FUNDAÇÃO
UNIVERSIDADE
FEDERAL DE
MATO GROSSO DO SUL

Gerência de Configuração de Software – T01

Aula 3 - Git Rebase, Fetch e Tags

Prof. Ricardo M. Kondo

Comando - Git Rebase

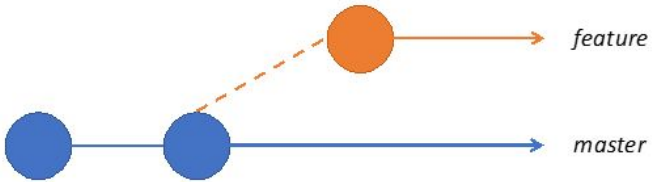
`git rebase <branch>`

- Semelhante ao Merge, porém é diferente na ordem de aplicar os commits
- No Resabe, os seus commits na frente da base são removidos temporariamente, os commits de outras branch são aplicadas na sua branch e por fim seus commits são aplicados um a um
- Pode acontecer conflitos que serão resolvidos para cada commit
- Normalmente utilizado quando se cria muitos novos arquivos e outras modificações que não geram conflitos

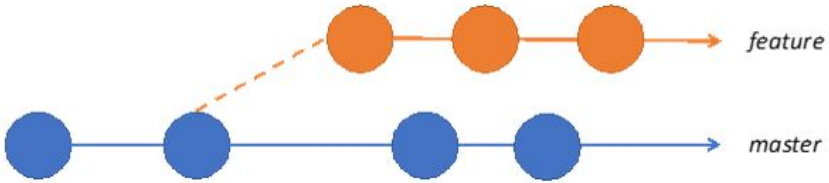
Diferenças git rebase x git merge

git merge

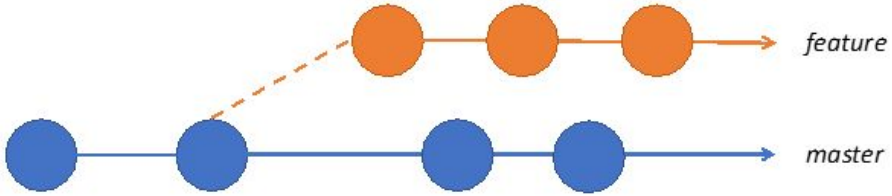
- criar branch a partir da master



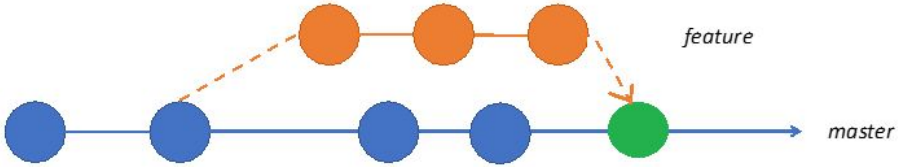
- gerar os commits, salvando nosso trabalho gradativamente...



- … enquanto outras pessoas podem ir misturando seus commits a master



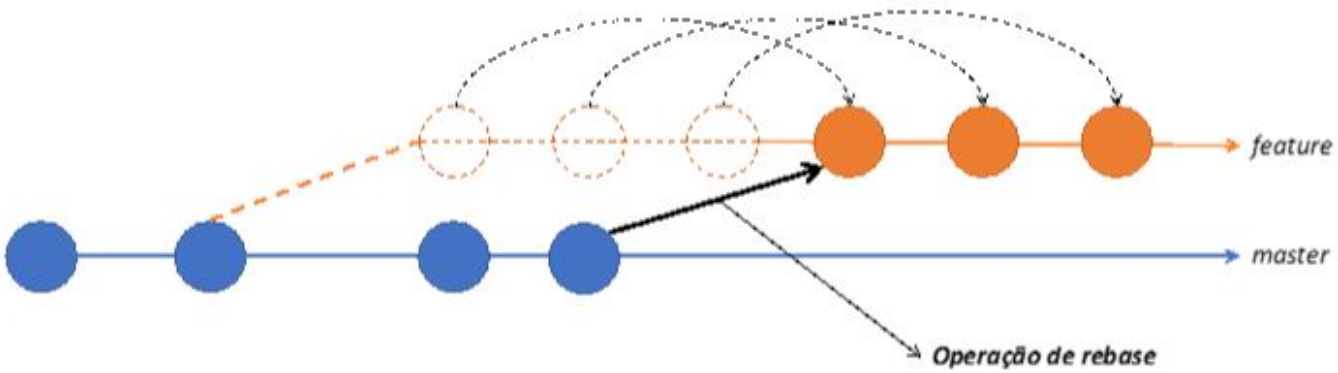
- integrar nossos commits a master



Diferenças git rebase x git merge

git rebase

- trazer os commits de master para feature de maneira “transparente”, ou seja: sem a necessidade de um commit adicional

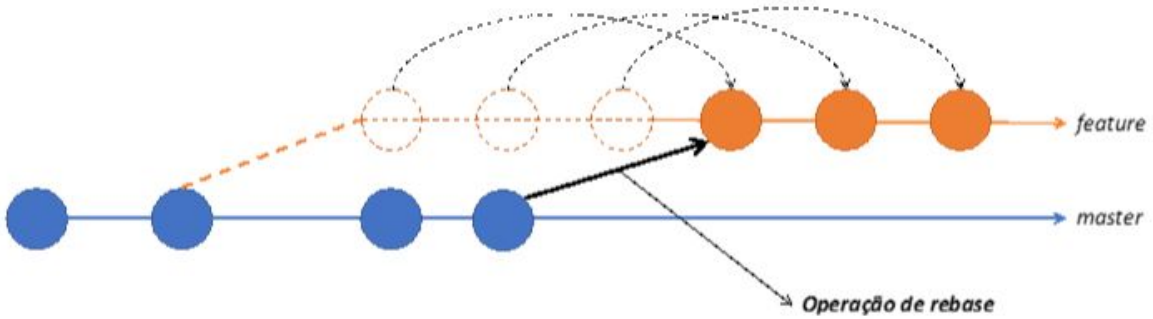


Diferenças git rebase x git merge

- branch master, que é o branch principal, ainda está defasado, pois ele ainda não possui os nossos commits. Nós precisamos puxar nossos commits do branch feature para o branch master. Como estamos falando do branch principal, é importante mantermos a rastreabilidade do processo de gestão do código, deixando claro inclusive quando os commits de branches locais foram puxados para o branch master.
- Por isso, para essa segunda etapa, o merge é muito mais interessante do que o rebase.

Diferenças git rebase x git merge

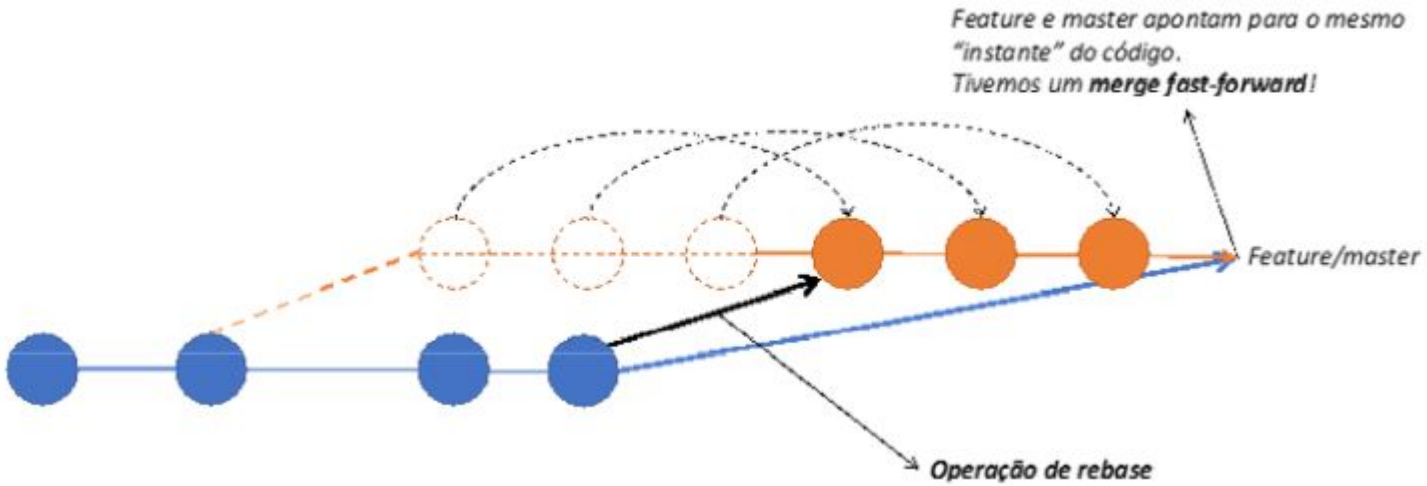
- Nesta situação, todos os possíveis problemas de “mistura” dos dois branches (como conflitos) foram resolvidos durante o processo de rebase. Nesta segunda etapa, nós podemos simplesmente trocar o ponteiro do nosso branch master, apontando-o para o último commit do nosso branch feature. Quando esse processo acontece, nós chamamos esta estratégia de merge de fast-foward. O fast-foward é possível quando não existem conflitos entre os branches envolvidos e quando o branch de desenvolvimento está à frente do branch principal. E, pela ilustração abaixo, é exatamente esta a situação.



Diferenças git rebase x git merge

`git checkout master` # indo para o branch do master

`git merge feature` # fazendo o merge entre o master e o feature



Diferenças git rebase x git merge

- Nossa branch master agora está da seguinte maneira:

- Dessa maneira, temos o melhor dos dois mundos:
  - não temos o histórico de commits poluído pelos commits secundários no branch feature (graças ao rebase)
  - temos a rastreabilidade do momento onde o branch master foi modificado por uma integração (por causa do merge).

Exemplo

- Faça um commit na master e outro em uma branch
- Faça o rebase da branch com a master
- Veja a ordem de commits
- Mesmo que tenha conflitos o histórico de commits é preservado.

Passo a Passo

- Na master
  - git checkout -b branch
  - Fazer um commit na master (git add e git commit)
- Na branch
  - Fazer um commit (git add e git commit) no mesmo arquivo no qual realizei a alteração na master
- Juntar os arquivo
  - Na branch criada
    - git rebase master
    - voltou o commit, retirou o commit que havia dado e adicionou o commit da master
    - resolver conflito
      - Não precisa fazer o commit igual ao merge
    - git add .
    - git status
    - git rebase --continue
    - git log # verificar o log de atividades

Git fetch

```
git fetch
```

- Baixa as atualizações do remote, porém, não aplica elas no repositório
- Permite fazer o rebase de uma branch em vez de fazer o merge
- Pull = Fetch + Merge
- Fetch e o Rebase é melhor para manter o histórico do desenvolvimento

Git tag

- Útil para definir versões estáveis do projeto
- Semelhante a branch, porém não recebe mais commits
- Guarda um estado do repositório

```
git tag [nome da tag]
```

```
git push <remote> <tag>
```

Git tag

- Acessar uma tag
  - git checkout [nome da tag]
- Pode criar uma branch da tag
  - git checkout -b [nome da branch]
  - git branch [nome da branch]

Git Commit Amend

`git commit --amend`

- Altera o último commit
  - Mensagem de commit
  - Adiciona arquivos

Exemplo:

- Criar uma branch
- Realizar a alteração de um arquivos do seu projeto.
- Adicionar as modificações e dar commit
- Em seguida, na mesma branch modificar o mesmo arquivo e adicionar uma nova informação.
- Aplicar o comando git commit --amend
- Verificar o que aconteceu

Git stash

`git stash`

- Guarda as alterações do *Working Directory* para poder ser utilizado depois
- Permite fazer rebase, merge, trocar de branch sem a necessidade de fazer um commit

`git stash list` // exibe todos stash existente

`git stash pop`

- Aplica o último stash armazenado

Git Stash

Exemplo:

- Acessar uma branch do seu projeto
- Modificar um arquivo dessa branch
- Mudar para um outra branch do projeto e alterar um arquivo
- Adicionar as modificações e realizar um commit
- Voltar para a primeira branch e alterar o mesmo arquivo da branch anterior.
- Mudar para a segunda branch, perceba que será exibida uma mensagem no qual é necessário dar um commit nas alterações dessa branch
- Como eu não quero commitar isso agora, vamos utilizar o comando git stash
- Trocar para a segunda branch (git checkout branch)
- Volto para a primeira branch e aplico o git stash pop
    - Perceba que os arquivos modificados voltaram