Git 入門

話すことと話さないこと

話すこと

- ローカルリポジトリでの作業の流れ
- ブランチとマージ
- リモートリポジトリを使う作業の流れ

話さないこと

- Git が提供するさまざまなコマンドの解説
- GUI クライアントや IDE のプラグインの使い方
- Git の開発モデル (git-flow とか)
- その他諸々

このスライドの内容は git 1.8.3.1 で確認しています

ローカルリポジトリ

～個人用バージョン管理システムとして～

リポジトリを作成する

git init

```
$ pwd
/home/y-uti/proj

$ git init
Initialized empty Git repository in /home/y-uti/proj/.git/

$ ls -a
./  ../  .git/
```

- 管理ディレクトリが作成される

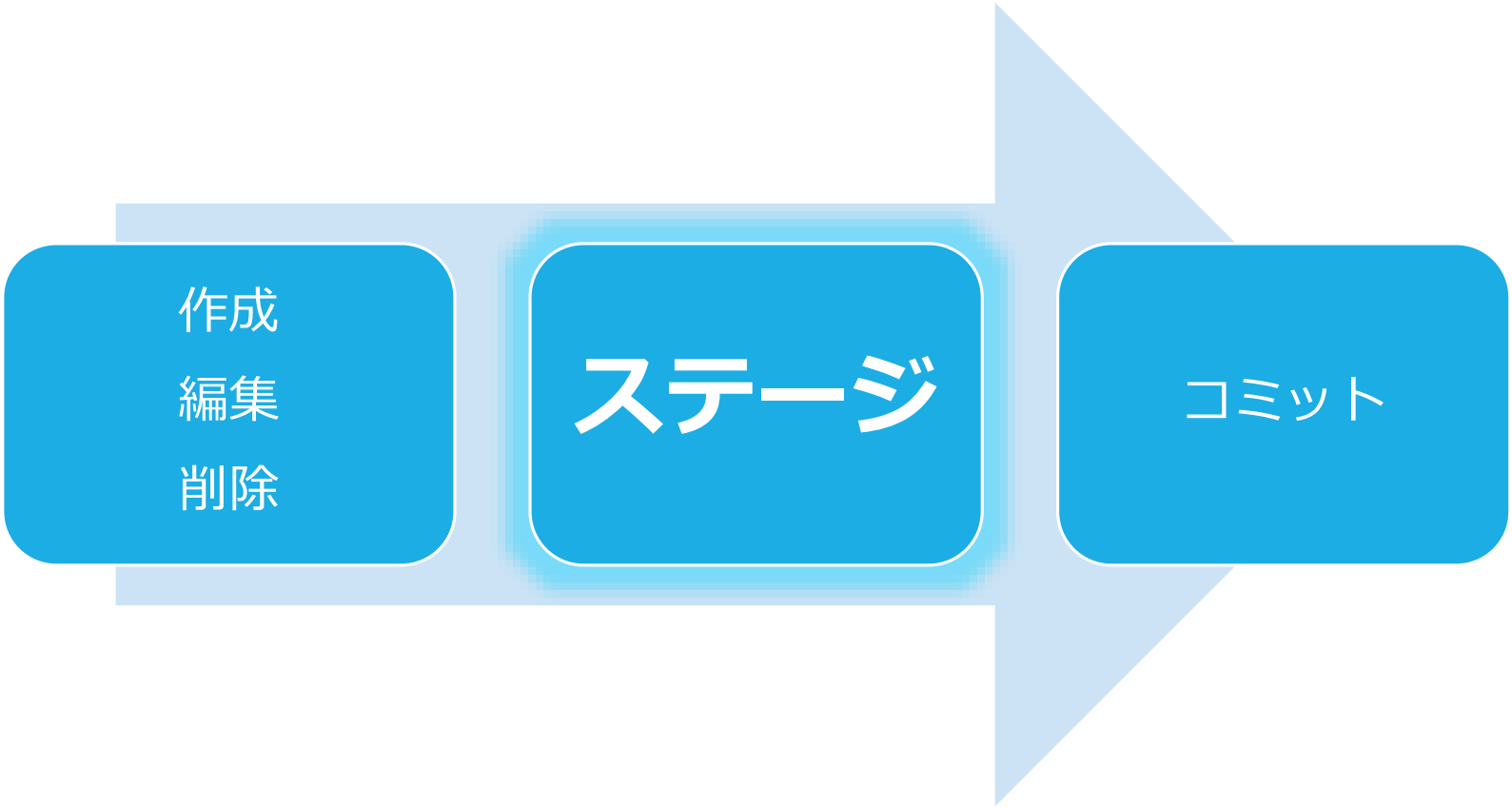
ユーザ情報を登録する

git config

```
$ git config user.email 'y-uti@mycompany.com'
$ git config user.name 'y-uti'

$ git config -l
core.repositoryformatversion=0
core.filemode=true
core.bare=false
core.logallrefupdates=true
user.email=y-uti@mycompany.com
user.name=y-uti
```

- 設定しておかないとコミットするときに怒られる

Git の作業フロー



ファイルを作成する

好きなエディタを開いて自由にファイルを作成する

```
$ vi README
Git 入門

$ ls -a
./  ../  .git/  README
```

状態を確認する

git status

```
$ git status
# On branch master
#
# Initial commit
#
# Untracked files:
#   (use "git add <file>..." to include in what will be
committed)
#
#       README
nothing added to commit but untracked files present (use
"git add" to track)
```

ステージする

git add

```
$ git add README

$ git status
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#       new file:   README
#
```

コミットする

git commit

```
$ git commit -m 'README を作成'
[master (root-commit) 1fd89c5] README を作成
 1 file changed, 1 insertion(+)
 create mode 100644 README

$ git status
# On branch master
nothing to commit, working directory clean
```

ファイルを編集する

好きなエディタを開いて自由にファイルを編集する

```
$ vi README
Git 入門
Git はとても難しい
```

状態を確認する

git status

```
$ git status
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be
committed)
#   (use "git checkout -- <file>..." to discard changes in
working directory)
#
#       modified:   README
#
no changes added to commit (use "git add" and/or "git
commit -a")
```

- ここで git commit しても変更はコミットされない

ステージする

git add

```
$ git add README

$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       modified:   README
#
```

- ここで git commit すれば変更がコミットされる
- が、次のスライドでちょっと寄り道してみましょう

さらにファイルを編集する

好きなエディタを開いて自由にファイルを編集する

```
$ vi README
Git 入門
Git はとても難しい
Git は難しすぎる
```

状態を確認する

git status

```
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       modified:   README
#
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working
directory)
#
#       modified:   README
#
```

- 二箇所に出てくる

今どうなっているのか

作業ディレクトリ	ステージング領域	リポジトリ（の最新）
Git 入門 Git はとても難しい Git は難しすぎる	Git 入門 Git はとても難しい	Git 入門

差分を確認する

From: ステージング領域
To: 作業ディレクトリ

作業ディレクトリ	ステージング領域	リポジトリ（の最新）
Git 入門 Git はとても難しい Git は難しすぎる	Git 入門 Git はとても難しい	Git 入門

```
$ git diff
diff --git a/README b/README
index a1247d5..7ffac0f 100644
--- a/README
+++ b/README
@@ -1,2 +1,3 @@
 Git 入門
 Git はとても難しい
+Git は難しすぎる
```

差分を確認する

From: リポジトリの最新
To: 作業ディレクトリ

作業ディレクトリ	ステージング領域	リポジトリ（の最新）
Git 入門 Git はとても難しい Git は難しすぎる	Git 入門 Git はとても難しい	Git 入門

```
$ git diff HEAD
diff --git a/README b/README
index 4dfa228..7ffac0f 100644
--- a/README
+++ b/README
@@ -1 +1,3 @@
 Git 入門
+Git はとても難しい
+Git は難しすぎる
```

差分を確認する

From: リポジトリの最新
To: ステージング領域

作業ディレクトリ	ステージング領域	リポジトリ（の最新）
Git 入門 Git はとても難しい Git は難しすぎる	Git 入門 Git はとても難しい	Git 入門

```
$ git diff --cached
diff --git a/README b/README
index 4dfa228..a1247d5 100644
--- a/README
+++ b/README
@@ -1 +1,2 @@
 Git 入門
+Git はとても難しい
```

コミットする

git commit

```
$ git commit -m 'README を編集'
[master ac1ee4a] README を編集
 1 file changed, 1 insertion(+)

$ git status
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working
directory)
#
#       modified:   README
#
no changes added to commit (use "git add" and/or "git commit -a")
```

- "nothing to commit, working directory clean" にはならない

何が起きたのか

コミット前

作業ディレクトリ	ステージング領域	リポジトリ（の最新）
Git 入門 Git はとても難しい Git は難しすぎる	Git 入門 Git はとても難しい	Git 入門

コミット後

作業ディレクトリ	ステージング領域	リポジトリ（の最新）
Git 入門 Git はとても難しい Git は難しすぎる	Git 入門 Git はとても難しい	Git 入門 Git はとても難しい

- git commit はステージング領域をリポジトリに反映するだけ

大切な余談

ステージを取り消す

作業ディレクトリ	ステージング領域	リポジトリ（の最新）
Git 入門 Git はとても難しい Git は難しすぎる	Git 入門 Git はとても難しい	Git 入門

```
$ git reset README
Unstaged changes after reset:
M       README
```

作業ディレクトリ	ステージング領域	リポジトリ（の最新）
Git 入門 Git はとても難しい Git は難しすぎる	Git 入門	Git 入門

- リポジトリの最新の内容をステージング領域に取得

大切な余談

未ステージの変更を取消す

作業ディレクトリ	ステージング領域	リポジトリ（の最新）
Git 入門 Git はとても難しい Git は難しすぎる	Git 入門 Git はとても難しい	Git 入門

```
$ git checkout -- README
```

作業ディレクトリ	ステージング領域	リポジトリ（の最新）
Git 入門 Git はとても難しい	Git 入門 Git はとても難しい	Git 入門

- ステージング領域の内容を作業ディレクトリに取得

ブランチとマージ

～コミットで作られるグラフ構造を理解する～

その前に

作業ディレクトリを掃除しておきます

```
$ git reset --hard
HEAD is now at ac1ee4a README を編集
```

- git reset --hard = git reset + git checkout
  - リポジトリの最新をステージング領域に取得
  - ステージング領域を作業ディレクトリに取得

それぞれの内容はこのようになっている（としましょう）

作業ディレクトリ	ステージング領域	リポジトリ（の最新）
Git 入門 Git はとても難しい	Git 入門 Git はとても難しい	Git 入門 Git はとても難しい

コミットの履歴を確認する

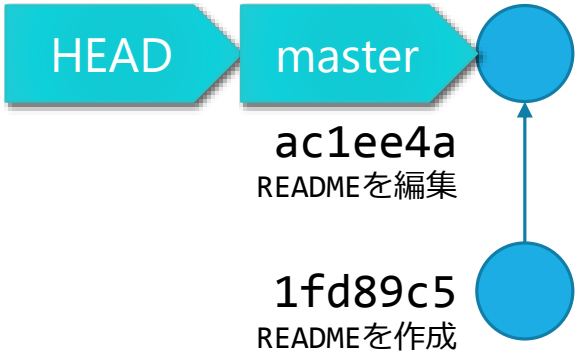
git log

```
$ git log
commit ac1ee4a27ec36764178ef73f2490270e87610cc6
Author: y-uti <y-uti@mycompany.com>
Date:   Sat Nov 1 15:08:44 2014 +0900

    README を編集

commit 1fd89c51a0946827c77c357dfe3287e2e96af875
Author: y-uti <y-uti@mycompany.com>
Date:   Sat Nov 1 13:50:48 2014 +0900

    README を作成
```

コミットの履歴を確認する

git log --graph --oneline --decorate=full

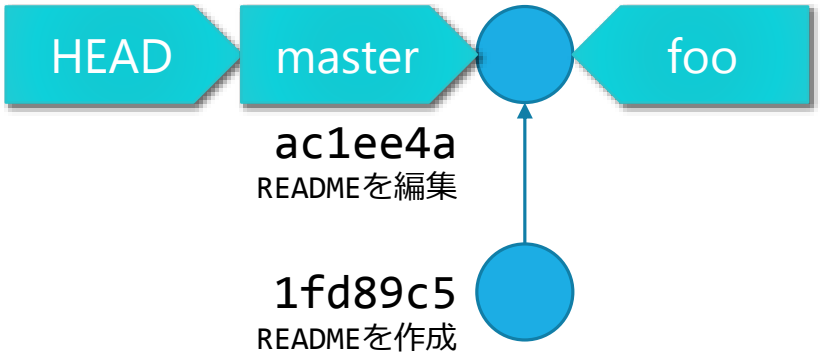
```
$ git log --graph --oneline --decorate=full
* ac1ee4a (HEAD, refs/heads/master) README を編集
* 1fd89c5 README を作成
```



ブランチを作成する

git branch

```
$ git branch foo

$ git log --graph --oneline --decorate=full
* ac1ee4a (HEAD, refs/heads/master, refs/heads/foo) README
を編集
* 1fd89c5 README を作成
```

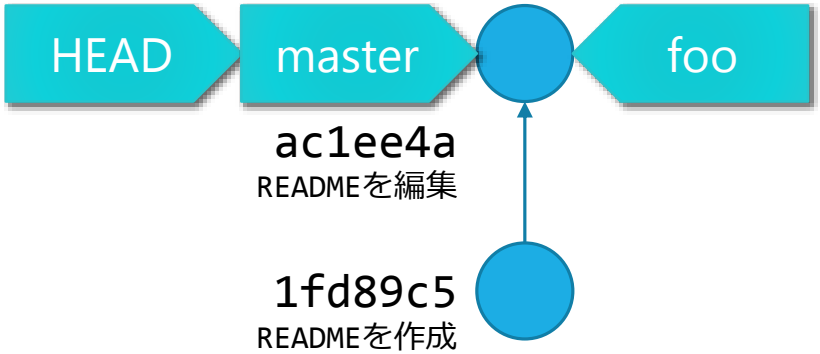


ブランチを一覧表示する

git branch

```
$ git branch
  foo
* master
```

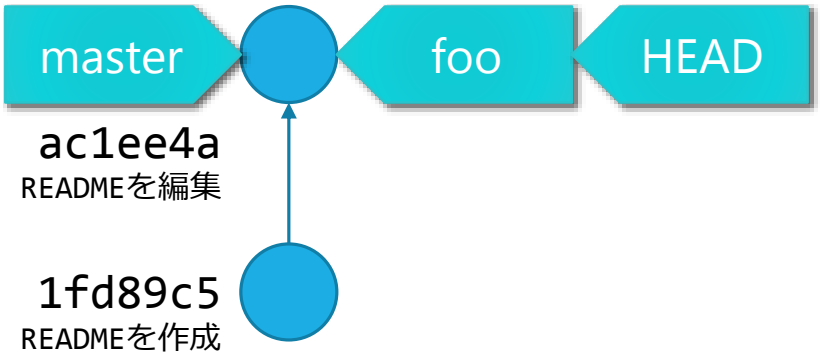
- ブランチを作成しただけでは現在のブランチは切り替わらない



ブランチを切り替える

git checkout

```
$ git checkout foo
Switched to branch 'foo'

$ git branch
* foo
  master
```



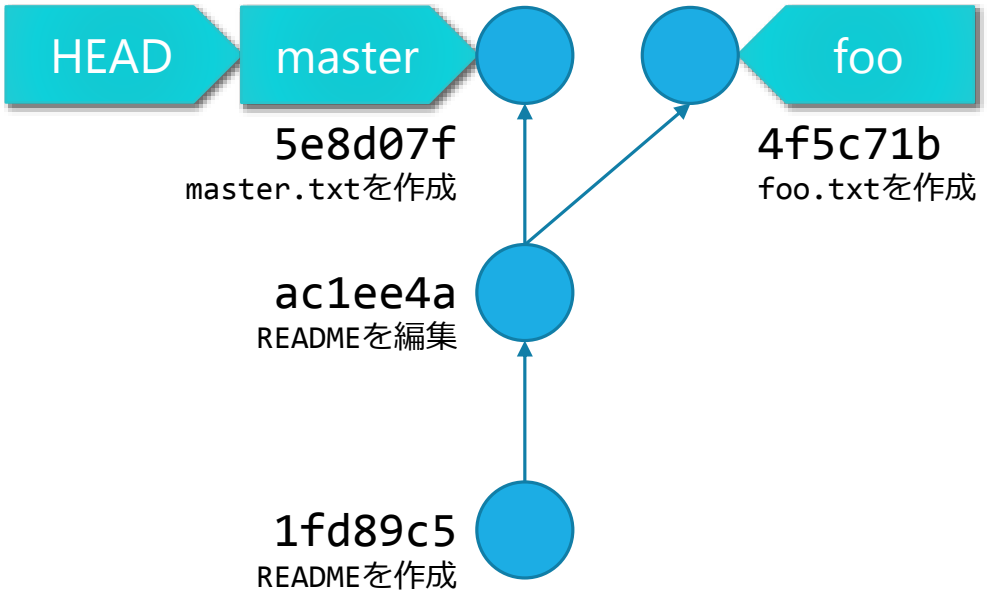
ブランチで作業する

各ブランチで適当に作業してコミットする

```
$ echo foo >foo.txt
$ git add foo.txt
$ git commit -m 'foo.txt を作成'

$ git checkout master
$ ls
README

$ echo master >master.txt
$ git add master.txt
$ git commit -m 'master.txt を作成'
```

- master ブランチには foo.txt が存在していないことに注目

今どうなっているのか


HEAD
master
foo
5e8d07f
master.txtを作成
4f5c71b
foo.txtを作成
ac1ee4a
READMEを編集
1fd89c5
READMEを作成

ブランチをマージする

git merge

```
$ git merge foo
```

```
Merge branch 'foo'

# 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.
```

- エディタでコミットメッセージを入力するとマージが実行される

```
Merge made by the 'recursive' strategy.
 foo.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 foo.txt
```

マージ結果を確認する

git log を使う

```
$ ls
README  foo.txt  master.txt

$ git log --graph --oneline --decorate=full
*   cbf2105 (HEAD, refs/heads/master) Merge branch 'foo'
|¥
| * 4f5c71b (refs/heads/foo) foo.txt を作成
* | 5e8d07f master.txt を作成
|/
* ac1ee4a README を編集
* 1fd89c5 README を作成
```

- マージも一つのコミット

衝突

各ブランチで適当にファイルを編集して衝突させる

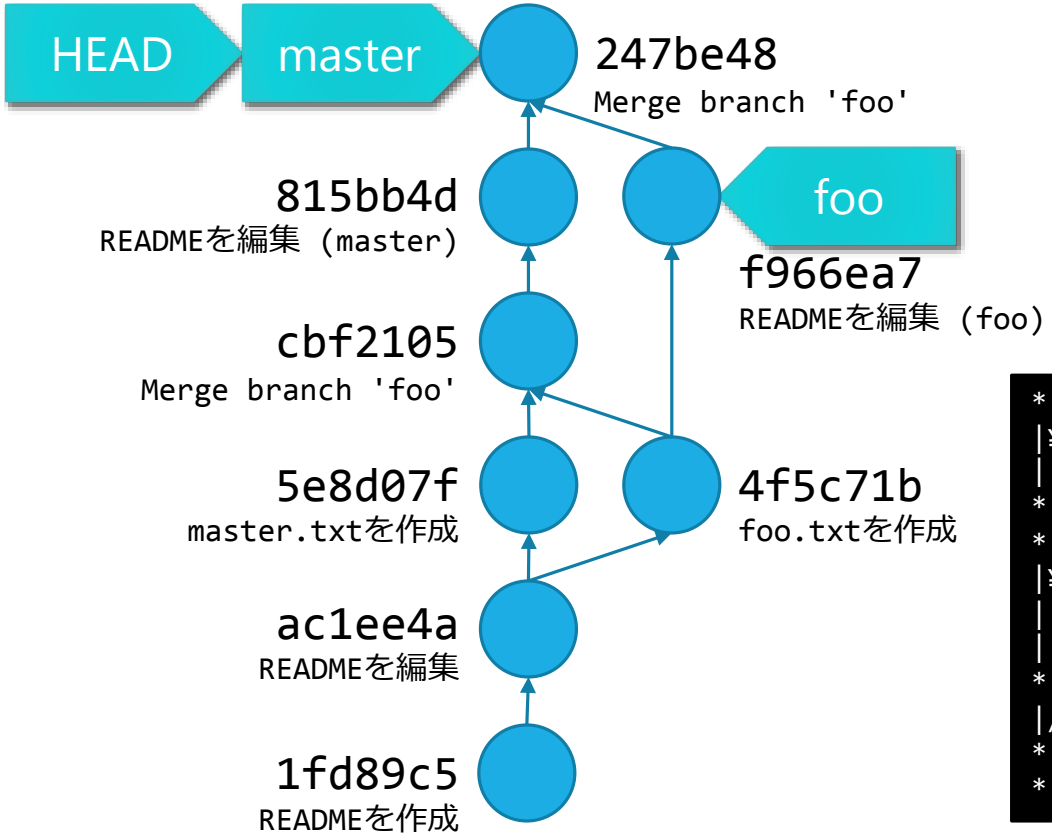
```
$ git checkout foo
$ echo foo >>README
$ git commit -am 'README を編集 (foo)'

$ git checkout master
$ echo master >>README
$ git commit -am 'README を編集 (master)'

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

衝突を解決する

衝突箇所を編集してから git add して git commit する

```
$ cat README
Git 入門
Git はとても難しい
<<<<<<< HEAD
master
=======
foo
>>>>>>> foo

$ vi README
$ git add README
$ git commit
[master 247be48] Merge branch 'foo'
```

今どうなっているのか




```
*   247be48 Merge branch 'foo'
|¥
| * f966ea7 README を編集 (foo)
* | 815bb4d README を編集 (master)
* |   cbf2105 Merge branch 'foo'
|¥ ¥
| |/
| * 4f5c71b foo.txt を作成
* | 5e8d07f master.txt を作成
|/
* ac1ee4a README を編集
* 1fd89c5 README を作成
```

大切な余談

衝突の解決を諦めて戻す

衝突する予定のなかった git merge で衝突して途方に暮れたら

```
$ git reset --hard
HEAD is now at 815bb4d README を編集 (master)
```

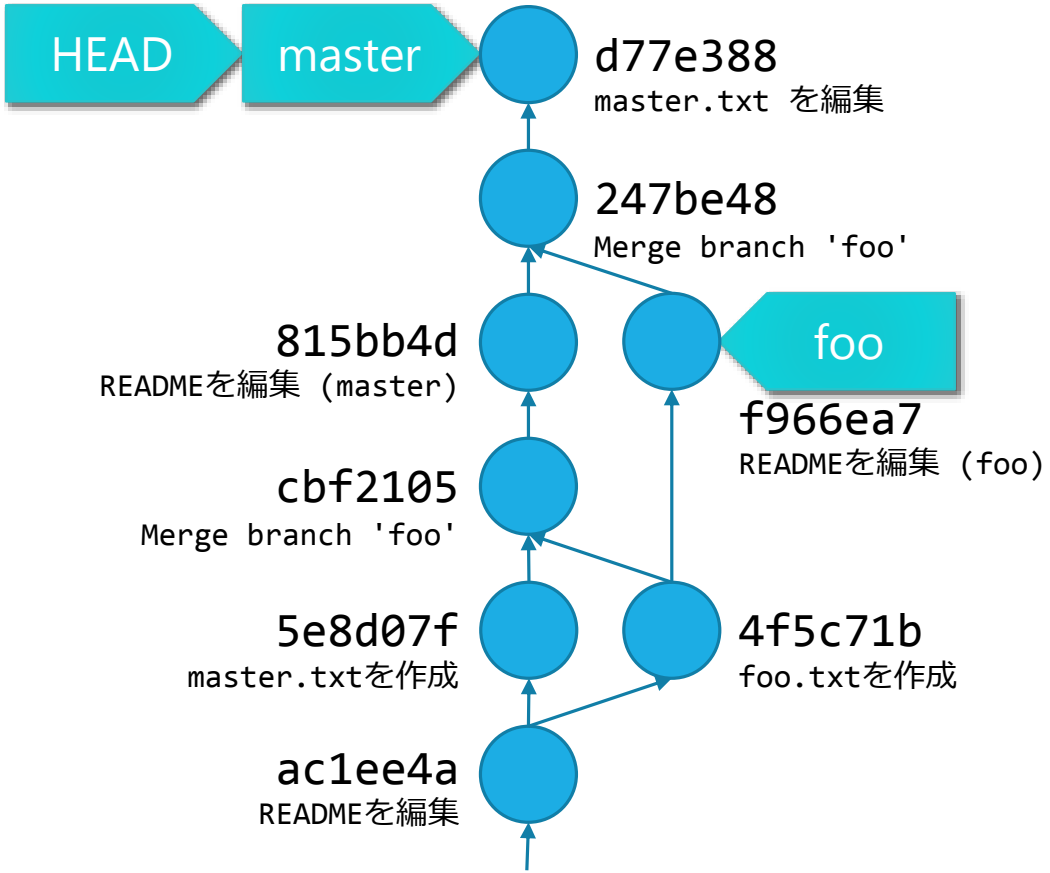
- 明日やることにして帰って寝る

fast-forward マージ

引き続き master ブランチでファイルを編集する

```
$ echo edited >>master.txt
$ git commit -am 'master.txt を編集'
[master d77e388] master.txt を編集
 1 file changed, 1 insertion(+)
```

今どうなっているのか


HEAD
master
d77e388
master.txt を編集
247be48
Merge branch 'foo'
815bb4d
READMEを編集 (master)
foo
f966ea7
READMEを編集 (foo)
cbf2105
Merge branch 'foo'
5e8d07f
master.txtを作成
4f5c71b
foo.txtを作成
ac1ee4a
READMEを編集

fast-forward マージ

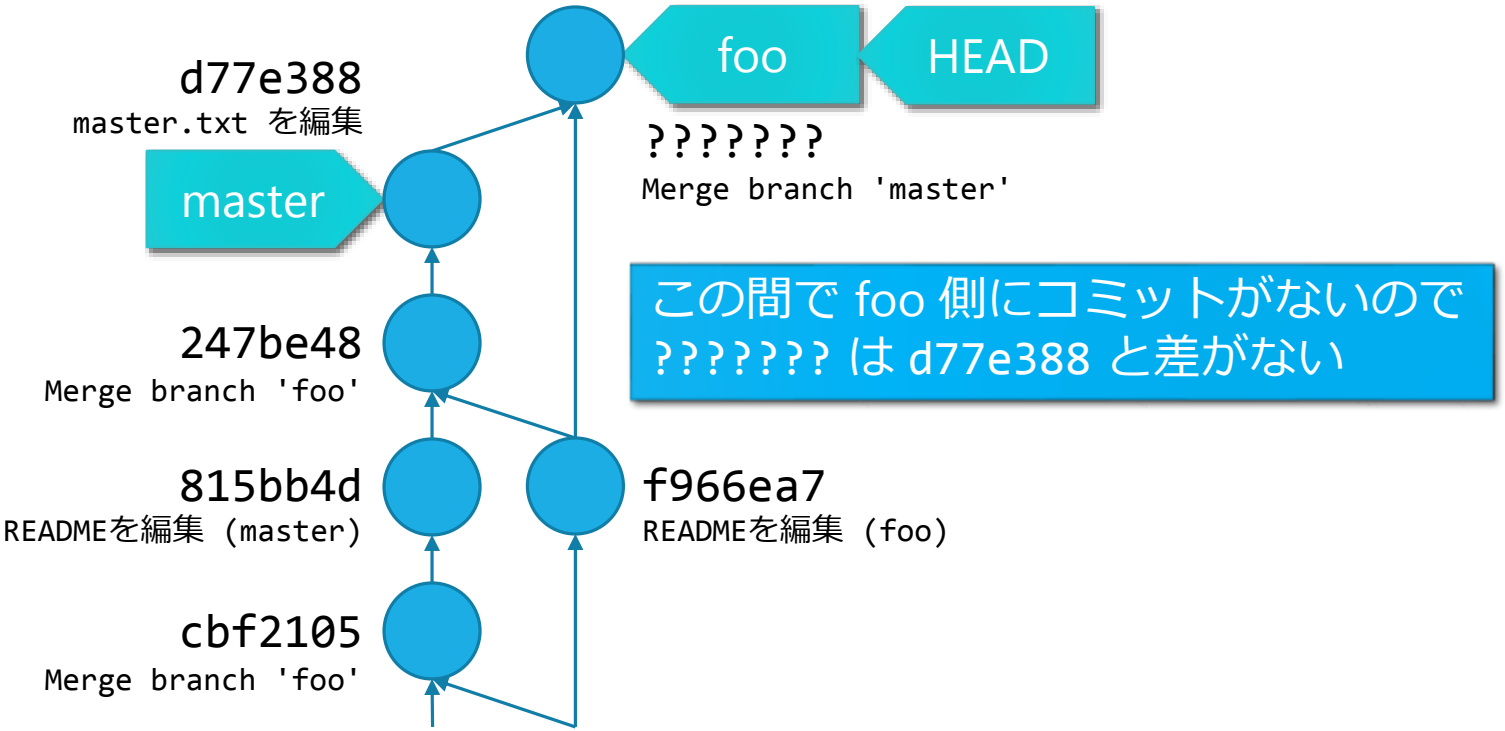
master の変更を foo ブランチにマージする

- 今までと逆

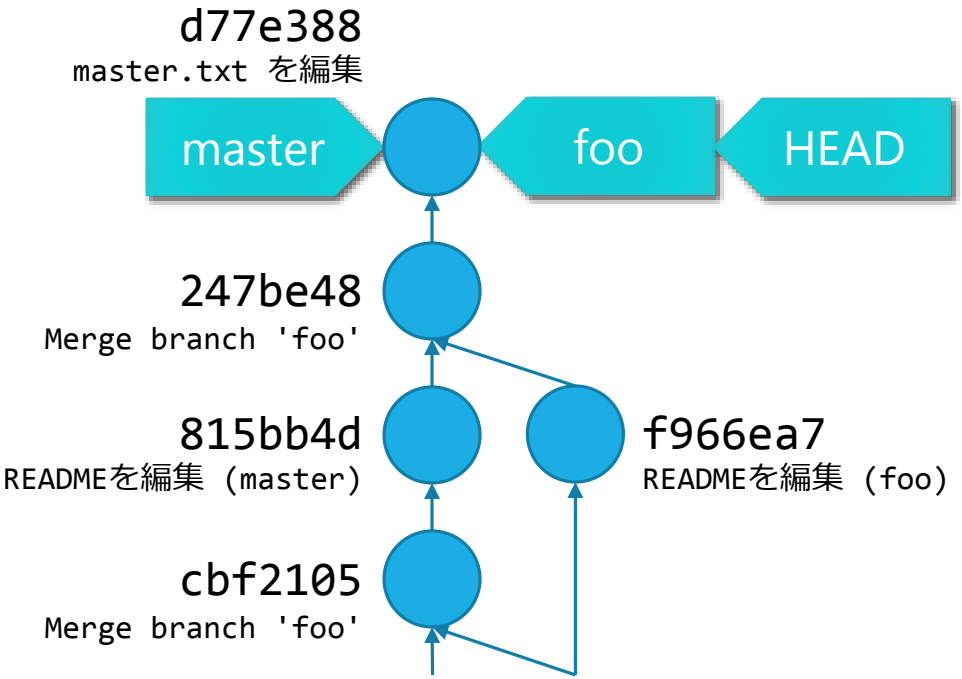
```
$ git checkout foo
Switched to branch 'foo'

$ git merge master
Updating f966ea7..d77e388
Fast-forward
 README     | 1 +
 master.txt | 2 ++
 2 files changed, 3 insertions(+)
 create mode 100644 master.txt
```

- コミットメッセージの入力も無いまま勝手にマージが完了する

このようにはならない


foo
HEAD
d77e388
master.txt を編集
master
???????
Merge branch 'master'
この間で foo 側にコミットがないので
??????? は d77e388 と差がない
247be48
Merge branch 'foo'
815bb4d
README を編集 (master)
f966ea7
README を編集 (foo)
cbf2105
Merge branch 'foo'

こうなる


d77e388
master.txt を編集
master
foo
HEAD
247be48
Merge branch 'foo'
815bb4d
READMEを編集 (master)
f966ea7
READMEを編集 (foo)
cbf2105
Merge branch 'foo'

リモートリポジトリ

～みんなで使う分散バージョン管理システム～

その前に

今まで作業したリポジトリを共有しておきます

```
$ pwd
/home/y-uti

$ git clone --bare proj proj.git
$ cd proj.git
$ git remote remove origin
```

- /home/y-uti/proj.git にリモートリポジトリが作成された
- 中身は今まで作業してきたローカルリポジトリのコピー

開発者 Bob がいます

ということにしましょう

```
$ mkdir ~/bob
```

Bob は開発を進めるためリモートリポジトリを取得します

```
$ cd ~/bob
$ git clone ~/proj.git
Cloning into 'proj'...
done.

$ git config user.email 'bob@mycompany.com'
$ git config user.name Bob
$ git config push.default simple
```

- 最後の一行はオマジナイ (書かないと後で文句を言われる)

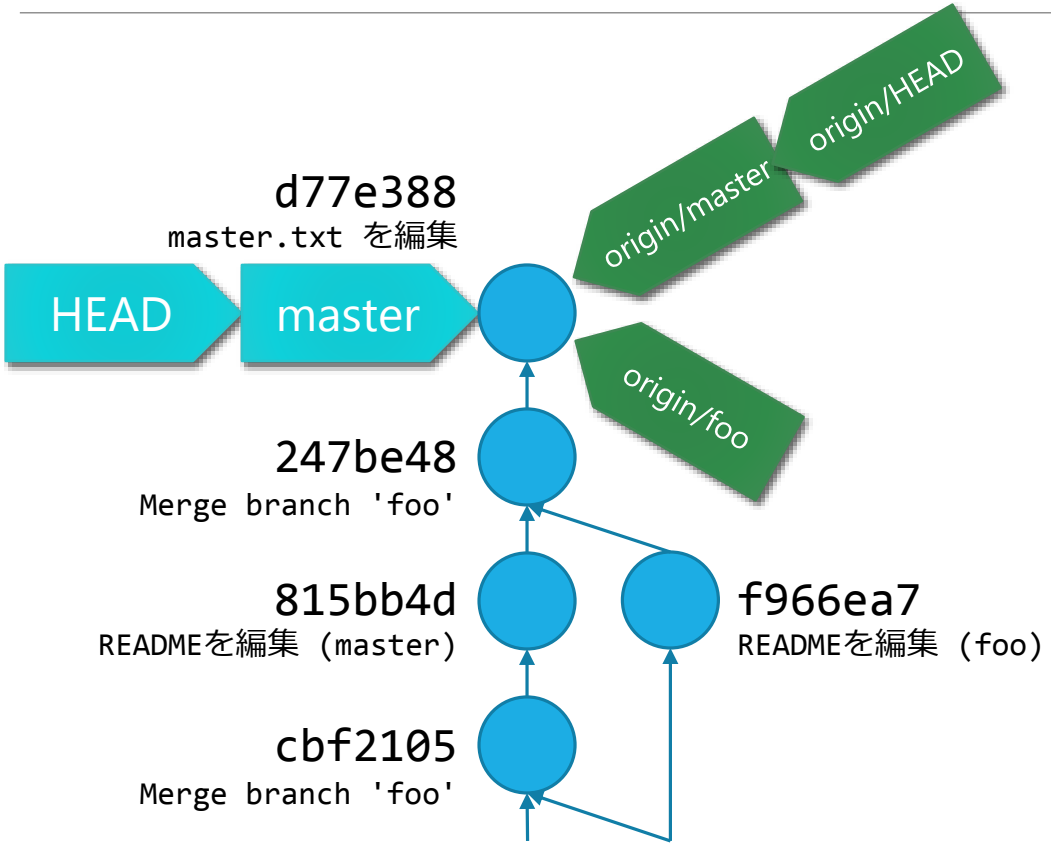
状況の確認

Bob のローカルリポジトリを確認する

```
$ cd ~/bob/proj

$ git log --graph --oneline --decorate=full --all
* d77e388 (HEAD, refs/remotes/origin/master,
refs/remotes/origin/foo, refs/remotes/origin/HEAD,
refs/heads/master) master.txt を編集
*   247be48 Merge branch 'foo'

... (以下省略)
```

- --all を付けるとリモートブランチも表示される

今どうなっているのか

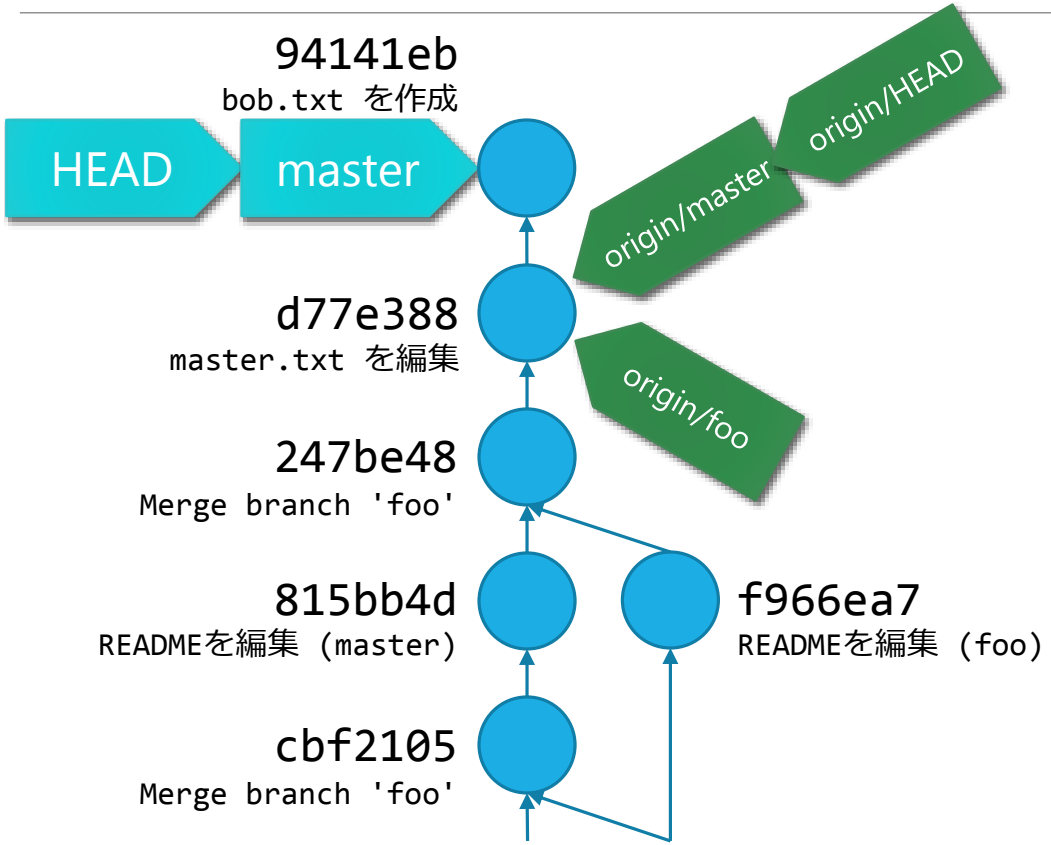

origin/HEAD
origin/master
d77e388
master.txt を編集
HEAD
master
origin/foo
247be48
Merge branch 'foo'
815bb4d
README を編集 (master)
f966ea7
README を編集 (foo)
cbf2105
Merge branch 'foo'

作業をコミットする

ローカルリポジトリで適当にファイルを編集する

```
$ echo Bob >bob.txt
$ git add bob.txt
$ git commit -m 'bob.txt を作成'
[master 94141eb] bob.txt を作成
 1 file changed, 1 insertion(+)
 create mode 100644 bob.txt

$ git status
# On branch master
# Your branch is ahead of 'origin/master' by 1 commit.
#   (use "git push" to publish your local commits)
#
nothing to commit, working directory clean
```

今どうなっているのか

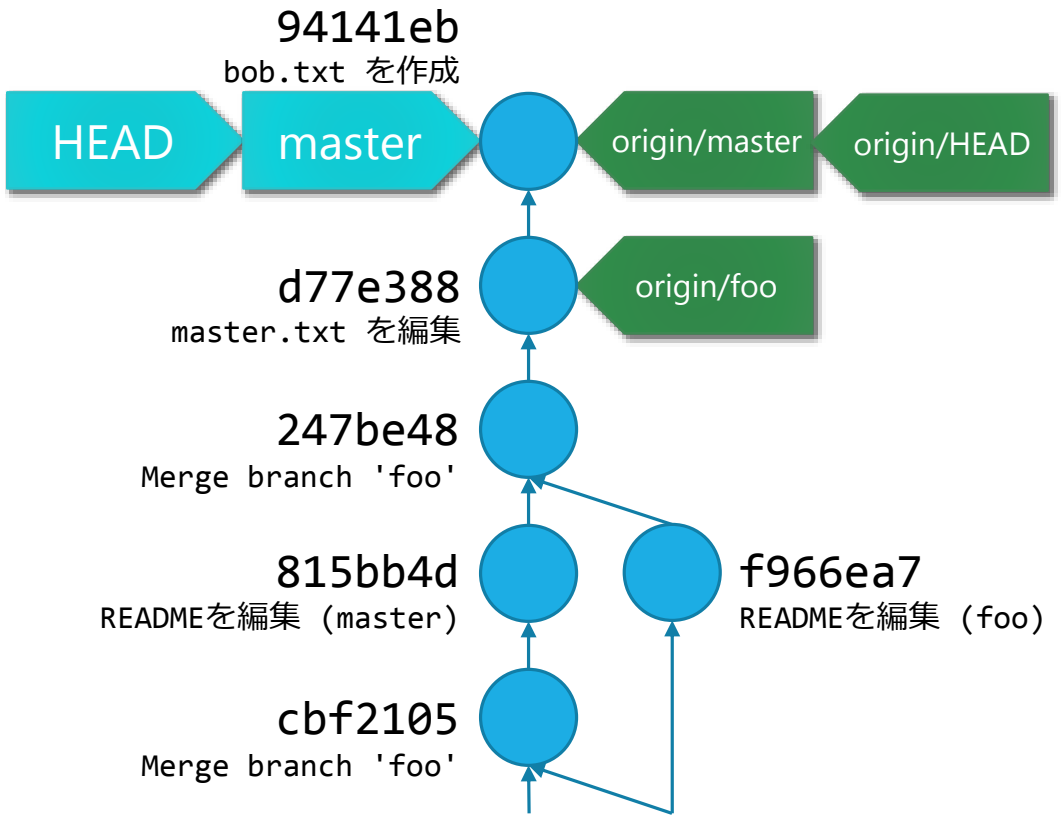

94141eb
bob.txt を作成
HEAD
master
origin/HEAD
origin/master
d77e388
master.txt を編集
origin/foo
247be48
Merge branch 'foo'
815bb4d
README を編集 (master)
f966ea7
README を編集 (foo)
cbf2105
Merge branch 'foo'

リモートリポジトリに反映する

git push

```
$ git push
Counting objects: 4, done.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 343 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To /home/y-uti/proj.git
   d77e388..94141eb  master -> master

# On branch master
nothing to commit, working directory clean
```

今どうなっているのか


94141eb
bob.txt を作成
HEAD
master
origin/master
origin/HEAD
d77e388
master.txt を編集
origin/foo
247be48
Merge branch 'foo'
815bb4d
READMEを編集 (master)
f966ea7
READMEを編集 (foo)
cbf2105
Merge branch 'foo'

Alice 登場

Alice も proj の開発者だとしましょう

```
$ mkdir ~/alice
```

Alice も開発を進めるためリモートリポジトリを取得します

```
$ cd ~/alice
$ git clone ~/proj.git
Cloning into 'proj'...
done.

$ git config user.email 'alice@mycompany.com'
$ git config user.name Alice
$ git config push.default simple
```

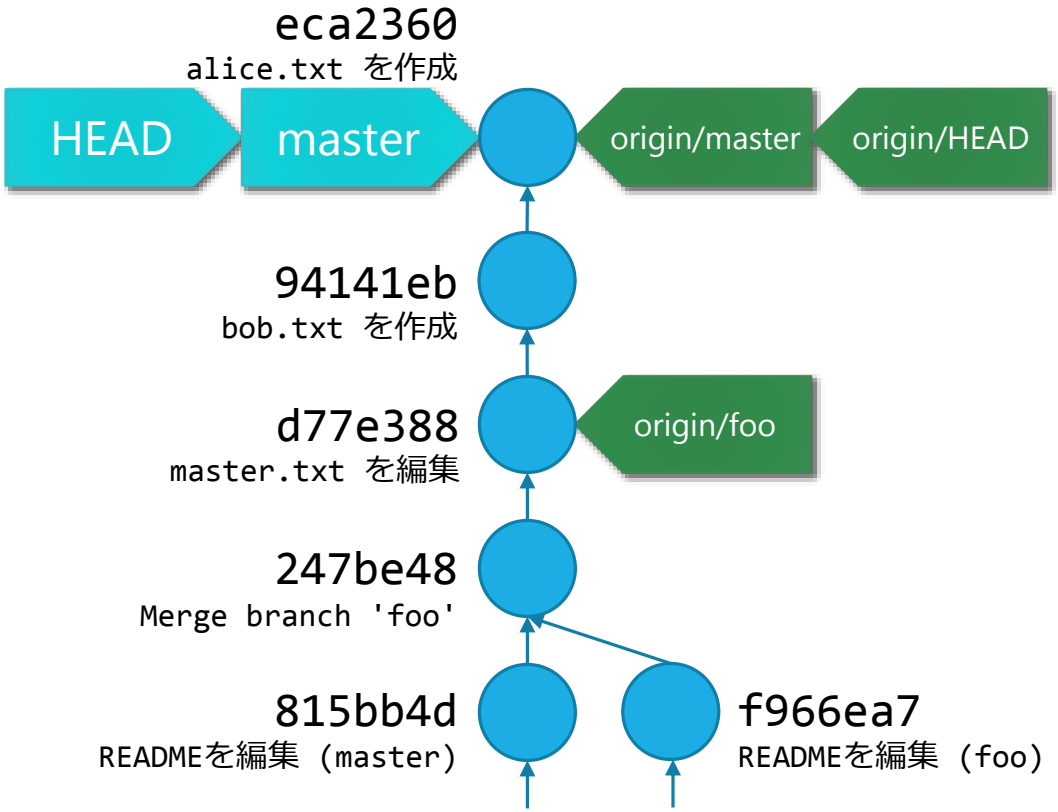
- Alice は Bob の push 後に clone したので直前のスライドの状態

Alice が作業する

ローカルリポジトリで適当にファイルを編集して push する

```
$ echo Alice >alice.txt
$ git add alice.txt
$ git commit -m 'alice.txt を作成'
[master eca2360] alice.txt を作成
 1 file changed, 1 insertion(+)
 create mode 100644 alice.txt
$ git push
Counting objects: 4, done.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 284 bytes | 0 bytes/s, done.
Total 3 (delta 1), reused 0 (delta 0)
To /home/y-uti/proj.git
   94141eb..eca2360  master -> master
```

今どうなっているのか


eca2360
alice.txt を作成
HEAD
master
origin/master
origin/HEAD
94141eb
bob.txt を作成
d77e388
master.txt を編集
origin/foo
247be48
Merge branch 'foo'
815bb4d
READMEを編集 (master)
f966ea7
READMEを編集 (foo)

その頃 Bob は・・・

Alice と並行して黙々と作業を続ける

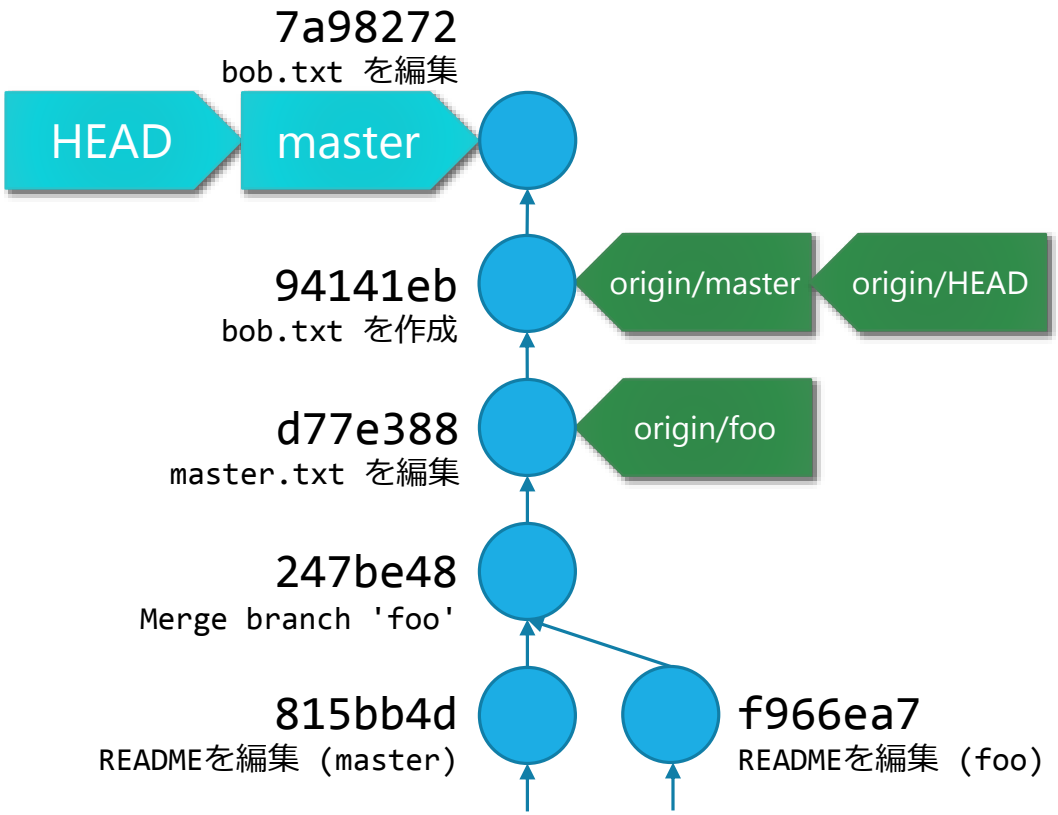
```
$ cd ~/bob/proj

$ echo 'edited' >>bob.txt
$ git commit -am 'bob.txt を編集'
[master 7a98272] bob.txt を編集
 1 file changed, 1 insertion(+)
```

軽やかに push

そして失敗する

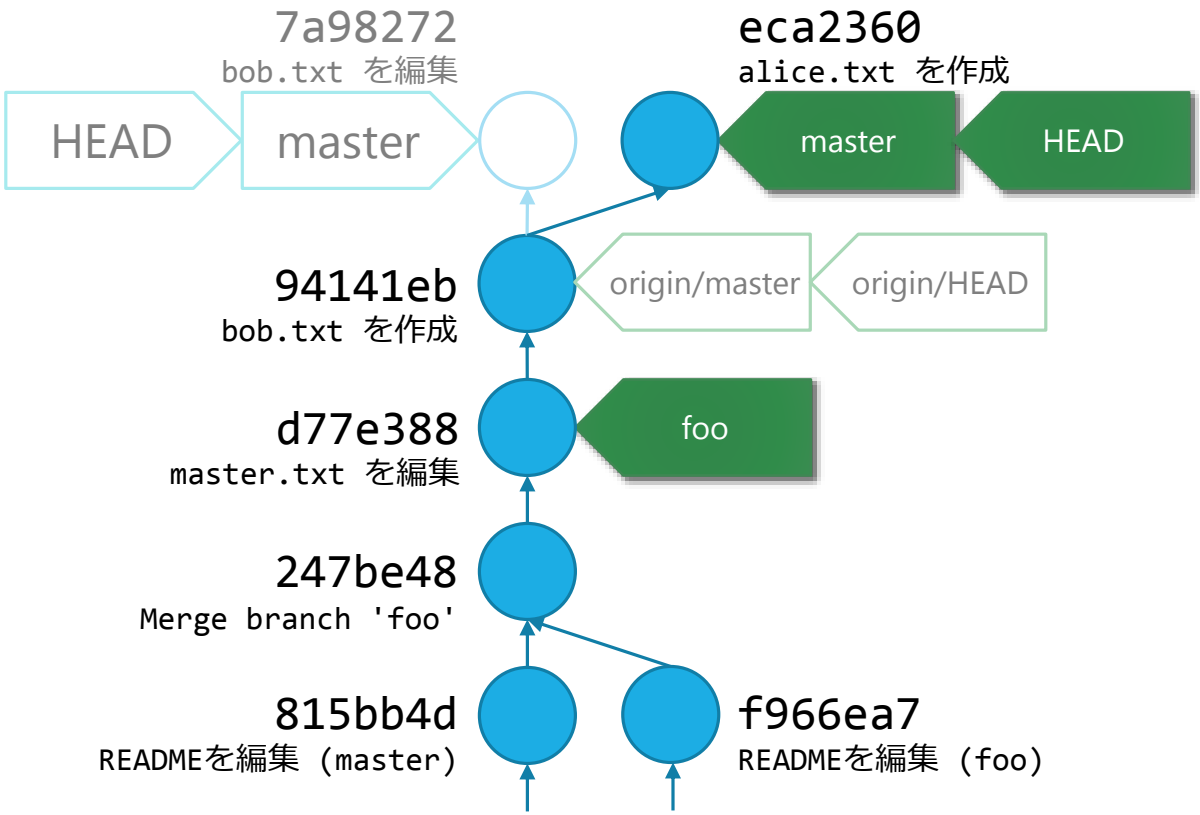
```
$ git push
To /home/y-uti/proj.git
 ! [rejected]        master -> master (fetch first)
error: failed to push some refs to '/home/y-uti/proj.git'
hint: Updates were rejected because the remote contains work that
you do
hint: not have locally. This is usually caused by another
repository pushing
hint: to the same ref. You may want to first merge the remote
changes (e.g.,
hint: 'git pull') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for
details.
```

今どうなっているのか


7a98272
bob.txt を編集
HEAD
master
94141eb
bob.txt を作成
origin/master
origin/HEAD
d77e388
master.txt を編集
origin/foo
247be48
Merge branch 'foo'
815bb4d
README を編集 (master)
f966ea7
README を編集 (foo)

```
$ cd ~/proj.git
$ git log --graph --oneline --decorate=full
```

今どうなっているのか

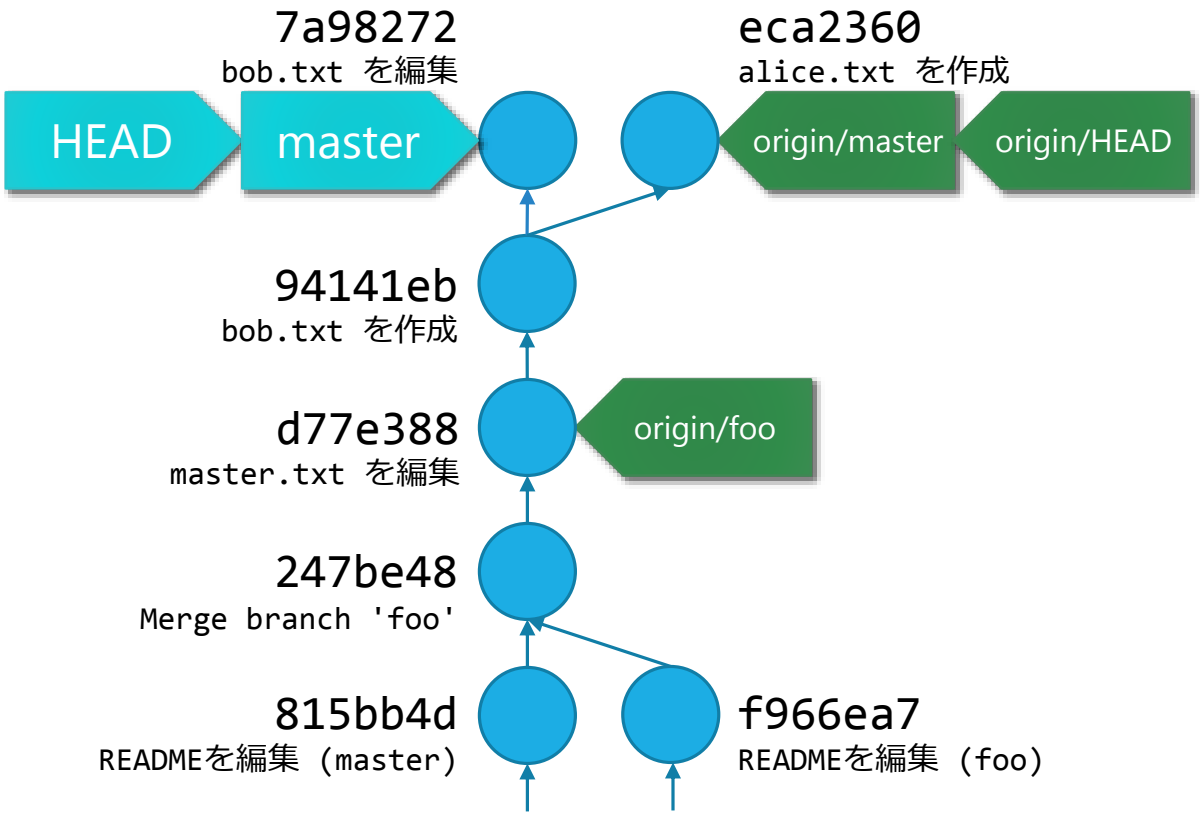


リモートリポジトリを取得する

git fetch

```
$ git fetch
remote: Counting objects: 4, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From /home/y-uti/proj
   94141eb..eca2360  master     -> origin/master
```

今どうなっているのか


7a98272
bob.txt を編集
eca2360
alice.txt を作成
HEAD
master
origin/master
origin/HEAD
94141eb
bob.txt を作成
d77e388
master.txt を編集
origin/foo
247be48
Merge branch 'foo'
815bb4d
README を編集 (master)
f966ea7
README を編集 (foo)

リモートブランチをマージする

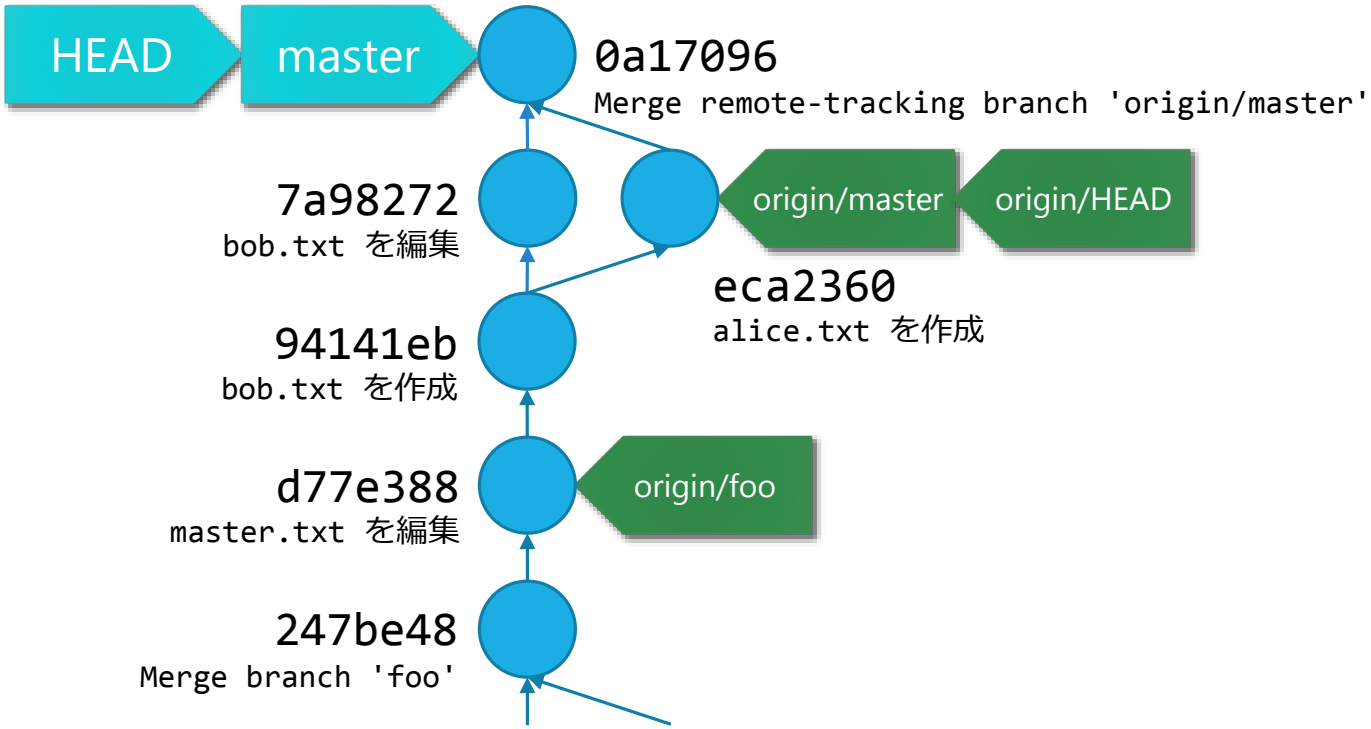
git merge

```
$ git merge origin/master
Merge made by the 'recursive' strategy.
 alice.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 alice.txt
```

- fast-forward マージにならないのでコミットメッセージを入力
- 変更が衝突した場合は修正して git add して git commit する

このスライドで Bob が実行した操作は実はあまり良くない
後で振り返ります

今どうなっているのか


HEAD
master
0a17096
Merge remote-tracking branch 'origin/master'
7a98272
bob.txt を編集
origin/master
origin/HEAD
eca2360
alice.txt を作成
94141eb
bob.txt を作成
d77e388
master.txt を編集
origin/foo
247be48
Merge branch 'foo'

改めて push する

今度は無事に成功する

```
$ git push
Counting objects: 8, done.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (5/5), 497 bytes | 0 bytes/s, done.
Total 5 (delta 2), reused 0 (delta 0)
To /home/y-uti/proj.git
   eca2360..0a17096  master -> master
```

- この変更を Alice は知らないので、次は Alice が push に失敗する
- これを繰り返すのが分散バージョン管理

大切な余談

ところで foo どこ行った

clone では master 以外のローカルブランチは用意されない

```
$ git branch -a
* master
  remotes/origin/HEAD -> origin/master
  remotes/origin/foo
  remotes/origin/master
```

git checkout するときに作成する

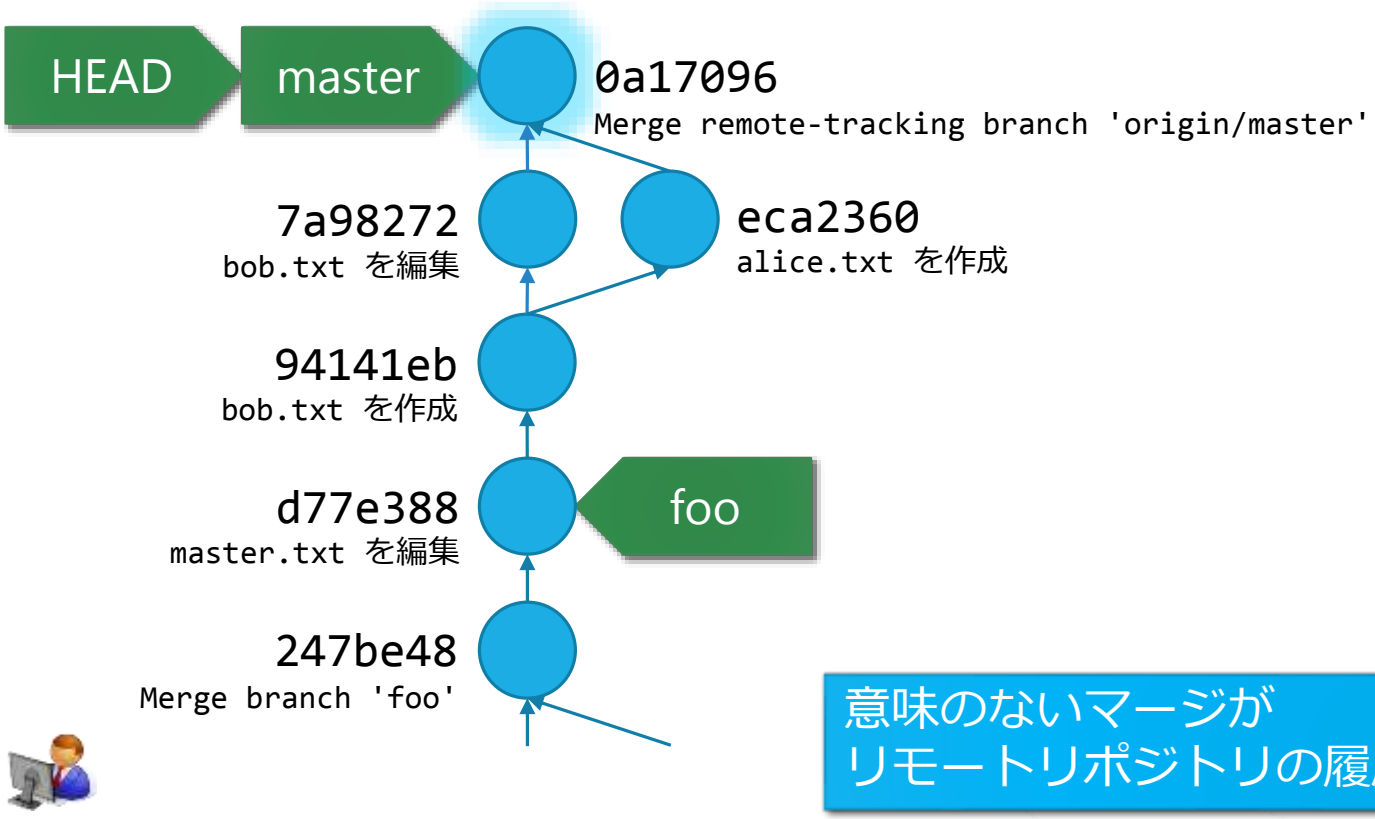
```
$ git checkout -b foo origin/foo
Branch foo set up to track remote branch foo from origin.
Switched to a new branch 'foo'
```

- -b オプションを指定する
- checkout なので同時に HEAD も foo を指すように切り替わる

Enjoy Git!

ちょっと待てコラ！

これを push した人は
正直に手を挙げてください



意味のないマージが
リモートリポジトリの履歴を汚す

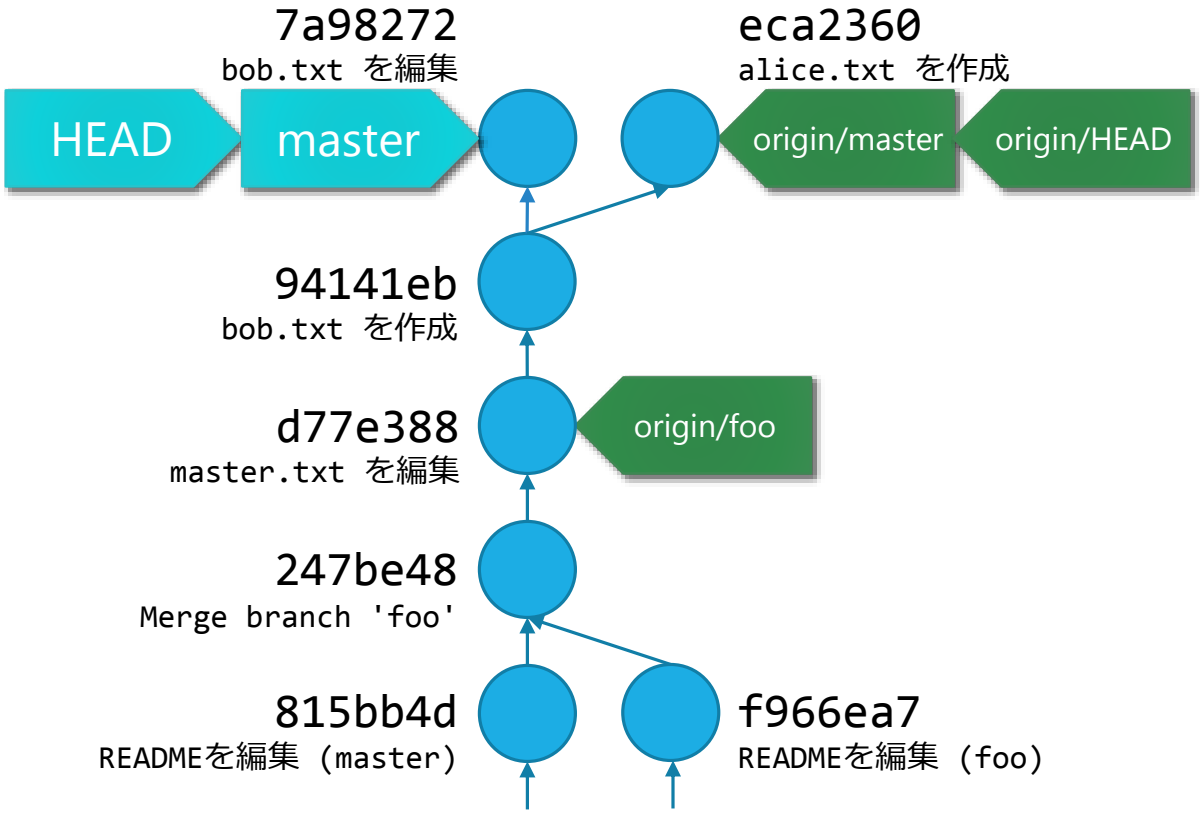
過去に戻る・・・

Bob が push に失敗して fetch してきた状態を再現する

```
$ cd ~/proj.git
$ git reset --soft eca2360

$ cd ~/bob/proj
$ git reset --hard 7a98272
$ git fetch
```

- 現実のプロジェクトで安易にこういう操作をしてはいけない

fetch してきた状態



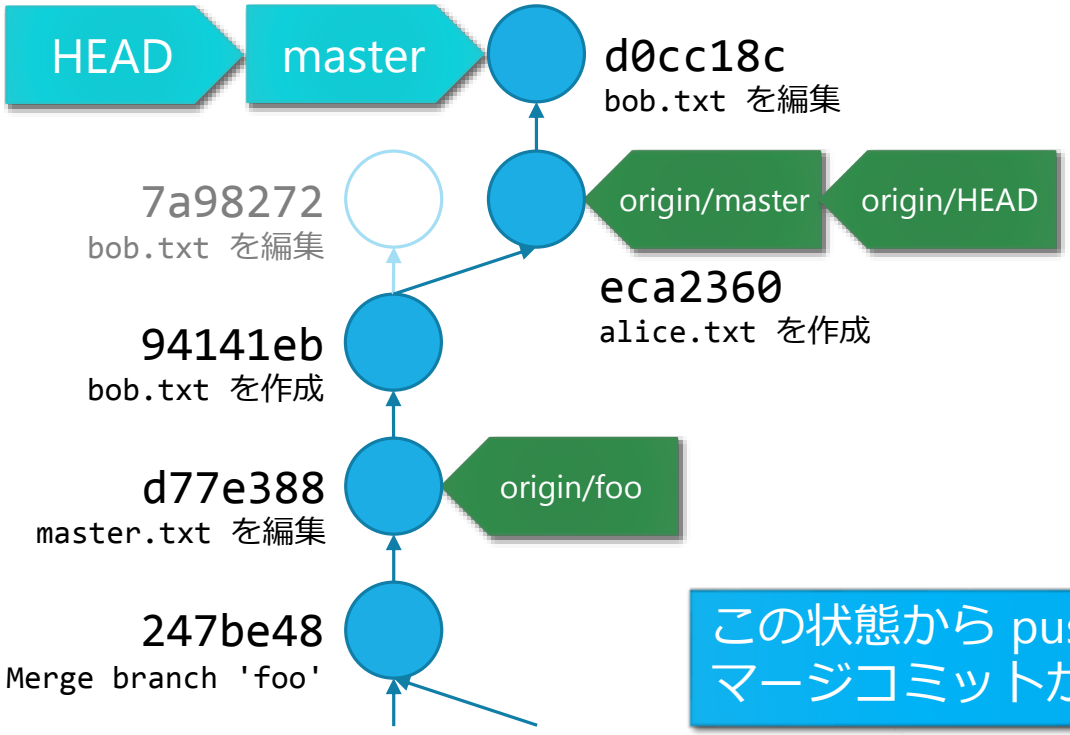
commit を作り直す

git rebase

```
$ git rebase
First, rewinding head to replay your work on top of it...
Applying: bob.txt を編集
```

- master でのコミットを origin/master に対してやり直す
- 衝突した場合は修正する必要がある

今どうなっているのか



この状態から push すれば
マージコミットが履歴に残らない

終わり

話さなかったこと

たくさんある

```
$ git help -a
usage: git [--version] [--help] [-c name=value]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p|--paginate|--no-pager] [--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
           <command> [<args>]

available git commands in '/usr/libexec/git-core'

  add                         count-objects             grep                      merge-recursive           remote                    show-ref
  add--interactive            credential                hash-object               merge-resolve             remote-ext                stage
  am                          credential-cache          help                      merge-subtree             remote-fd                 stash
  annotate                    credential-cache--daemon  http-backend              merge-tree                remote-ftp                status
  apply                       credential-gnome-keyring  http-fetch               mergetool                 remote-ftps               stripspace
  archive                     credential-store          http-push                 mktag                     remote-http               submodule
  bisect                      describe                  imap-send                 mktree                    remote-https              subtree
  bisect--helper              diff                      index-pack                mv                        remote-testpy             symbolic-ref
  blame                       diff-files                init                      name-rev                  repack                    tag
  branch                      diff-index                init-db                   notes                     replace                   tar-tree
  bundle                      diff-tree                 instaweb                  pack-objects              repo-config               unpack-file
  cat-file                    difftool                  log                       pack-redundant            request-pull              unpack-objects
  check-attr                  difftool--helper          lost-found                pack-refs                 rerere                    update-index
  check-ignore                fast-export               ls-files                  patch-id                  reset                     update-ref
  check-ref-format            fast-import               ls-remote                 peek-remote               rev-list                  update-server-info
  checkout                    fetch                     ls-tree                   prune                     rev-parse                 upload-archive
  checkout-index              fetch-pack                mailinfo                  prune-packed              revert                    upload-pack
  cherry                      filter-branch             mailsplit                 pull                      rm                        var
  cherry-pick                 fmt-merge-msg             merge                     push                      send-pack                 verify-pack
  clean                       for-each-ref              merge-base                quiltimport               sh-i18n--envsubst         verify-tag
  clone                       format-patch              merge-file                read-tree                 shell                     web--browse
  column                      fsck                      merge-index               rebase                    shortlog                  whatchanged
  commit                      fsck-objects              merge-octopus             receive-pack              show                      write-tree
  commit-tree                 gc                        merge-one-file            reflog                    show-branch
  config                      get-tar-commit-id         merge-ours                relink                    show-index

'git help -a' and 'git help -g' lists available subcommands and some
concept guides. See 'git help <command>' or 'git help <concept>'
to read about a specific subcommand or concept.
```

参考になる情報

探せばいくらでも見つかる

- 公式ウェブサイト
  - リファレンスマニュアル　http://git-scm.com/docs
  - 解説書『Pro Git』の和訳　http://git-scm.com/book/ja/v1
- 初心者向けの全般的な解説
  - Git チュートリアル　https://www.atlassian.com/ja/git/
  - いつやるの？ Git 入門　http://www.slideshare.net/matsukaz/git-28304397
  - サルでもわかる Git 入門　http://www.backlog.jp/git-guide/
- リンク集
  - Git 初心者が見るべきサイトまとめ
    http://matome.naver.jp/odai/2136491451473222801