**[Tutorial simples parte 2: git branch e merge](http://codexico.com.br/blog/linux/tutorial-simples-parte-2-git-branch-e-merge/" \o "Tutorial simples parte 2: git branch e merge)**

Browse: [Home](http://codexico.com.br/blog) / [git](http://codexico.com.br/blog/category/git/), [linux](http://codexico.com.br/blog/category/linux/) / Tutorial simples parte 2: git branch e merge

By [codexico admin](http://codexico.com.br/blog/author/admin/) on 16 de novembro de 2010

A parte 1 ([Como usar o git e o github](http://codexico.com.br/blog/linux/tutorial-simples-como-usar-o-git-e-o-github)) foi sobre instalar o git, ssh-key, criar um projeto no github e os commits básicos.

Em projetos super simples tudo bem, mas para qualquer coisa maiorzinha já é melhor usar branchs, é fácil também, nem tirei o *simples* do título do post.

Vou abordar os branchs de forma educacional, como se fosse um projeto de uma só pessoa. A parte 3 trará uma visão mais prática, com um fluxo de trabalho completo e voltado para equipes, graças ao git ainda será simples.

Será usado o mesmo [repositório da parte 1](https://github.com/codexico/tutorial-github).

**Branch**

Para criar um branch no diretório do projeto:

|  |  |
| --- | --- |
| 1 | $ git branch parte2 |

Atenção, somente foi criado o branch, o git não mudou para ele, qualquer alteração ou commit será feito no master. O git mostra onde está:

|  |  |
| --- | --- |
| 1 2 3 | $ git branch \* master   parte2 |

Para mudar para o branch é necessário um checkout:

|  |  |
| --- | --- |
| 1 2 | $ git checkout parte2 Switched to branch 'parte2' |

Os dois passos podem ser substituídos por um só (mas é sempre bom saber de onde veio):

|  |  |
| --- | --- |
| 1 | git checkout -b parte2 |

Agora sim as modificações serão feitas no branch correto. Que tal criar um diretório, uns arquivos e também alterar o README da parte 1?

|  |  |
| --- | --- |
| 1 2 3 4 5 6 | mkdir parte2 touch parte2/testeparte2 git add . git commit -m "arquivo para testes da parte2" vim README git commit -a -m "README parte2" |

Foram criados dois commits, ambos no branch *parte2*, um push agora não irá alterar nada no repositório.

**Merge**

Para enviar as modificações é necessário primeiro um checkout para voltar ao master, depois o merge com o branch e então o push.

Dica: TAB autocompleta os comandos e também os nomes dos branchs.

|  |  |
| --- | --- |
| 1 2 3 | git checkout master git merge parte2 git push origin master |

Foi enviado o código mas o github não percebeu que havia um branch pois o branch está somente na máquina local, para enviar o branch ao repositório o push muda um pouco:

|  |  |
| --- | --- |
| 1 | git push origin parte2:parte2 |

O formato é o seguinte *git push repositorioRemoto nomeLocalDoBranch:nomeRemotoDoBranch*, assim dá para enviar com um nome diferente se quiser.

|  |  |
| --- | --- |
| 1 2 3 4 | git checkout parte2 vim parte2/testeparte2  git commit -a -m "texto teste" git push origin parte2:parte2 |

Agora sim dá para ver que existe um branch, olha lá <https://github.com/codexico/tutorial-github/network>

Quando não for mais usá-lo, o branch pode ser deletado com:

|  |  |
| --- | --- |
| 1 | git branch -d nome-do-branch |

Legal, já deu pra perceber que dá por exemplo para trabalhar em uma nova funcionalidade do projeto sem alterar o código principal. Quando tudo estiver pronto, os testes passaram e tal, pode ser feito o merge e só então enviar para o repositório.

Pronto, acabou o básico sobre branch e merge com git.

**Branch e Merge exemplo 2**

Até agora foi muito básico, normalmente enquanto se está trabalhando num branch aparece um bug para ser arrumado no master, ou outro branch também está sendo trabalhado, ou seu sistema de desenvolvimento é dividido em produção, desenvolvimento, homologação e outros, são muitas possibilidades, cada lugar trabalha de uma maneira. Na parte 3 terá mais sobre isso.

Agora um exemplo um pouquinho mais trabalhado:

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 | git checkout -b funcionalidadeX touch funcionalidadeX git add . git commit -m "funcionalidadeX iniciada" vim funcionalidadeX git add . git commit -m "funcionalidadeX parcialmente executada" |

Foi iniciado um branch para a funcionalidadeX e depois de alguns commits ela ainda não está finalizada.

Então apareceu um bug no master que precisa ser corrigido rapidamente (colocar o link da parte2 antes do texto da parte2 do README, urgente né, o usuário é fogo).

|  |  |
| --- | --- |
| 1 2 3 4 5 6 | git checkout -b bug42 vim README git commit -a -m "bug 42 corrigido" git checkout master git merge --no-ff bug42 git branch -d bug42 |

Foi criado um branch para o bug, o bug foi corrigido, voltou ao master, foi dado um merge para incluir as alterações no master e então o branch foi deletado. A opção **–no-ff** é utilizada para forçar a criação de um commit, sem ela o merge não cria commit e fica mais difícil recuperar o código anterior.

Agora é só completar a funcionalidade X.

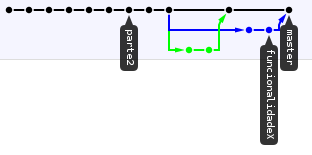
|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 | git checkout funcionalidadeX vim funcionalidadeX git commit -a -m "funcionalidadeX finalizada" vim README git commit -a -m "README atualizado com a funcionalidadeX" git checkout master git merge --no-ff funcionalidadeX |

Correu tudo bem? Pode ser que sim, o github vai fundir o código da funcionalidadeX com o master, que já contém a correção do bug. Mas e se a correção conflita com as alterações da funcionalidade X? Neste caso o git mostra um alerta e não permite o merge, mas mostra os conflitos:

|  |  |
| --- | --- |
| 1 | git status |

Os conflitos precisam ser corrigidos manualmente e então o merge será aceito. O git inclui marcações no código dos arquivos em conflito para ajudar na resolução do conflito.

Com um fluxo de trabalho melhorzinho dá para evitar muitos conflitos, logo mais na Parte 3 do tutorial…

No final os merges ficaram assim:  
[](https://github.com/codexico/tutorial-github/network)

Para finalizar uma tag:

|  |  |
| --- | --- |
| 1 2 | git tag -a v2.0 -m 'parte 2' git push origin v2.0 |

**Referências:**

[Pro Git – Basic Branching and Merging](http://progit.org/book/ch3-2.html)  
[github – Working with remotes](http://help.github.com/remotes/)  
[nvie.com – A successful Git branching model](http://nvie.com/posts/a-successful-git-branching-model/)

* [Email](http://codexico.com.br/blog/linux/tutorial-simples-parte-2-git-branch-e-merge/?share=email&nb=1)
* [Digg](http://digg.com/submit?url=http%3A%2F%2Fcodexico.com.br%2Fblog%2Flinux%2Ftutorial-simples-parte-2-git-branch-e-merge%2F&title=Tutorial%20simples%20parte%202%3A%20git%20branch%20e%20merge)

Posted in [git](http://codexico.com.br/blog/category/git/), [linux](http://codexico.com.br/blog/category/linux/) | Tagged [git](http://codexico.com.br/blog/tag/git/), [github](http://codexico.com.br/blog/tag/github/), [linux](http://codexico.com.br/blog/tag/linux/), [tutorial](http://codexico.com.br/blog/tag/tutorial/)

**Articles in this series**

* Tutorial simples parte 2: git branch e merge
* [Tutorial simples: Como usar o git e o github](http://codexico.com.br/blog/linux/tutorial-simples-como-usar-o-git-e-o-github/)
* *Pedro*

Muito bacana o tutorial… Me ajudou bastante ! Valeu!

* *Lau*

Muito bom, tirou muitas dúvidas. Parabéns!

* *Anônimo*

@Pedro, @Lau, muito obrigado! Assim me motiva a continuar.

Sugestões para a parte 3?

* + *svgit*

Poderia falar sobre o dia-a-dia, por exemplo, reverter um arquivo para revisão X. Até o momento, o tutorial cobre as possibilidades de um programador infalível! ;)

* *Anônimo*

svgit, verdade!  
Mas programadores são falíveis? Controle de versão é frescura de quem se diz ‘agile’! Hehehe, brincadeira ;)

Até escrevi rapidamente como recuperar commits em um comentário na parte 1 do tutorial, como vc é a segunda pessoa a solicitar esse tipo de informação acho que focarei neste assunto o próximo post.

Obrigado pelo feedback.

* *http://twitter.com/juliobitencourt Julio Bitencourt*

Não entendi bem o que fazer no momento que aparece o bug. Em qual branch eu executo o comando git checkout -b bug42 ?

Volto para o master ou mantem no funcionalidadeX?