Git 102 - MAC0110

Matheus Tavares

matheus.bernardino@usp.br

Agenda

1. Conversa: como foi a experiência com Git até agora? Dificuldades, dúvidas, comentários? Rever algo de Git 101?
2. Comentários sobre MiniEP6
3. Branching
4. Merge
5. Resolução de conflitos

Como foi a experiência com Git até agora?

Alguns comentários do MiniEP6

Comentários gerais

- No geral, organização, mensagens e conteúdo dos commits, muito bons :)
- Alguns usaram o corpo da mensagem para descrever detalhes. Legal!
- Alguns fizeram commits adicionais (também bem padronizados)
  - Traduzindo nomes para PT, melhorando o README, etc.
  - E.g.: "Correct minor grammatical problem in MiniEP6's 2.2 exercise answer"
- Alguns adicionaram um README mais detalhado.

Possíveis melhorias

- Alguns commits e/ou mensagens de commit duplicados.
- Algumas mensagens foram muito genéricas (em alguns dos commits “extra”, i.e., além dos pedidos no enunciado).
- Mensagens fora do padrão (título em 2 linhas, tempo verbal incorreto ou linhas com mais de 72 colunas no corpo).
- Mensagens com mistura de PT e EN.
- Separar mudanças não relacionadas (resposta do 2.1 e 2.2).

Algumas dicas

- Cuidado para não misturar estilos de indentação (tabs e espaços)!
- Evitar linhas com espaços adicionais ao fim.

(Ex: veja como git-log “acusa” ambos os problemas acima)

- Confira o que será *commitado* antes de fazer o commit.
    - Alguns adicionaram arquivos .swp
    - Evitar *git add -A* ou *git add .*
    - Usar *git status,* sempre!
    - Use *git diff --staged* para ver o que será *commitado.*

Branches e Merge Básico

Imagine que você quer construir uma bicicleta

Podemos trabalhar em uma única peça de cada vez

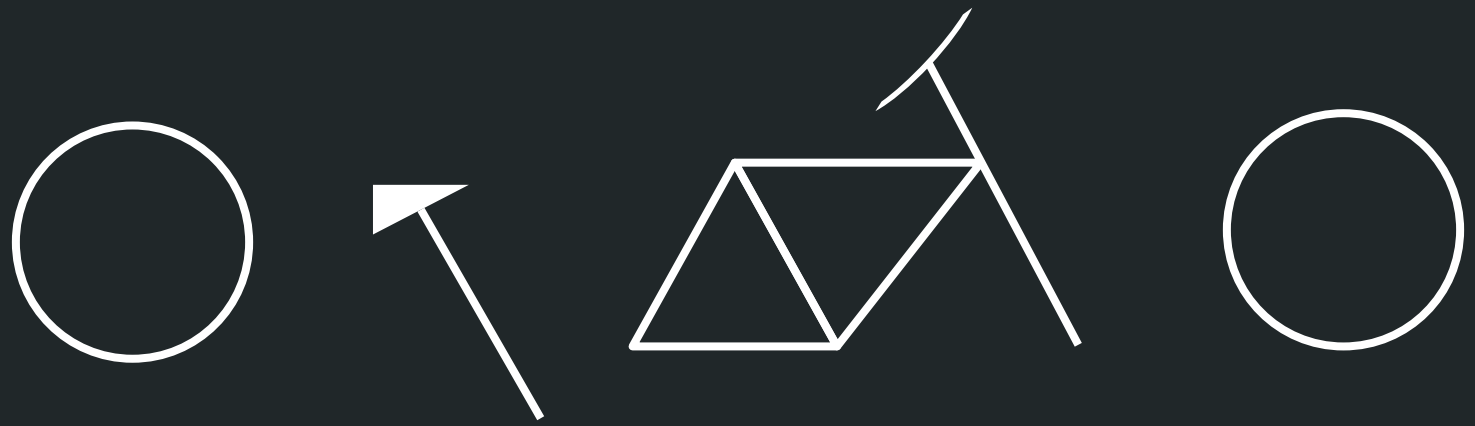
Podemos trabalhar em uma única peça de cada vez

Podemos trabalhar em uma única peça de cada vez

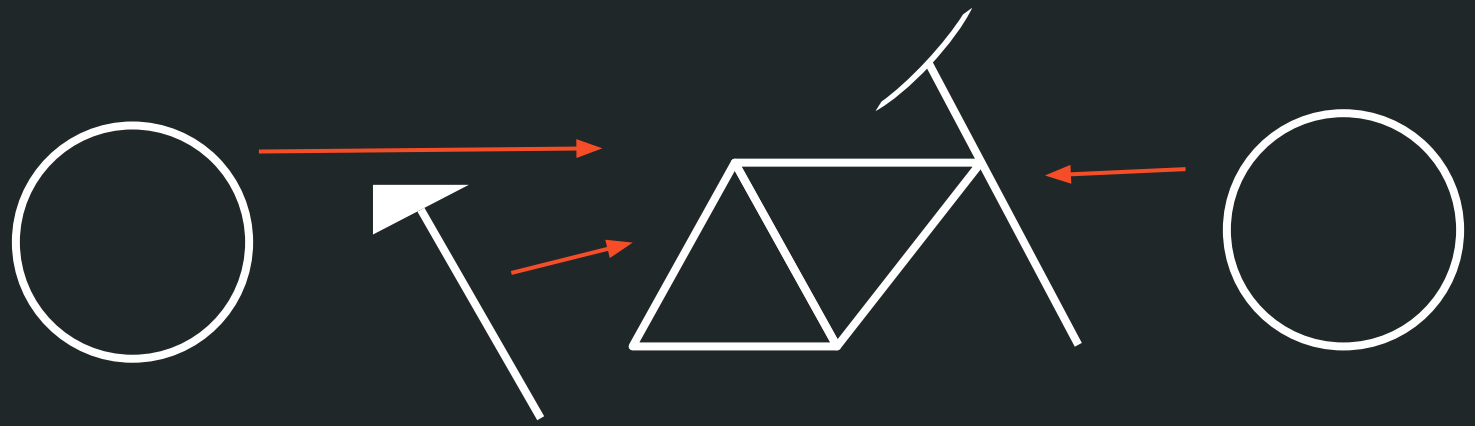
Podemos trabalhar em uma única peça de cada vez

Ou...

Cada peça pode ser fabricada em paralelo...



...e depois *integradas*



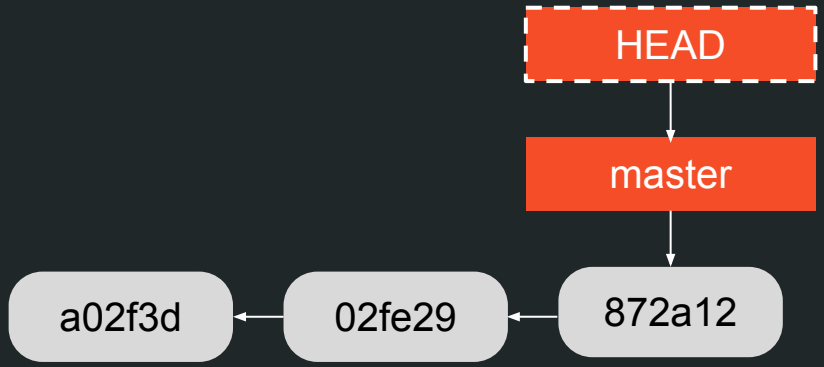
...e depois *integradas*

Isto te permite:

- Dividir o trabalho em uma equipe.
- Não “*travar*” o desenvolvimento de uma *feature* B enquanto a A não está pronta.
- Resolver problemas mais emergenciais (e.g. bugfixes), sem ter que abandonar a(s) *feature(s)* que está trabalhando agora.

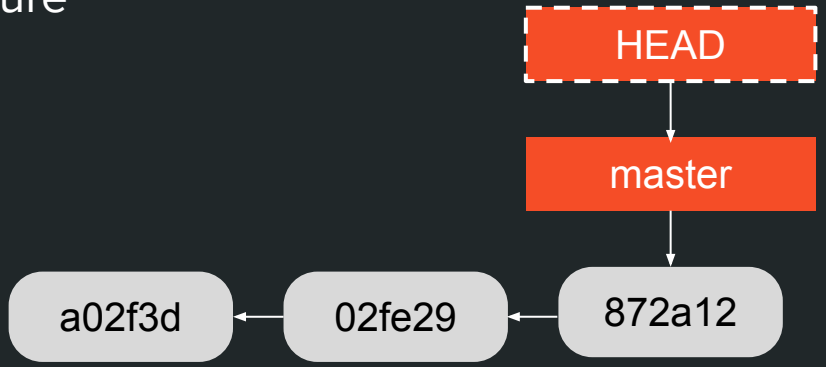
Em versionamento, cada linha de desenvolvimento é uma ***branch***.

Referências: branches



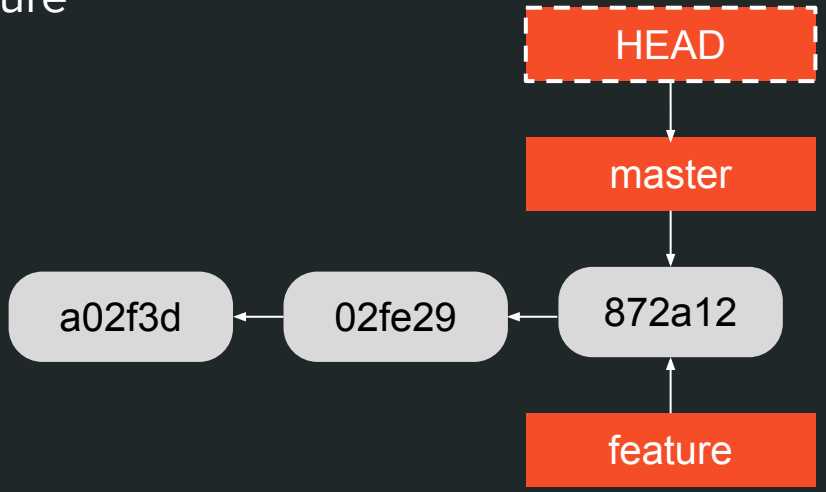
Referências: branches

$ git branch feature



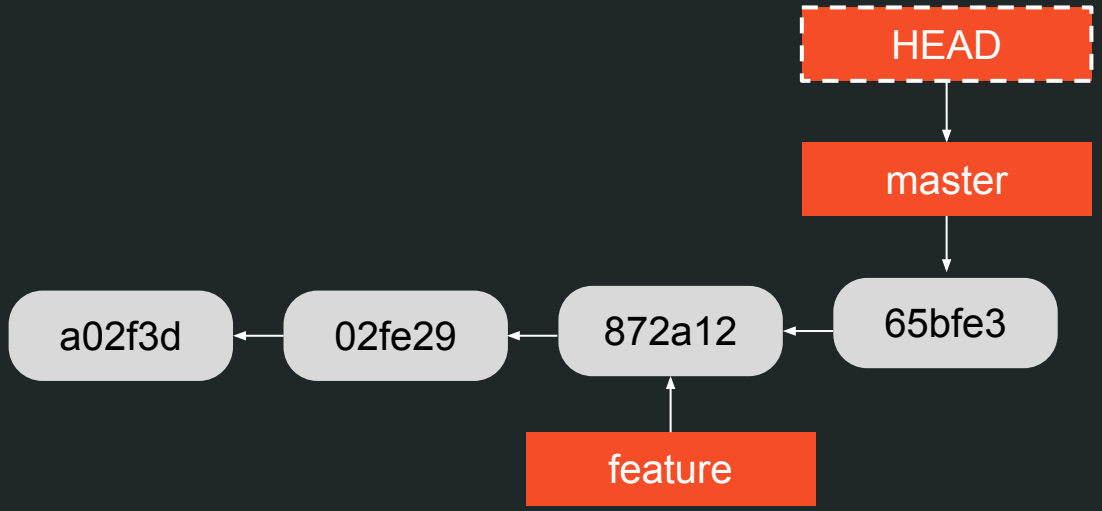
Referências: branches

$ git branch feature



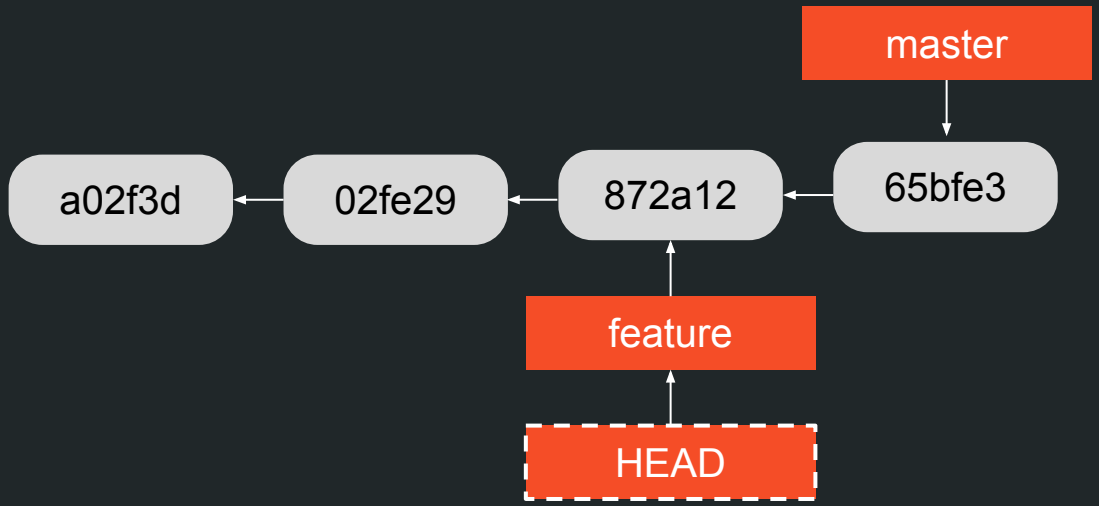
Referências: branches

$ git commit



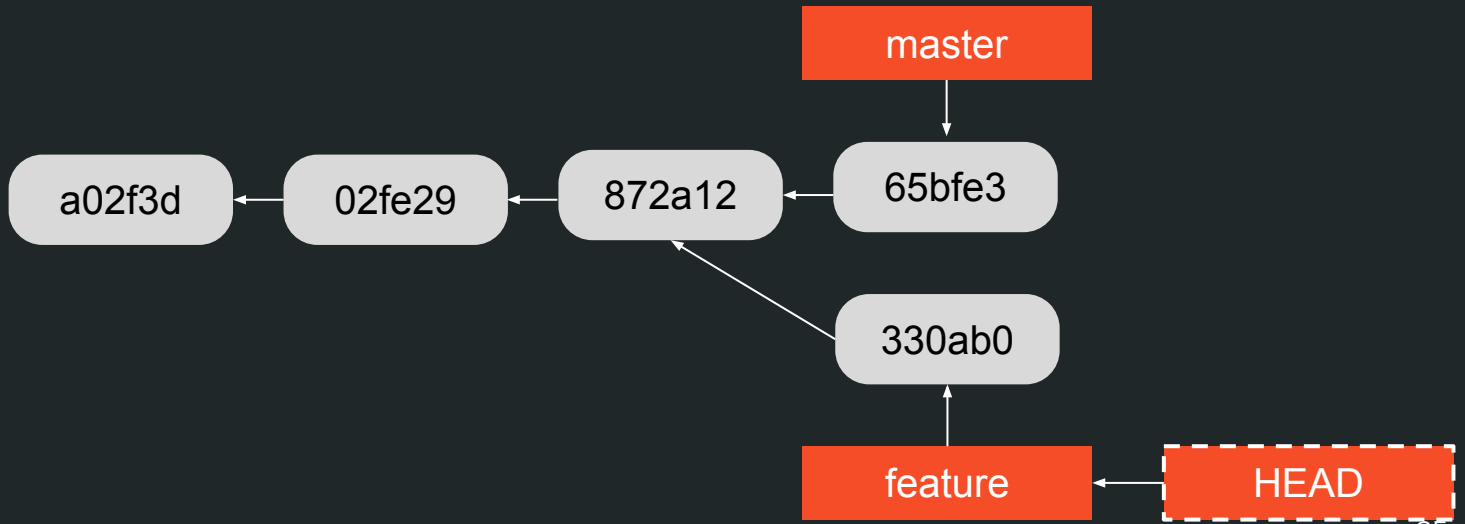
Referências: branches

$ git checkout feature



Referências: branches

$ git commit

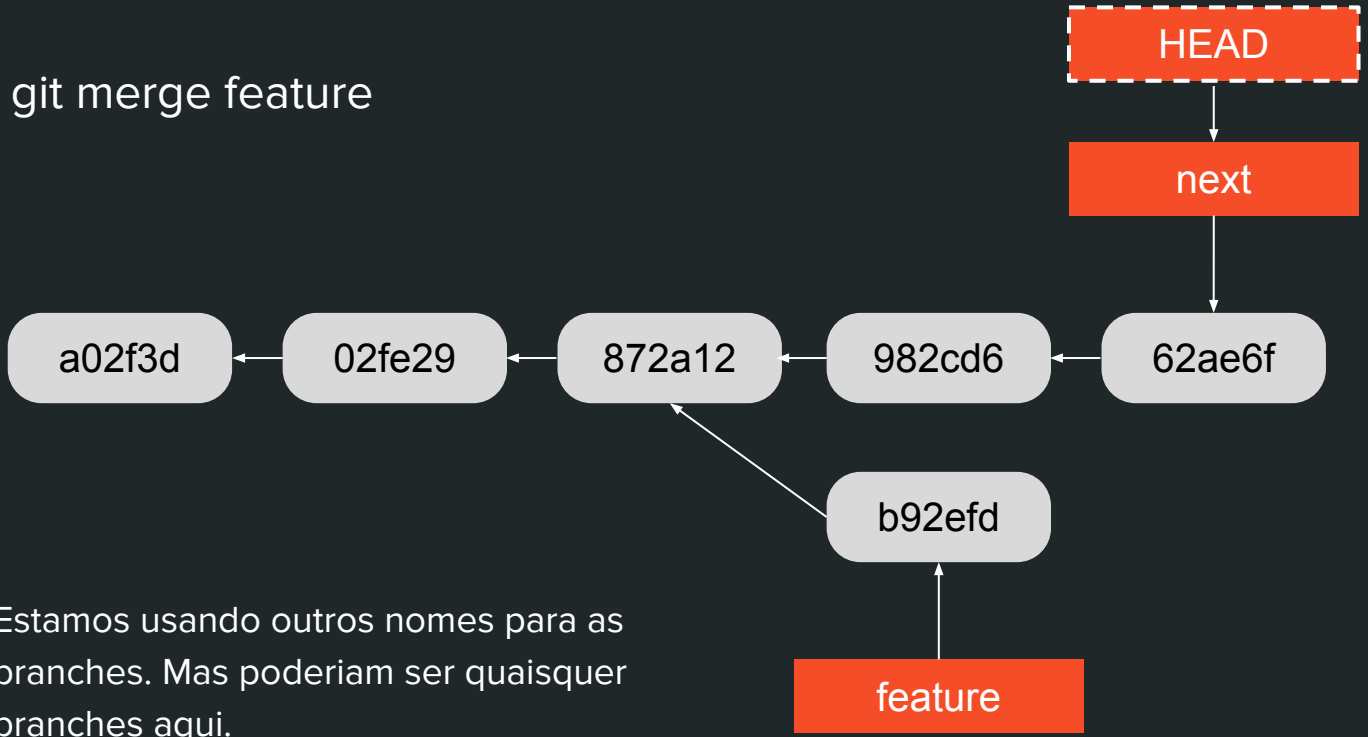


Merging

https://giphy.com/gifs/git-merge-cFkiFMDg3iFoI

Merging

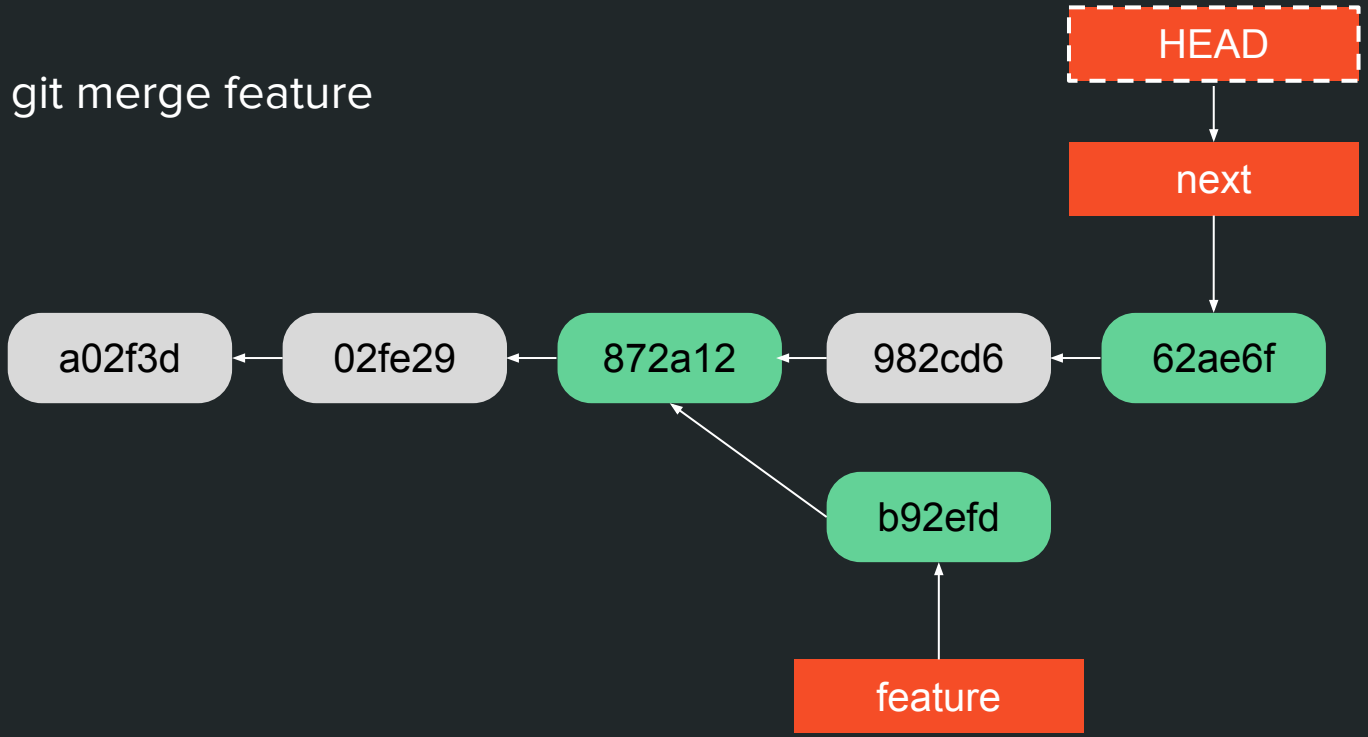
$ git merge feature



Estamos usando outros nomes para as branches. Mas poderiam ser quaisquer branches aqui.

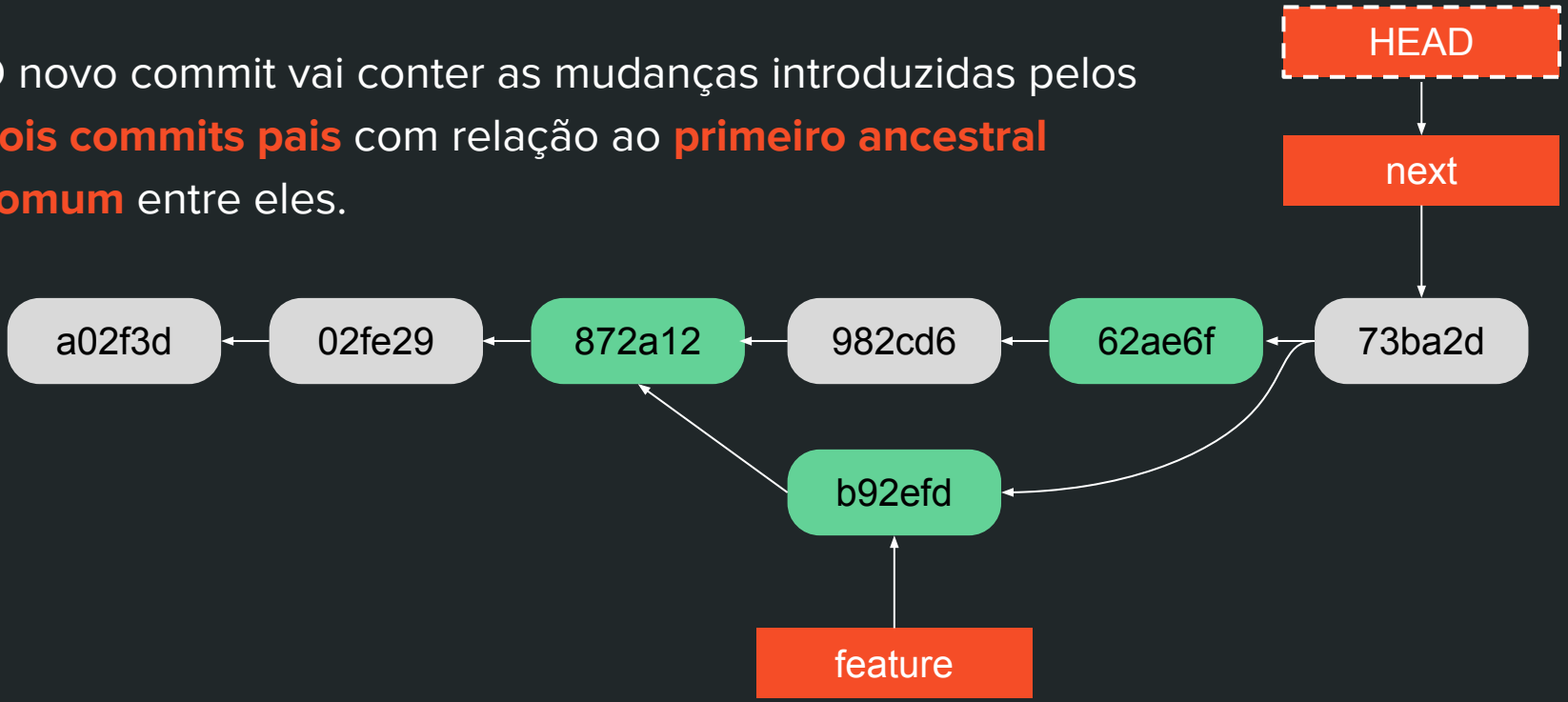
Merging

$ git merge feature



Merging

O novo commit vai conter as mudanças introduzidas pelos **dois commits pais** com relação ao **primeiro ancestral comum** entre eles.



Como funciona

Base

```
...
if (enemy == “grievous”)
  println(“hello”);
```

next

```
...
if (enemy == “grievous”)
  println(“hello there”);
```

feature

```
...
if (enemy == “grievous” && me == “obi-wan”)
  println(“hello”);
```

Result

```
...
if (enemy == “grievous” && me == “obi-wan”)
  println(“hello there”);
```

E se ambos alterarem a mesma linha?

Base

```
...
if (enemy == “grievous”)
  println(“hello”);
```

next

```
...
if (enemy == “grievous”)
  println(“hello there”);
```

feature

```
...
if (enemy == “grievous” && me == “obi-wan”)
  println(“Olá”);
```

E se ambos alterarem a mesma linha?

Base

```
...
if (enemy == “grievous”)
  println(“hello”);
```

next

```
...
if (enemy == “grievous”)
  println(“hello there”);
```

feature

```
...
if (enemy == “grievous” && me == “obi-wan”)
  println(“Olá”);
```

Result

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

Resolvendo conflitos

```
...
if (enemy == “grievous” && me == “obi-wan”)
<<<<<<< HEAD
  println("hello there");
=======
  println("Olá”);
>>>>>>> feature
```

- Entenda porque *next* **e** *feature* modificaram aquela linha.
- Escolha o lado a ser mantido, ou
- Faça uma “mistura” dos dois.

Resolvendo conflitos

```
...
if (enemy == “grievous” && me == “obi-wan”)
  println("Olá a todos”)
```

- Entenda porque *next* e *feature* modificaram aquela linha.
- Escolha o lado a ser mantido, ou
- Faça uma “mistura” dos dois.

Resolvendo conflitos

```
...
if (enemy == “grievous” && me == “obi-wan”)
  println("Olá a todos”)
```

- Marque o conflito como resolvido:
  $ git add greetings.c
- Finalize o merge:
  $ git merge --continue
- O editor será aberto para você editar a mensagem do commit de merge.

Referências

1. **Pro Git**, Scott Chacon and Ben Straub: https://git-scm.com/book/en/v2
2. **Git Docs:** https://git-scm.com/docs/
3. **The Zen of Git**, Tianyu Pu: https://speakerdeck.com/tianyupu/the-zen-of-git

Obrigado!

https://matheustavares.gitlab.io