E se fizermos um git log –oneline no ramo master, não veremos o commit que fizemos no ramo secundário. Ou seja, tanto os commits como as mudanças nos arquivos que fazemos nos ramos extras não serão vistos de outro ramo, nem mesmo do ramo principal.

Para eliminar uma filial que criamos:

git branch -d "meu nome de ramo".

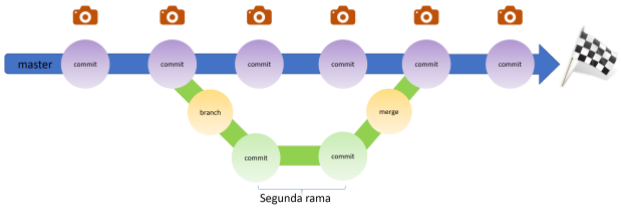
```
miPC@miPC MINGW64 /g/WORKSPACE/011
GitHub/Proyecto GIT (master)

git branch -d segundaRama

Deleted branch segundaRama (was
407013f).
```

Merge

Com o comando de merge, podemos fundir um ramo extra que criamos anteriormente no ramo master.



Em nosso projeto tínhamos criado uma filial extra que tínhamos nomeado secondaryBranch. Para fundir este ramo com o master, devemos usar o comando merge junto com o nome do ramo que queremos fundir.

A primeira coisa a fazer é mudar para o ramo master. Qualquer merge que quisermos fazer deve ser sempre feita a partir do ramo master. Em nosso caso, tínhamos chamado o ramo master de main.

As mudanças que fizemos em nosso index.html estavam em uma linha em que ninguém mais estava trabalhando. Mas suponha que façamos uma mudança a partir do ramo secundário em uma linha e outro programador tenha feito uma mudança na mesma linha no ramo master.

Conflitos

Vamos inserir o código na mesma linha do ramo master e do ramo secundário. Quando fizermos a merge, *Git* detectará o conflito e receberá uma mensagem como esta no console:

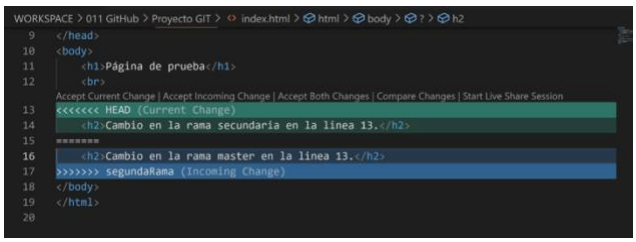
```
miPC@miPC MINGW64 /g/WORKSPACE/011
GitHub/Proyecto GIT (master)

$ git merge segundaRama

Auto-merging index.html
CONFLICT (content): Merge conflict in
index.html
Automatic merge failed; fix conflicts
and then commit the result.
```

Está nos dizendo que existem conflitos, que devemos resolvê-los manualmente e depois fazer a merge novamente.

E em nossa *IDE* também receberemos uma mensagem nos alertando sobre o problema:



Vemos como o *Visual Studio* detecta o problema e nos dá a opção de aceitar uma ou outra versão, compará-las, aceitar ambas... etc.

Agora teríamos que decidir qual versão queremos manter, salvar, fazer o commit e depois a merge.