【Git】コンフリクトの発生と適切な対応方法

概念説明

コンフリクトとは？

コンフリクトは「 衝突」や「対立」といった日本語の意味を持ちますが、Gitの世界では異なるブランチ間で発生したソースコードの変更内容を安全に合成(マージ)出来ない状態のことを示します。

具体的には、**同一ファイルに対して異なる変更が発生**した場合に「コンフリクトした」という様な使い方をしますが、これは主に複数人のエンジニアが1つのソースに対して同時並行して開発(ソースコードの変更)をする際に発生する概念です。

何故コンフリクトが発生するのか

例えば、下記の様な日本語の文字列が書かれた`animal.txt`というファイルのソースコードがあったとします。

```
animal.txt

いぬ、ねこ、うさぎ
```

このファイルを多言語対応するため、Aさんは英語対応を、Bさんは韓国語対応を担当をすることになりました。

```
animal.txt(Aさん)

いぬ、ねこ、うさぎ
Dog, Cat, Rabbit
```

```
animal.txt(Bさん)

いぬ、ねこ、うさぎ
개, 고양이, 토끼
```

この状態でAさんとBさんの変更を同時に合成(マージ)しようとした場合、`animal.txt`の2行目のコードが全く異なる内容で変更されてしまっているため、Git上でどちらを正として合成すればよいかが判断できません。

ですが、この時Aさんと Bさんの変更内容はどちらも正しいため、どちらかが間違っているということではありませんね。

この様に実際の開発現場では、1つのソースコードに対して同時並行して複数人のエンジニアが開発を進めているため、それぞれのエンジニアが行った変更内容がいずれも正しい場合であっても、そのまま合成することが出来ない場合があるのです。

安全にコンフリクトを解消する

AさんとBさん、2人の変更内容はどちらも正しいため、これらが適切に反映される様に修正対応をする必要があります。

例えば2人が変更した内容を`animal.txt`ファイル内で、別々の行に切り分けて合成すれば問題は解決しそうですね。

```
animal.txt

いぬ、ねこ、うさぎ
Dog, Cat, Rabbit
개, 고양이, 토끼
```

この様にコンフリクトが発生した場合は、どちらかの変更を破棄するのではなく、両方の変更内容が適切に反映されるように対応方針を検討する必要があります。

このコンフリクト対応をおざなりにしてしまうと、異なる変更内容を合成した時点でいずれか一方もしくはその両方の内容が適切に機能せずに不具合を生んでしまう可能性があるのです。

そのため、**コンフリクトの解消は開発現場ではとても重要な業務**の1つだと言えます。

コンフリクトの発生

複数ブランチの作成

今回は擬似的に複数人での開発現場の状態を再現するために、予め複数のブランチを作成したいと思います。

ブランチの分岐元は`master`ブランチとし、そこから`feature-a`と`feature-b`というブランチを切ることにしましょう。

まずは現在のブランチの確認をします。

```
command

$ git branch


output

* master
```

現在は`master`ブランチにいるので、このままfeatureブランチを作成していきましょう。

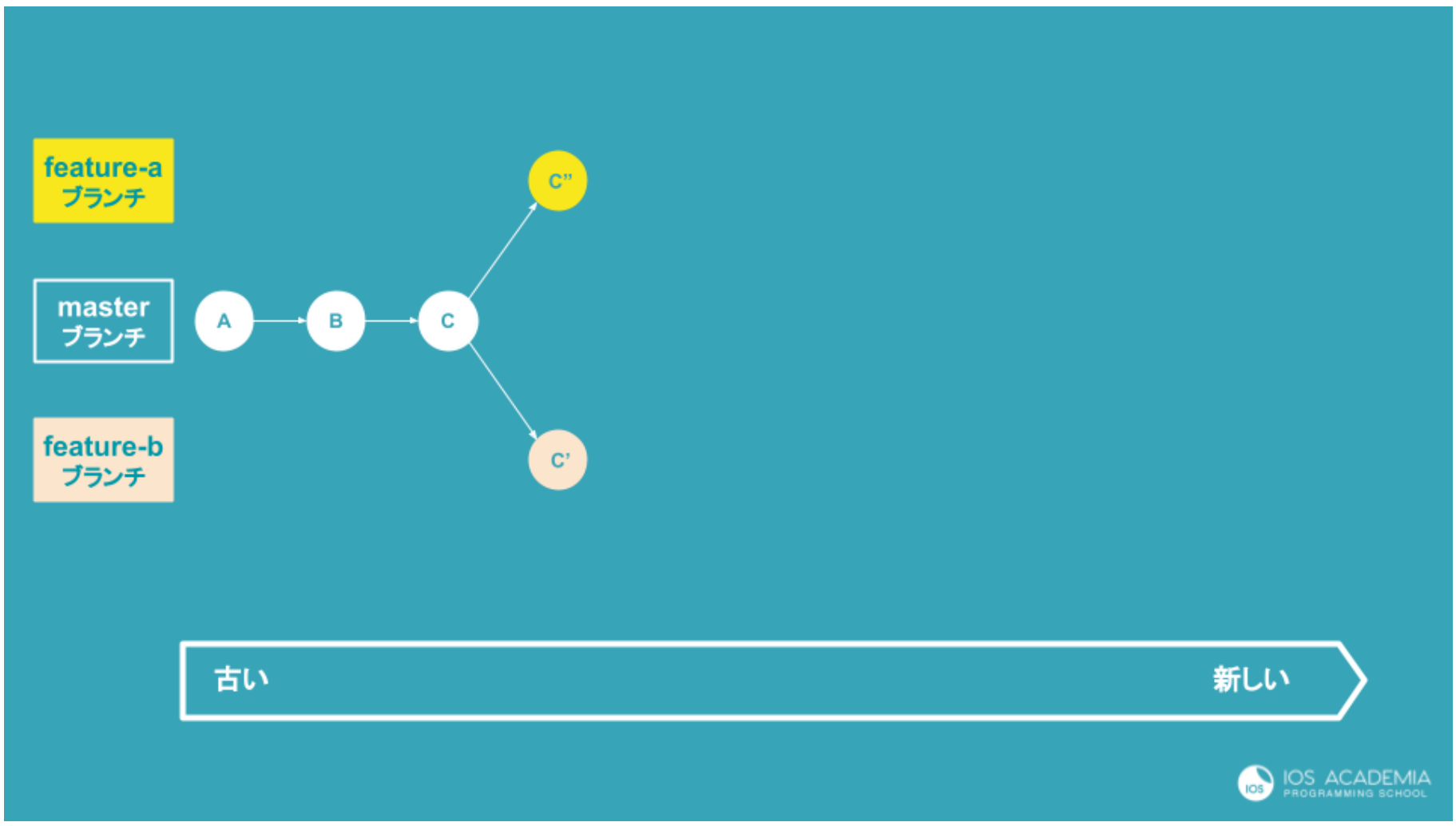
```
command

$ git branch feature-a
$ git branch feature-b
```

はい、ではもう一度ブランチの状況を確認してみます。

```
command

$ git branch


output

  feature-a
  feature-b
* master
```

これで正常に2つのfeatureブランチを作成できました。

イメージとしては、下記の様になります。



feature-bでコミットしてマージ

さて、では今回作成したブランチに対して、便宜上`feature-a`はAさんというエンジニアが、`feature-b`はBさんというエンジニアがそれぞれ開発を行っていると仮定します。

まずはBさんが開発を行っていることを想定し、`feature-b`ブランチ上へ移動してコミットをしていきましょう。

```
command

$ git checkout feature-b


output

Switched to branch 'feature-b'
```

これでブランチの切り替えができたので、早速変更を加えていきたいと思います。

まずは新しく`third.txt`ファイルを作成し、このファイルに文字列を記載しましょう。

```
command

$ touch third.txt
$ open third.txt
```

```
third.txt

コンフリクトについて学習中！
```

次にこの変更内容をコミットしておきます。

```
command

$ git add third.txt
$ git commit -m "third.txtに文字列を追加"
```

```
output

[feature-b 15d28cd] third.txtに文字列を追加
 1 file changed, 1 insertion(+)
```

さて、これでコミットができたので、この変更内容を`master`ブランチにマージするために、ブランチを移動してマージコミットを追加していきます。

```
command

$ git checkout master
$ git merge --no-ff feature-b
```

```
vim

Merge branch 'feature-b'
# Please enter a commit message to explain why this merge is necessary,
# especially if it merges an updated upstream into a topic branch.
#
# Lines starting with '#' will be ignored, and an empty message aborts
# the commit.
```

そうするとvimが立ち上がってマージコミットのコミットメッセージ入力が求められるので、このまま保存しておきます。

```
output

Merge made by the 'recursive' strategy.
 third.txt | 1 +
 1 file changed, 1 insertion(+)
```

これで`feature-b`ブランチの変更内容が適切に反映されたはずなので、コミットログを確認してみましょう。

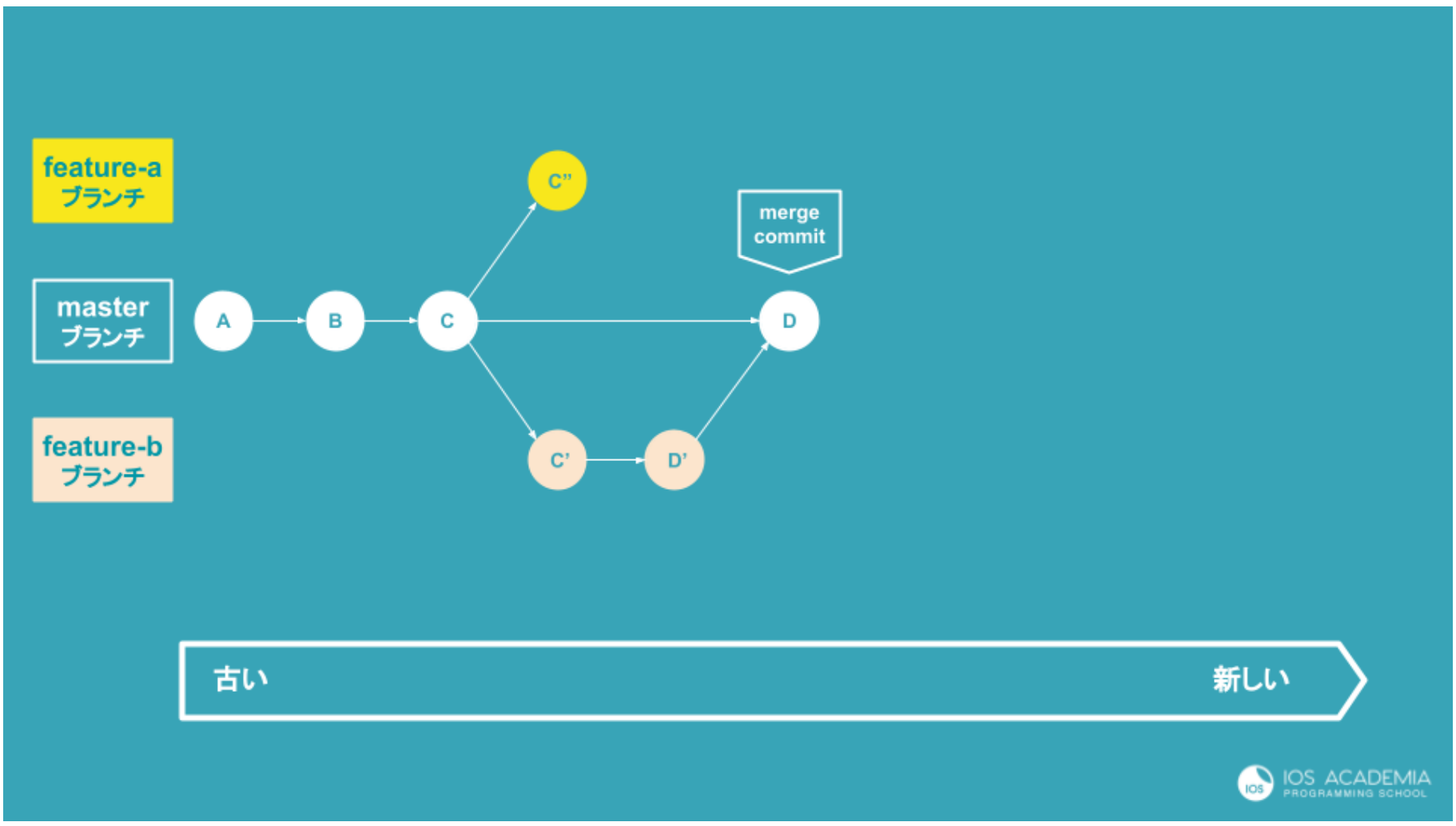
```
command

$ git log
```

```
output
```

```
commit d15e16219c3df43ac3bdcae36ed79ad9d5ffd40d (HEAD -> master)
Merge: 1c1db0c 15d28cd
Author: yamataku29
Date:   Wed Apr 14 09:56:19 2021 +0900

    Merge branch 'feature-b'
```

この様に`Merge branch 'feature-b'`というコミットメッセージが書かれたマージコミット残っているので、正常に`feature-b`のマージを行うことが出来ました。

今の状態は下記の様なイメージとなります。



feature-aでコミットしてマージ

では次にAさんが開発をしたことを想定するために、`feature-a`ブランチに移動してコミットをしましょう。

```
command

$ git checkout feature-a


output

Switched to branch 'feature-a'
```

先ほどと同じ様に`third.txt`の中身を書き換えていきます。

```
third.txt

Learning about conflicts!
```

次にこの変更内容をコミットしておきます。

```
command
```

```
$ git add third.txt
$ git commit -m "third.txtに文字列を追加"
```

```
output

[feature-a f4a8a4c] third.txtに文字列を追加
 1 file changed, 1 insertion(+)
```

では先ほどの`feature-b`の時と同様に、`master`ブランチに移動してマージしていきます。

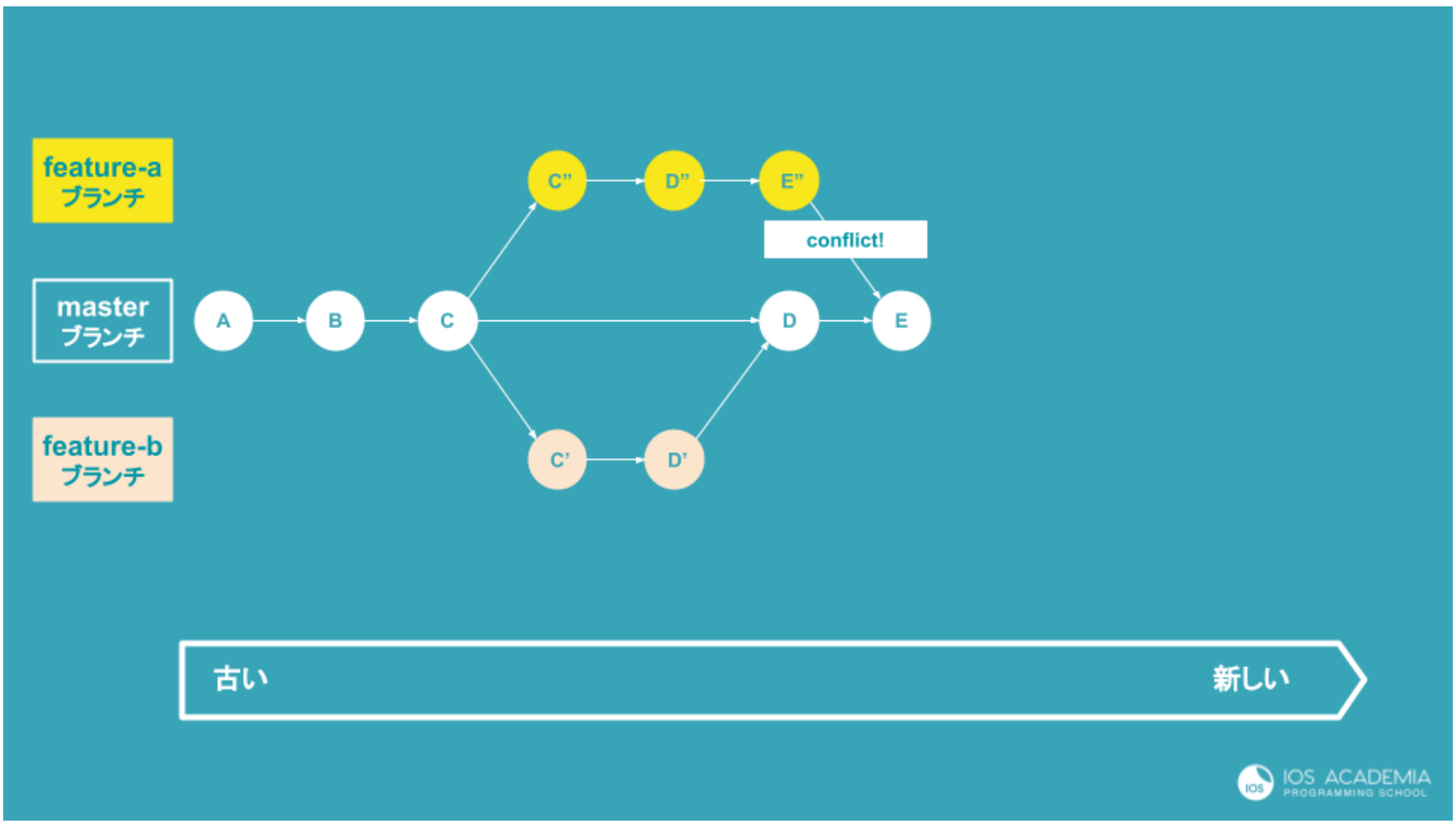
```
command

$ git checkout master
$ git merge --no-ff feature-a
```

```
output

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

そうすると、この様にマージが正常に行えなかったというエラーメッセージが出力されました。

よく見ると`Merge conflict in third.txt`とあるので、マージをする際に`third.txt`がコンフリクトしている事が分かりました。

今の状態としては以下のイメージとなります。



このままだと`master`ブランチ上に仮保存もコミットもされていない宙ぶらりんな変更内容が残ってしまうので、この差分は一旦破棄しておきましょう。

```
command

$ git reset --hard HEAD
```

```
output

HEAD is now at d15e162 Merge branch 'feature-b'
```

これでひとまず`master`ブランチ上の不要な差分を破棄できました。

コンフリクトの安全な解消

ベースとなるブランチの変更を取り込む

では、今回発生したコンフリクトを安全に解決をしていきたいと思います。

まずはコンフリクト発生の原因となった`feature-a`ブランチに移動しましょう。

```
command

$ git checkout feature-a


output

Switched to branch 'feature-a'
```

次に`msater`ブランチとのコンフリクトを解消するために、一旦`feature-a`に対して`master`ブランチの変更内容を取り込みます。

```
command

$ git merge --no-ff master


output

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

そうすると、先ほどと同じ様にコンフリクトが発生しているため、マージに失敗したという出力がされました。

この時点で一度ソースコードの差分を見てみます。

```
command

$ git diff


output
```

```
diff --cc third.txt
index 77a9ffe,9151d76..0000000
--- a/third.txt
+++ b/third.txt
@@@ -1,1 -1,1 +1,5 @@@
 -コンフリクトについて学習中！
++<<<<<<< HEAD
 +Learning about conflicts!
++=======
++コンフリクトについて学習中！
++>>>>>>> master
```

そうするとコンフリクトが発生した`third.txt`上でこの様な差分が発生している事がわかりました。

この差分の見方としては以下の通りとなります。

feature-aの差分

```
++<<<<<<< HEAD
 +Learning about conflicts!
++=======
```

masterの差分

```
++=======
++コンフリクトについて学習中！
++>>>>>>> master
```

これは`<<<<<<< HEAD`となっているのが今のブランチのHEADコミットに当たる部分で、`>>>>>>> master`となっているのが`master`ブランチ上での差分に当たる部分ですよ。という意味を示しています。

つまり同じファイルの同じ行に対して異なる変更が加わったため、Git上ではどちらが正しいか判断できない状態になっている。ということですね。

コンフリクトの解消とマージ

この状態だと適切にマージが出来ないので、Bさんが過去に開発した`master`ブランチの変更と、Aさんが今回開発した`feature-a`ブランチの変更の両方が崩れない様に修正を加えていきます。

今回は、日本語文の後に括弧を付けて英語文を記載する形式にすることになりました。

third.txt

```
コンフリクトについて学習中！(Learning about conflicts!)
```

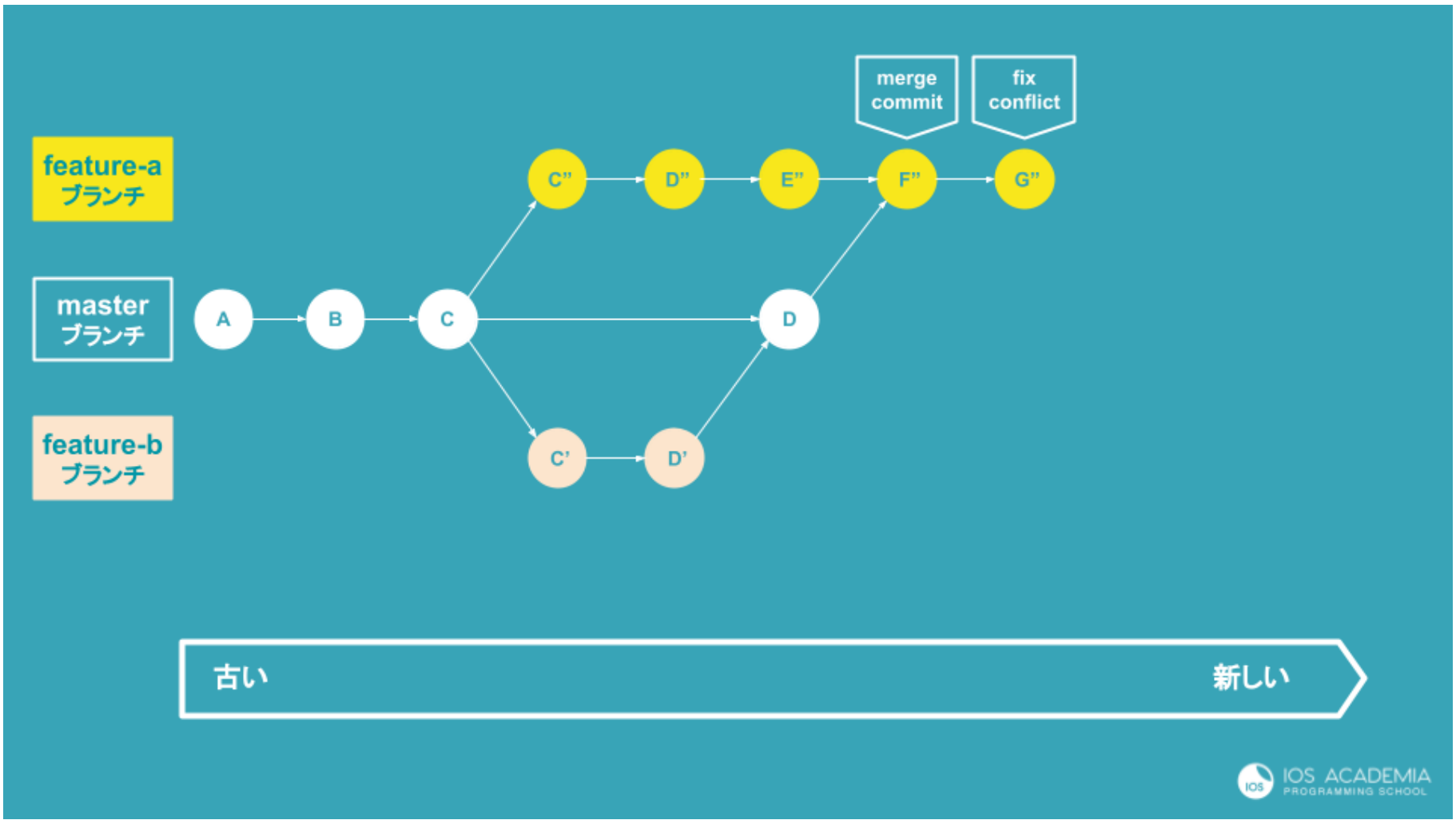
これで変更内容を保存するためにコミットをしましょう。

command

```
$ git add third.txt
$ git commit -m "masterをマージしてコンフリクト解消"
```

```
output

[feature-a c96982e] masterをマージしてコンフリクト解消
```

これでコンフリクトが解消されてコミットもされたので、以下の様な状態となりました。



この状態であれば、`feature-a`ブランチを安全に`master`ブランチにマージする事ができるので、`master`ブランチに移動してマージコミットをしましょう。

```
command

$ git checkout master
$ git merge --no-ff feature-a


vim

Merge branch 'feature-a'
# Please enter a commit message to explain why this merge is necessary,
# especially if it merges an updated upstream into a topic branch.
#
# Lines starting with '#' will be ignored, and an empty message aborts
# the commit.
```

ここでもvimが立ち上がるので、このまま保存します。

```
output

Merge made by the 'recursive' strategy.
 third.txt | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
```

はい、これで正常に`feature-a`ブランチも`master`ブランチへマージできました。

では、念のためにコミットログを確認してみましょう。

```
command
```

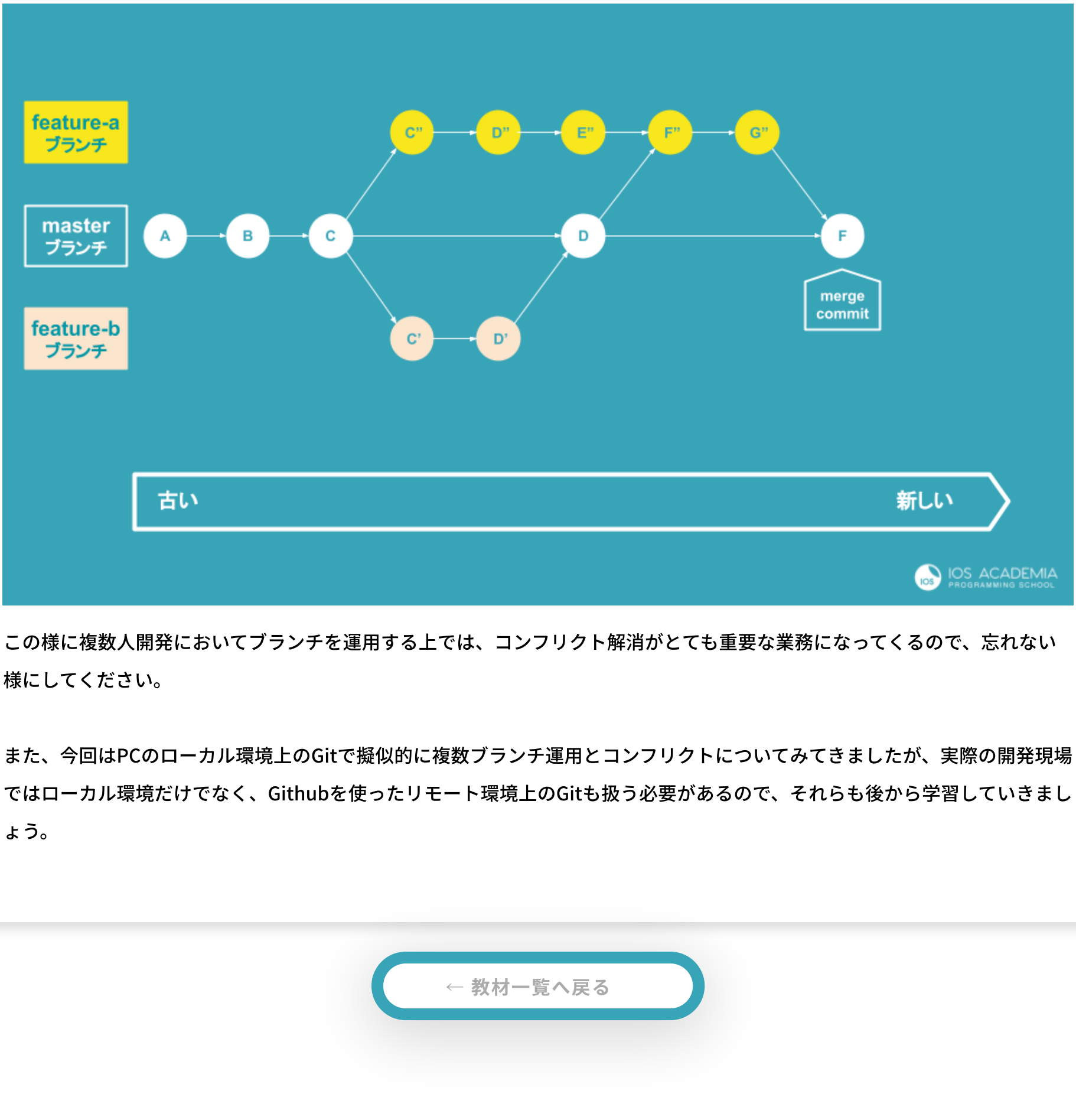
```
$ git log
```

```
output
```

```
commit 2320cfc5875e07a50d301a79c4156d83536abe2c (HEAD -> master)
Merge: d15e162 c96982e
Author: yamataku29
Date:   Wed Apr 14 10:32:58 2021 +0900

    Merge branch 'feature-a'

commit c96982e66f7b87d75fe94b0ac7ce90bf26391b88 (feature-a)
Merge: f4a8a4c d15e162
Author: yamataku29
Date:   Wed Apr 14 10:31:04 2021 +0900

    masterをマージしてコンフリクト解消

commit f4a8a4caefd7eecb3fb3f209ebebeb10554c6f97
Author: yamataku29
Date:   Wed Apr 14 10:08:06 2021 +0900

    third.txtに文字列を追加

commit d15e16219c3df43ac3bdcae36ed79ad9d5ffd40d
Merge: 1c1db0c 15d28cd
Author: yamataku29
Date:   Wed Apr 14 09:56:19 2021 +0900

    Merge branch 'feature-b'

commit 15d28cd30b61bfbd1779d4ff77cc5399fa9ac50c (feature-b)
Author: yamataku29
Date:   Wed Apr 14 09:52:04 2021 +0900

    third.txtに文字列を追加
```

これで正常にマージコミットも追加されていることが分かりました。

これまでのブランチ作成からマージまでの流れとしては、以下の様なイメージとなります。



この様に複数人開発においてブランチを運用する上では、コンフリクト解消がとても重要な業務になってくるので、忘れない様にしてください。

また、今回はPCのローカル環境上のGitで擬似的に複数ブランチ運用とコンフリクトについてみてきましたが、実際の開発現場ではローカル環境だけでなく、Githubを使ったリモート環境上のGitも扱う必要があるので、それらも後から学習していきましょう。

← 教材一覧へ戻る

受講申し込みはこちらから

まずは受講用アカウントの作成からスタート。
iOSアカデミアの受講に必要な各種情報を記載した、ご案内メールをお届けします。

受講申し込み ▸

IOS ACADEMIA PROGRAMMING SCHOOL

ホーム お知らせ 公式ブログ お問い合わせ ログイン 無料個別相談

利用規約 プライバシーポリシー 運営会社概要