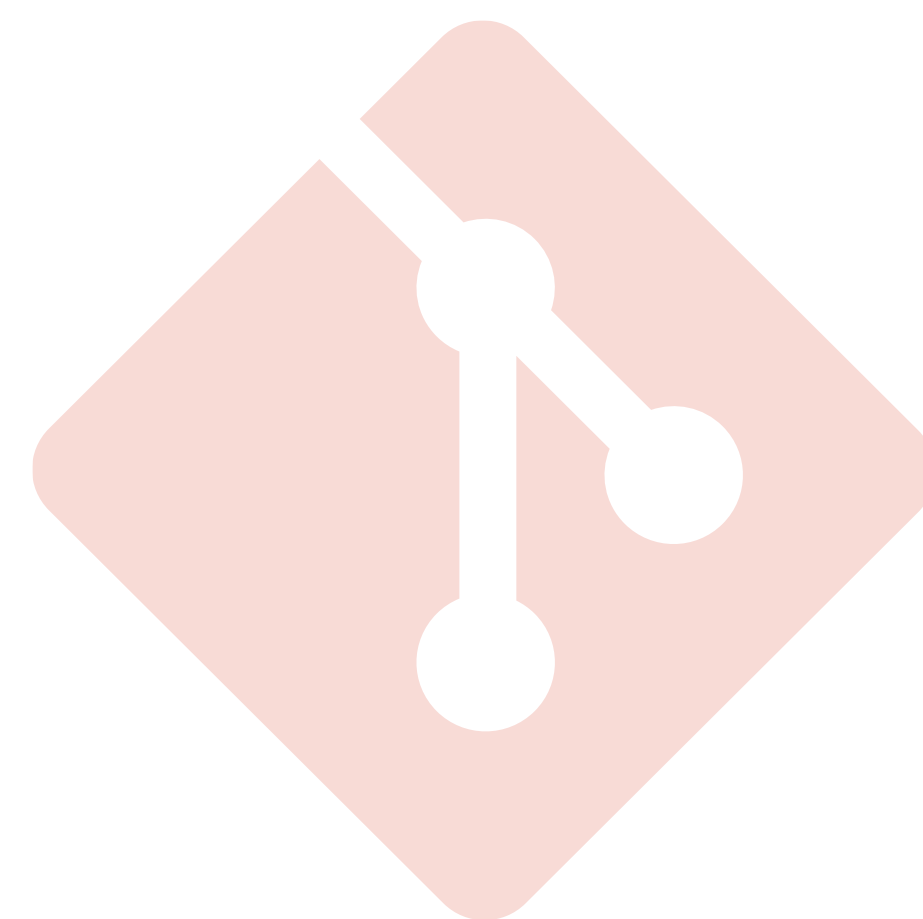


もう怖くないGit

Gitコマンド集



git

Gitの基本的なコマンド

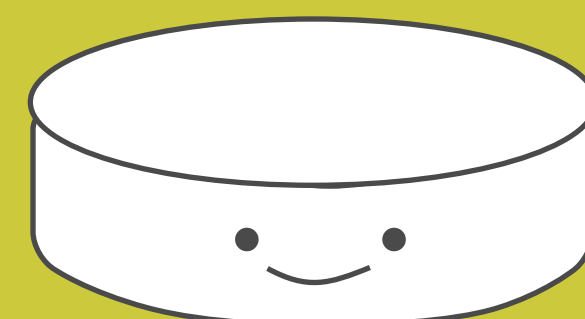
ローカルリポジトリの新規作成

>_ ターミナル

```
~ $ git init
```

initializeの略だよ

.gitディレクトリが作成される



.git/

- ・ リポジトリ
 - ・ 圧縮ファイル
 - ・ ツリーファイル
 - ・ コミットファイル
- ・ インデックスファイル
- ・ 設定ファイル

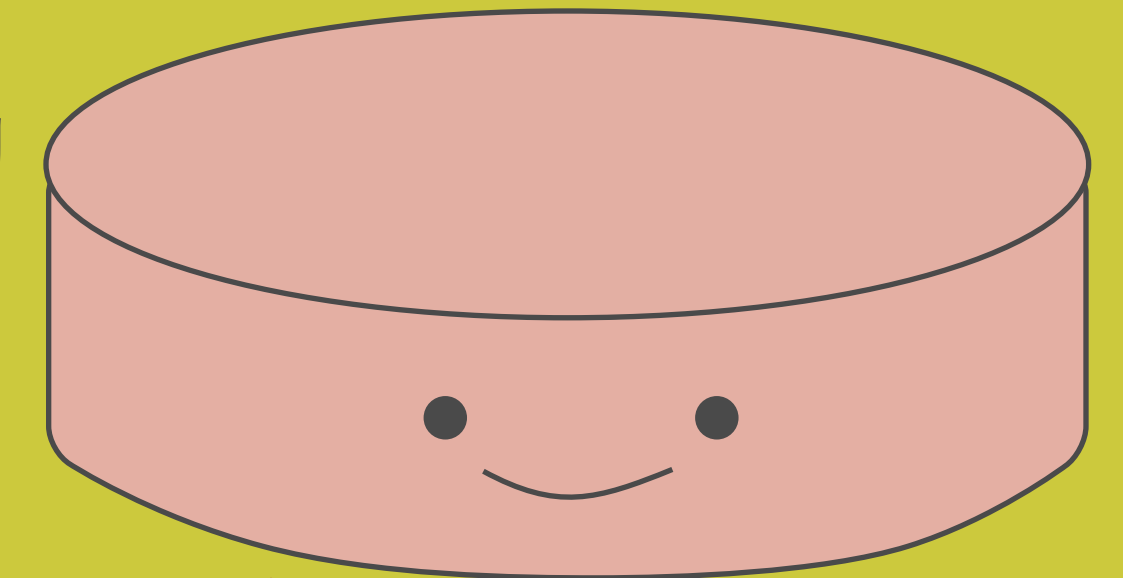
Gitリポジトリのコピーを作成する

>_ ターミナル

```
~ $ git clone <リポジトリ名>
```

クローン（コピーを）作成するよ

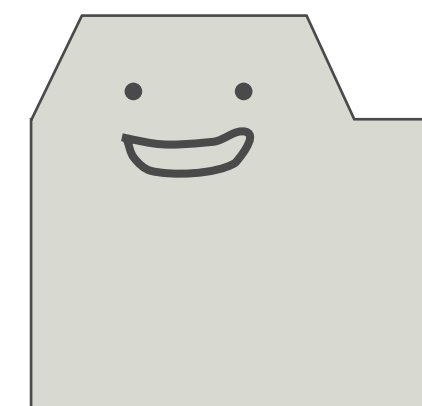
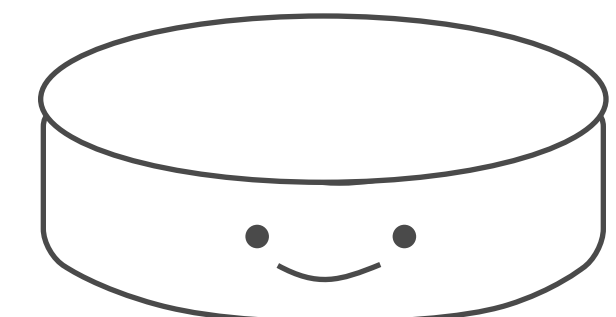
リモートリポジトリ
(GitHub)



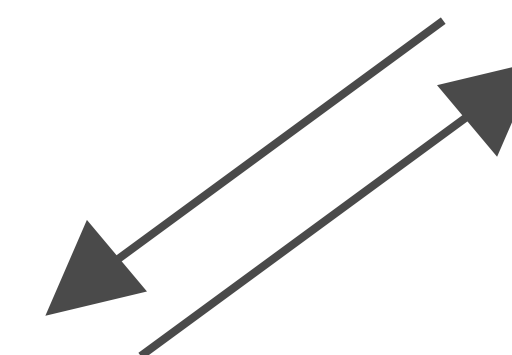
git clone

ローカル

リポジトリ



ワークツリー



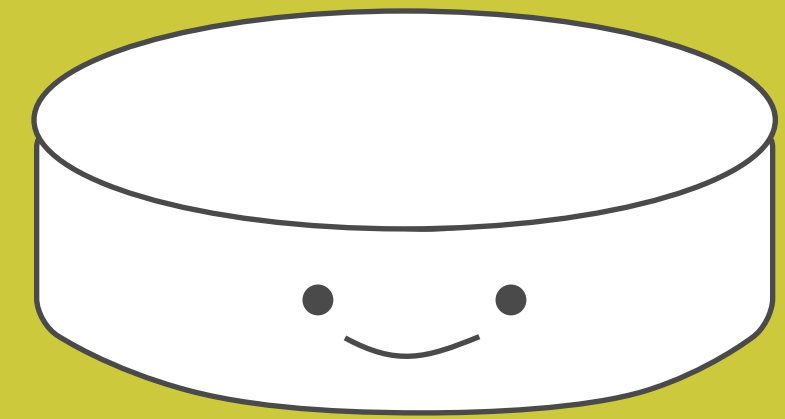
変更をステージに追加する

>_ ターミナル

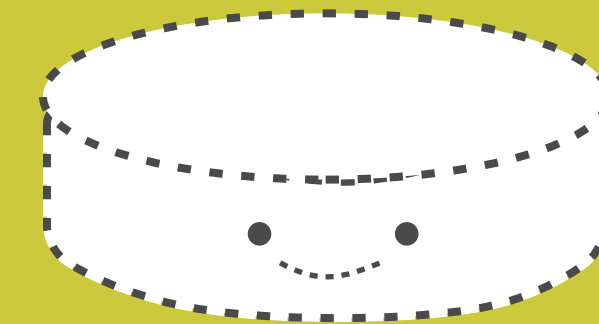
```
~ $ git add <ファイル名>  
~ $ git add <ディレクトリ名>  
~ $ git add .
```

追加するということだよ

リポジトリ

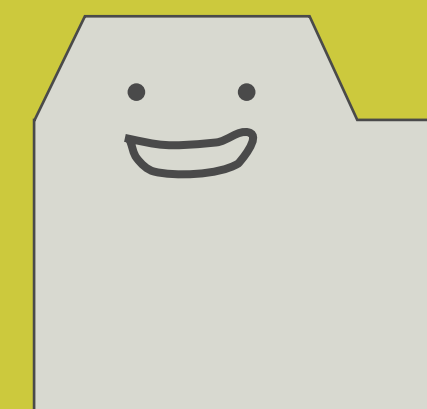


ステージ



コミットする
変更を準備

ワークツリー



git add



変更を記録する（コミット）

>_ ターミナル

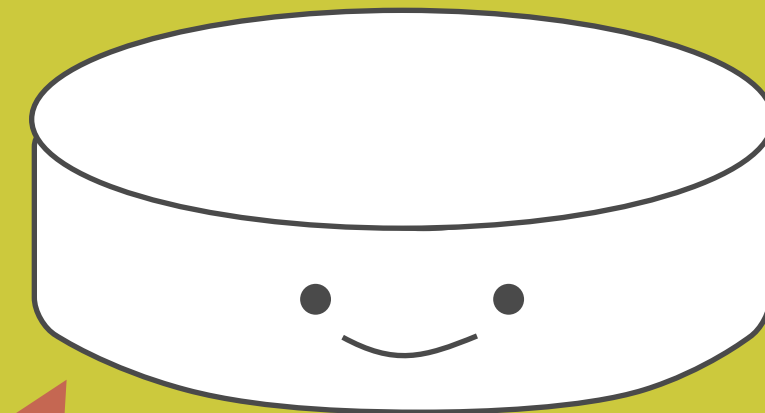
```
~ $ git commit  
~ $ git commit -m "<メッセージ>"  
~ $ git commit -v
```

メッセージ付き
で記録するよ

- ・ 変更
- ・ 新規作成
- ・ 削除
- ・ 複数ファイル
の変更、作成、
削除

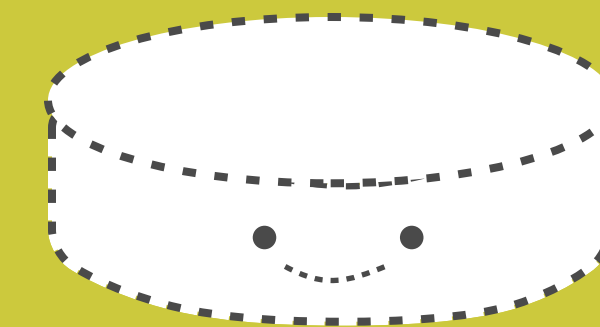
リポジトリ

スナップショットを記録

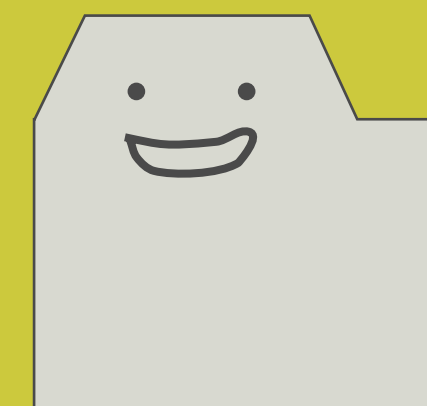


ステージ

git commit



ワークツリー



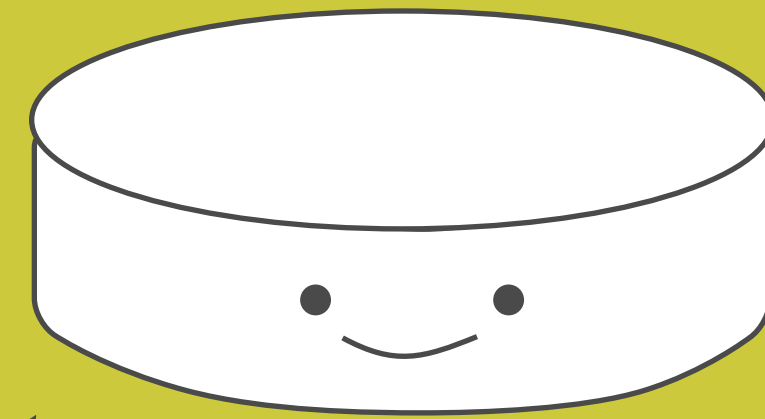
現在の変更状況を確認する

>_ ターミナル

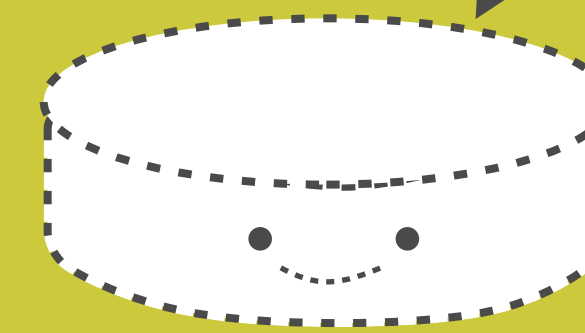
```
~ $ git status
```

変更されたファイルを確認するよ

リポジトリ

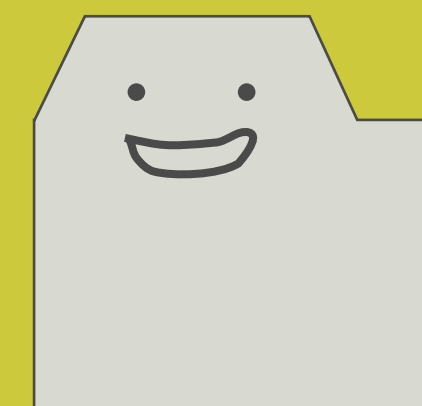


ステージ



変更された
ファイル

ワークツリー



変更されたファイル

変更差分を確認する

>_ ターミナル

```
# git addする前の変更分
```

```
~ $ git diff
```

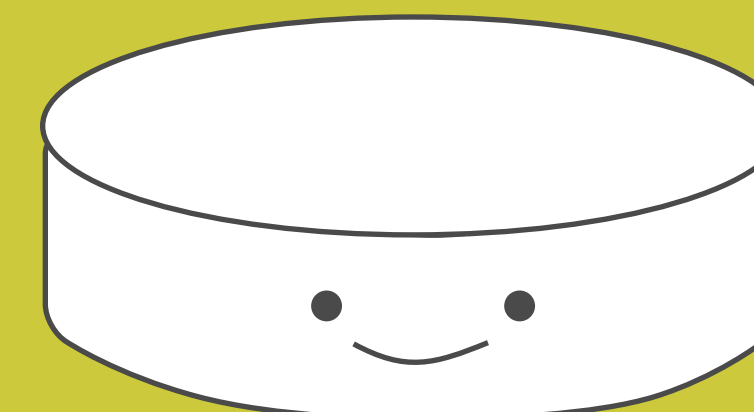
```
~ $ git diff <ファイル名>
```

```
# git addした後の変更分
```

```
$ git diff --staged
```

differenceの略だよ

リポジトリ

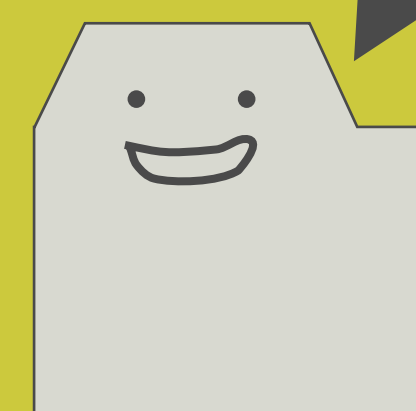


ステージ



git diff --staged

ワークツリー



git diff

変更履歴を確認する

>_ ターミナル

~ \$ git log

一行で表示する

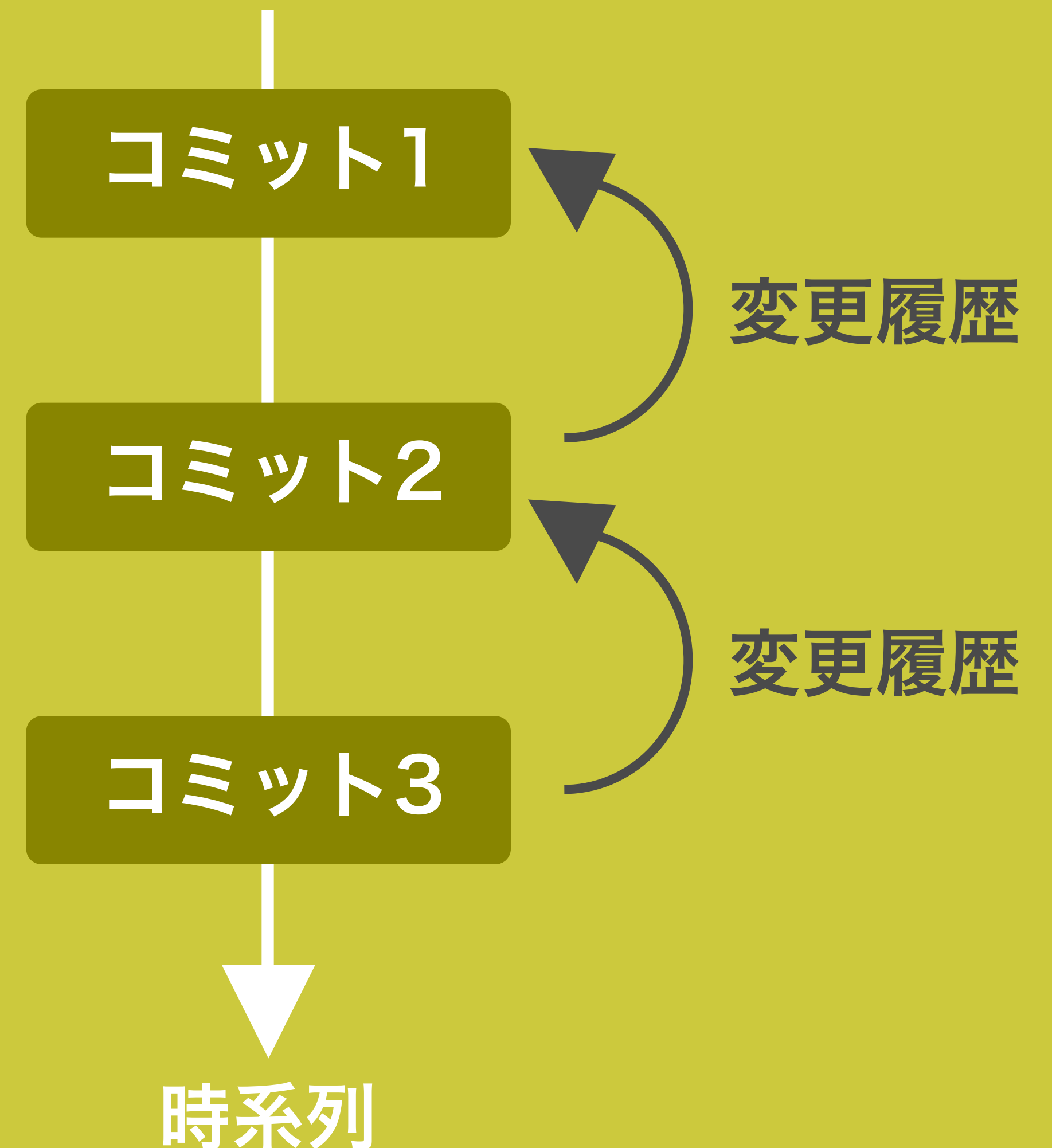
~ \$ git log --oneline

ファイルの変更差分を表示する

~ \$ git log -p index.html

表示するコミット数を制限する

~ \$ git log -n <コミット数>



ファイルの削除を記録する

>_ ターミナル

ファイルごと削除

~ \$ git rm <ファイル名>

~ \$ git rm -r <ディレクトリ名>

ファイルを残したいとき

~ \$ git rm --cached <ファイル名>

removeの略だよ

リポジトリ



index.html

git rm
--cached

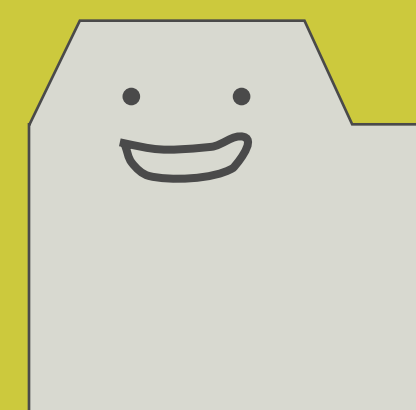
削除

ステージ



git rm

ワークツリー



index.html

削除

ファイルの移動を記録する

>_ ターミナル

```
~ $ git mv <旧ファイル> <新ファイル>
```

以下のコマンドと同じ

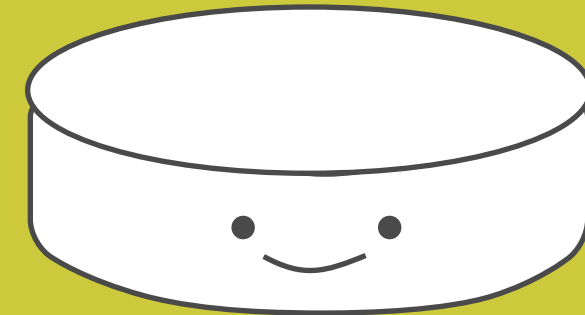
```
~ $ mv <旧ファイル> <新ファイル>
```

```
~ $ git rm <旧ファイル>
```

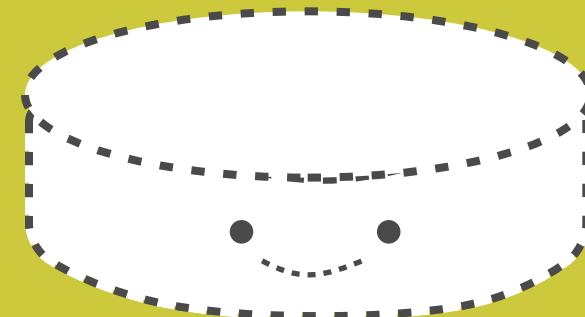
```
~ $ git add <新ファイル>
```

通常のコマンドを使っても代替できるので無理して使わなくても大丈夫だよ

リポジトリ



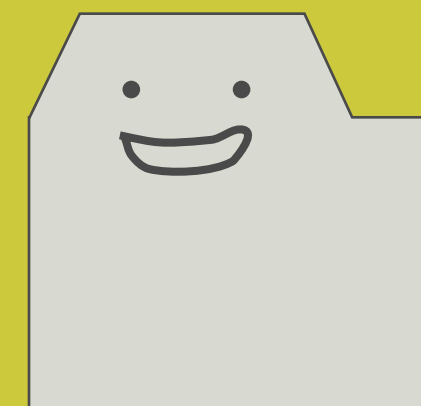
ステージ



インデックス

~~index.html~~
index2.html

ワークツリー



git mv
index.html → index2.html

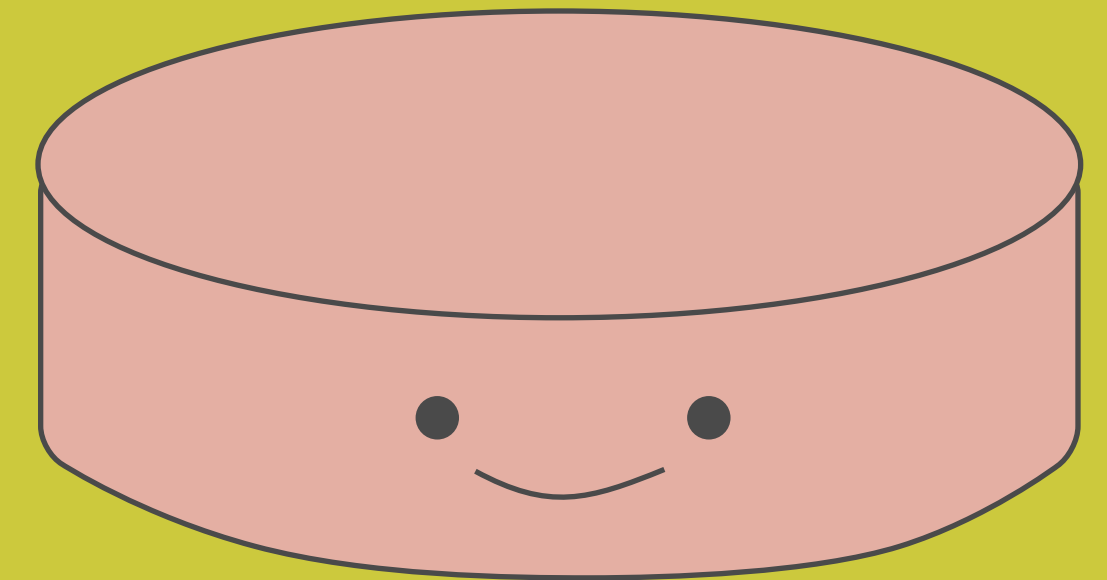
リモートリポジトリ (GitHub) を新規追加する

>_ ターミナル

```
~ $ git remote add origin  
https://github.com/user/repo.git
```

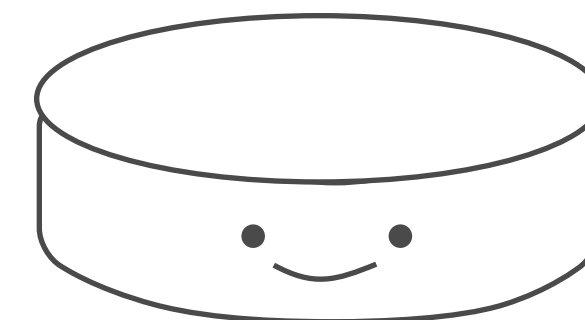
originというショートカットで
urlのリモートリポジトリを
登録するよ

リモートリポジトリ
(GitHub)



git remote add

ローカル



ローカル
リポジトリ

今後はorigin
という名前で
GitHubリポジトリに
アップしたり
取得したりできるよ

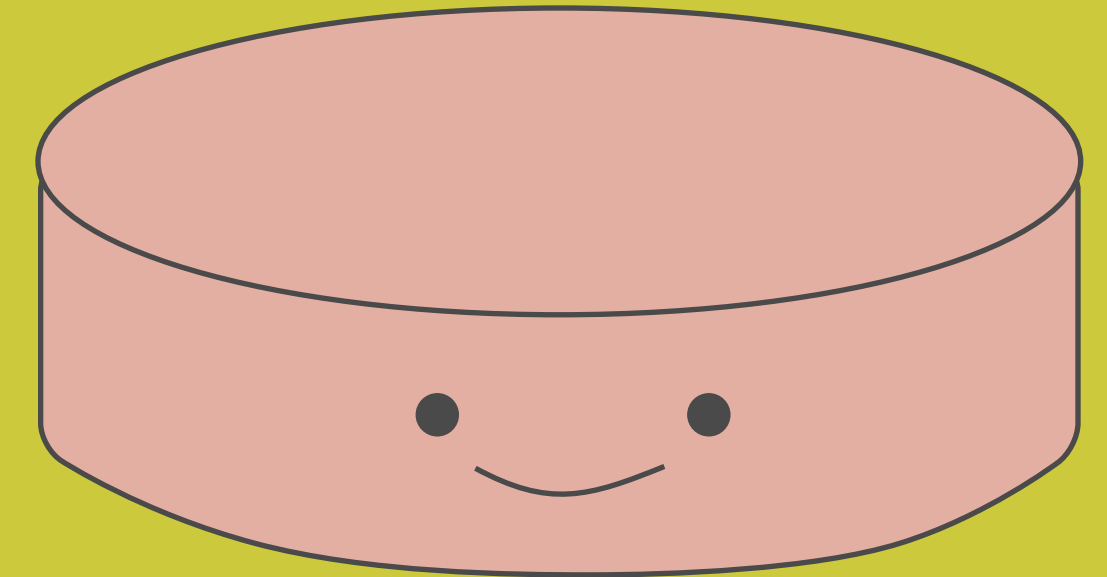
リモートリポジトリ (GitHub) へ送信する

>_ ターミナル

```
~ $ git push <リモート名> <ブランチ名>  
~ $ git push origin master
```

ローカルリポジトリの内容を
リモートリポジトリに送ることを
「プッシュ」と言うよ

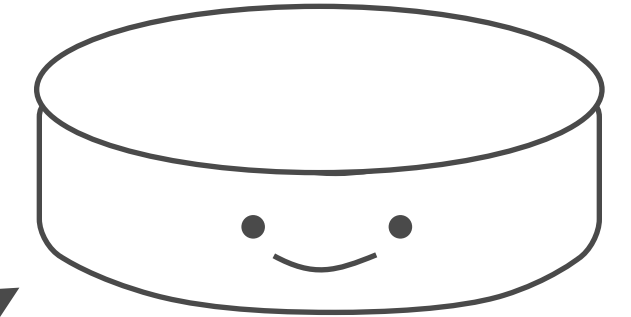
リモートリポジトリ
(GitHub)



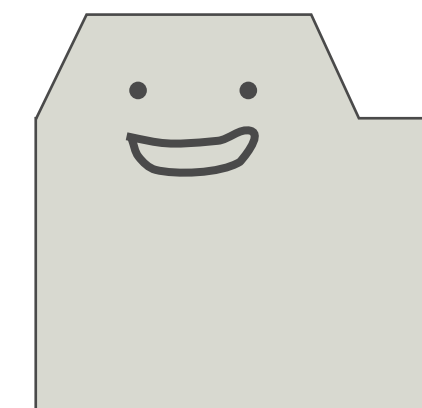
git push

ローカル

ローカルリポジトリ



git commit



ワークツリー

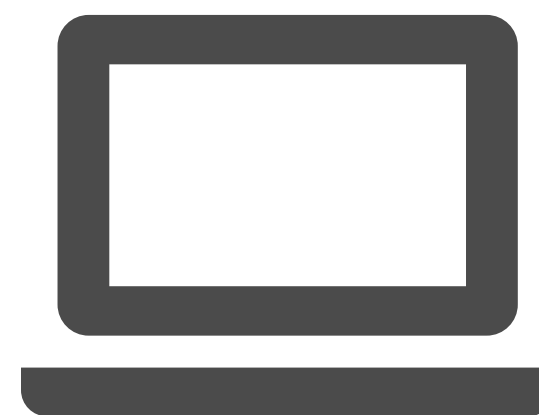
コマンドにエイリアスを付ける

>_ ターミナル

```
~ $ git config --global alias.ci commit  
~ $ git config --global alias.st status  
~ $ git config --global alias.br branch  
~ $ git config --global alias.co  
checkout
```

エイリアスを付けておくと
入力が楽になるよ

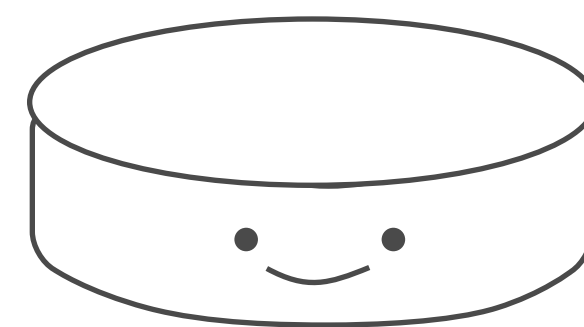
ローカル



~/.gitconfig

~/.config/git/config

--globalを付けると
PC全体の設定になるよ



project/.git/config

ローカルリポジトリ

管理しないファイルをGitの管理から外す

.gitignoreファイルに指定する



こういったファイル
は管理しないぜ

- ・ 自動生成される
ファイル
- ・ パスワードが記載
されているファイル

.gitignoreファイルの書き方

```
# #から始まる行はコメント  
  
# 指定したファイルを除外  
index.html  
# ルートディレクトリを指定  
/root.html  
# ディレクトリ以下を除外  
dir/  
# /以外の文字列にマッチ「*」  
/*/*.css
```

変更を元に戻すコマンド

ファイルへの変更を取り消す

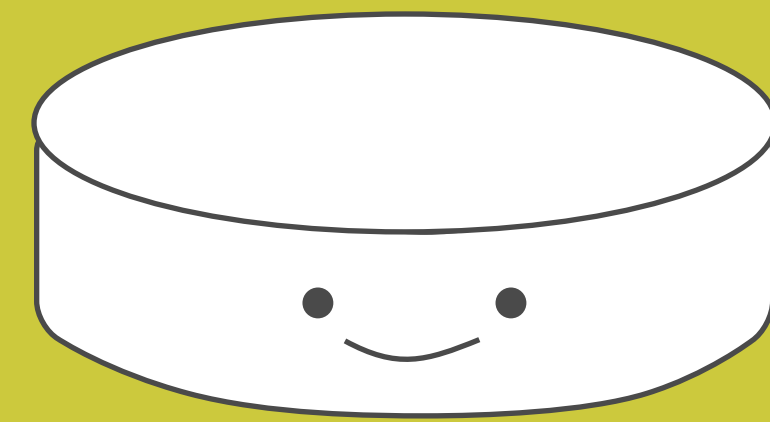
>_ ターミナル

```
~ $ git checkout -- <ファイル名>  
~ $ git checkout -- <ディレクトリ名>
```

```
# 全変更を取り消す  
~ $ git checkout -- .
```

"--"を付けているのは、ブランチ名とファイル名が被った時に、どちらを指しているのかGitが分からなくなるのを避けるためだよ。

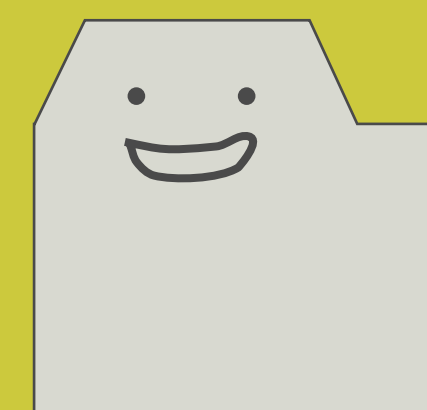
リポジトリ



ステージ



ワークツリー



git checkout
-- <ファイル名>

~~変更~~

ステージした変更を取り消す

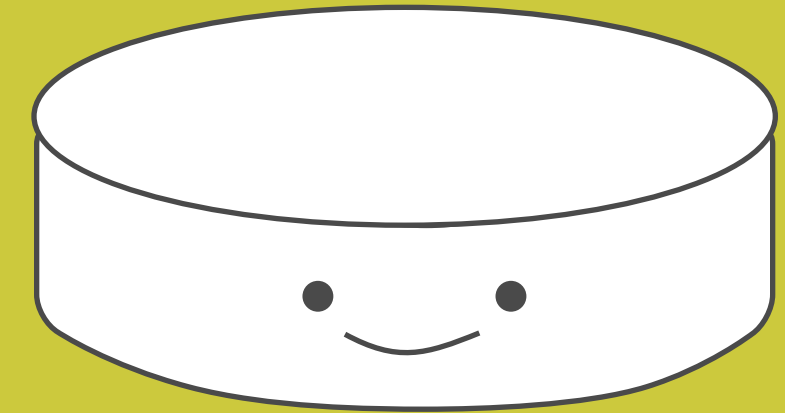
>_ ターミナル

```
~ $ git reset HEAD <ファイル名>  
~ $ git reset HEAD <ディレクトリ名>
```

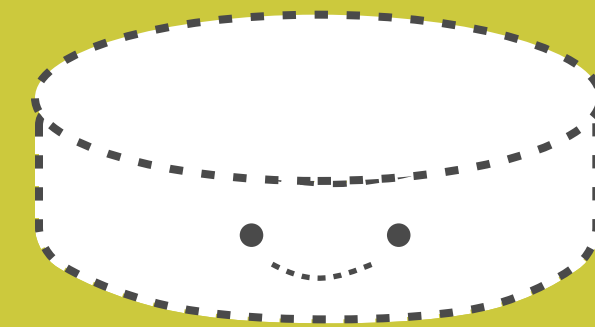
```
# 全変更を取り消す  
~ $ git reset HEAD .
```

指定した変更をステージから取り消すだけなので、ワークツリーのファイルには影響を与えないよ。

リポジトリ



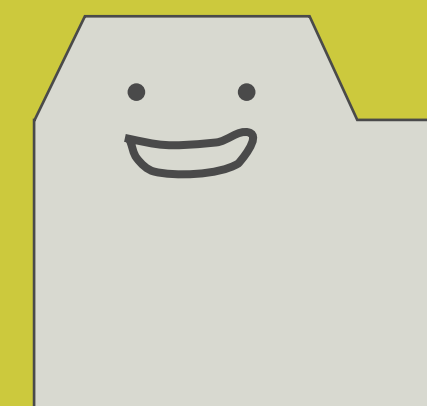
ステージ



git reset HEAD
<ファイル名>

~~変更~~

ワークツリー



git add

変更

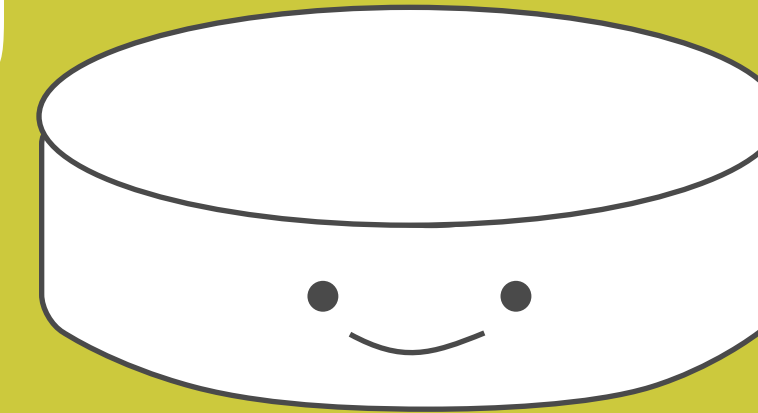
直前のコミットをやり直す

>_ ターミナル

```
~ $ git commit --amend
```

リモートリポジトリにPushした
コミットはやり直したらダメだよ。

リポジトリ



コミット

git commit
--amend

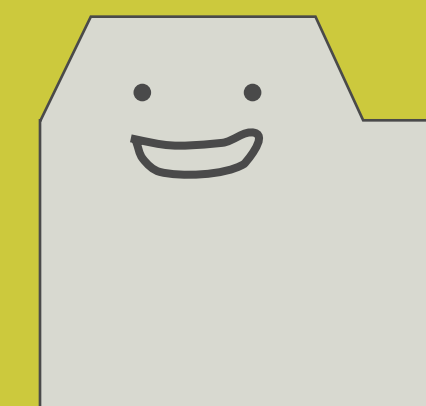
ステージ



変更

ワークツリー

git add



変更

GitHubとやり取りするコマンド

リモートを表示する

>_ ターミナル

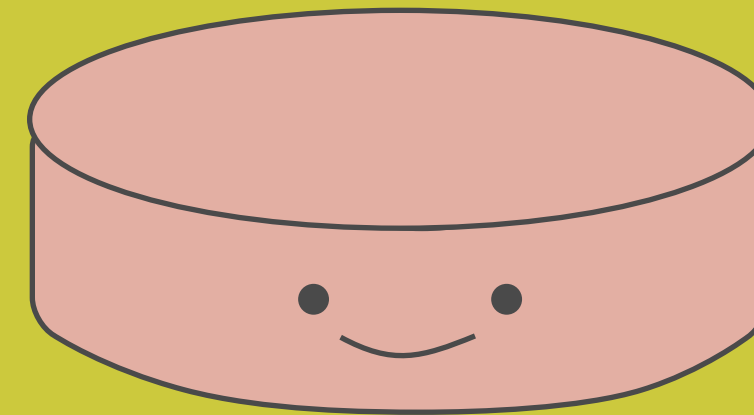
```
~ $ git remote
```

```
# 対応するURLを表示
```

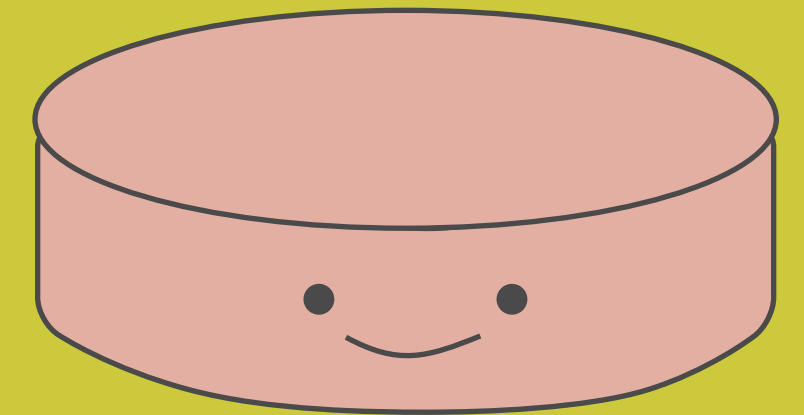
```
~ $ git remote -v
```

設定しているリモートリポジトリの
情報を表示するよ

リモート

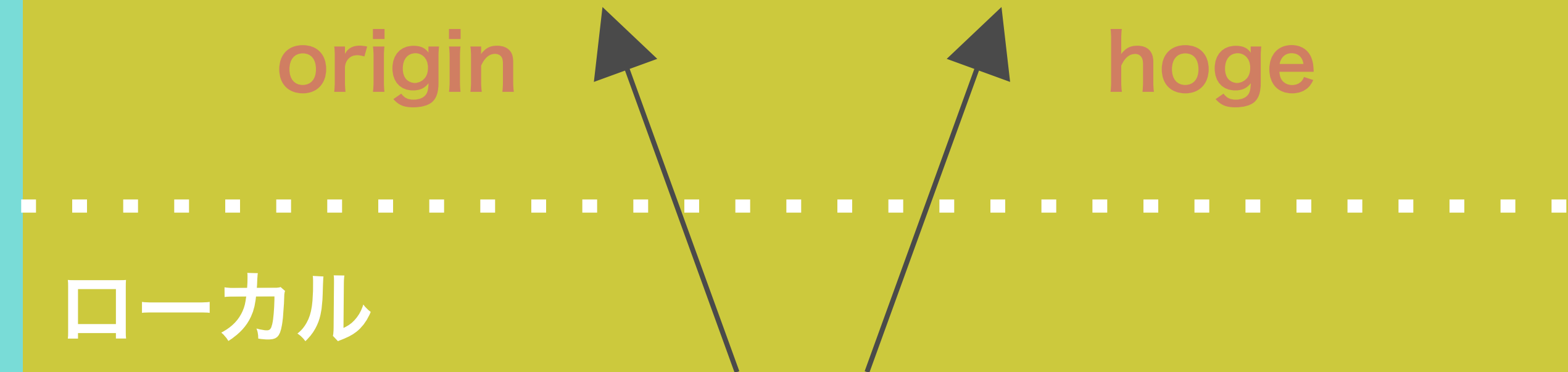
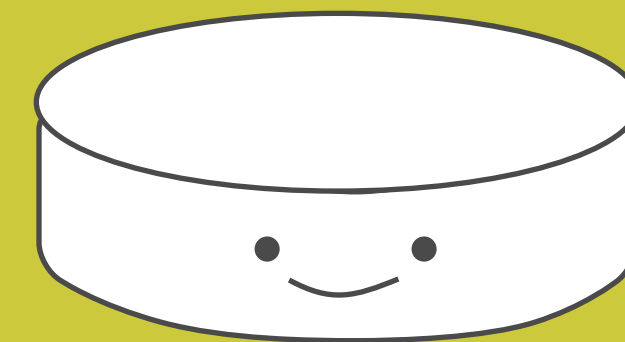


origin



hoge

ローカル



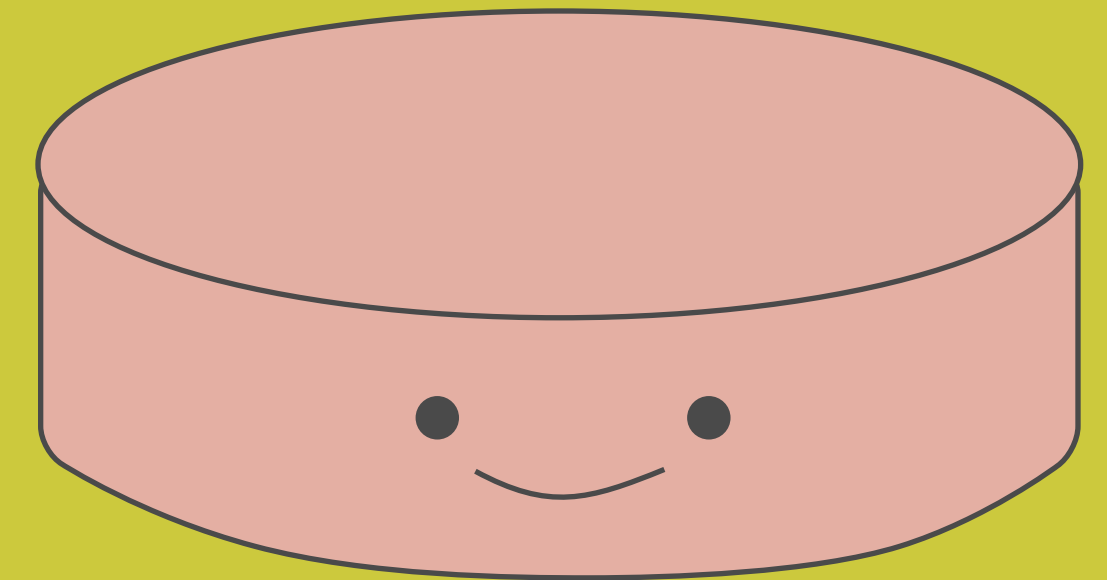
リモートリポジトリを新規追加する

>_ ターミナル

```
~ $ git remote add <リモート名>  
    <リモートURL>  
~ $ git remote add tutorial  
    https://github.com/user/repo.git
```

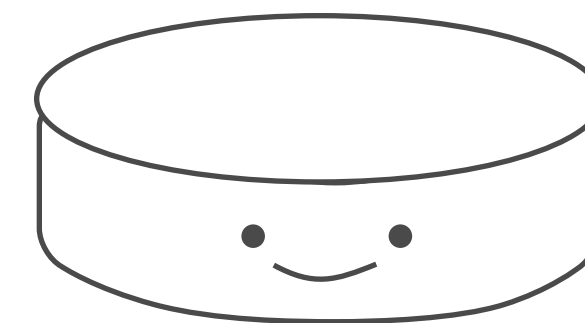
tutorialというショートカットで
urlのリモートリポジトリを
登録するよ

リモートリポジトリ



git remote add

ローカル



ローカル
リポジトリ

今後はtutorial
という名前で
GitHubリポジトリに
アップしたり
取得したりできるよ

リモートから情報を取得する（フェッチ）

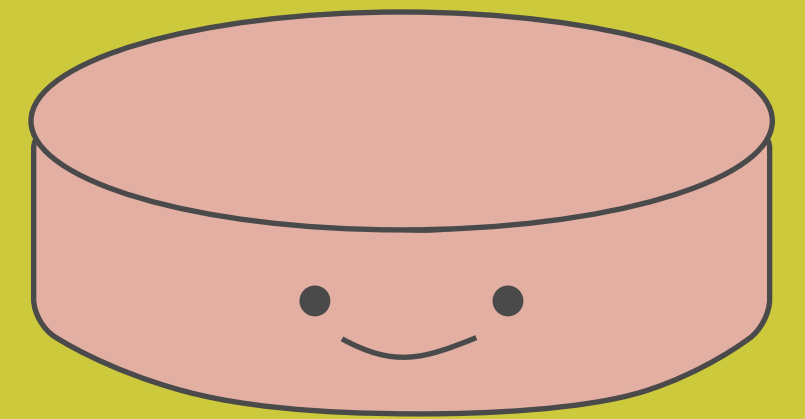
>_ ターミナル

```
~ $ git fetch <リモート名>  
~ $ git fetch origin
```

取ってくるということだよ

リモート

リモート
リポジトリ



git fetch
remotes/リモート/ブランチ



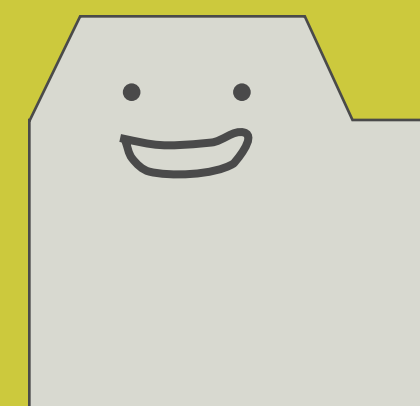
ローカル

ローカル
リポジトリ



git merge

ワークツリー



リモートから情報を取得してマージする (プル)

>_ ターミナル

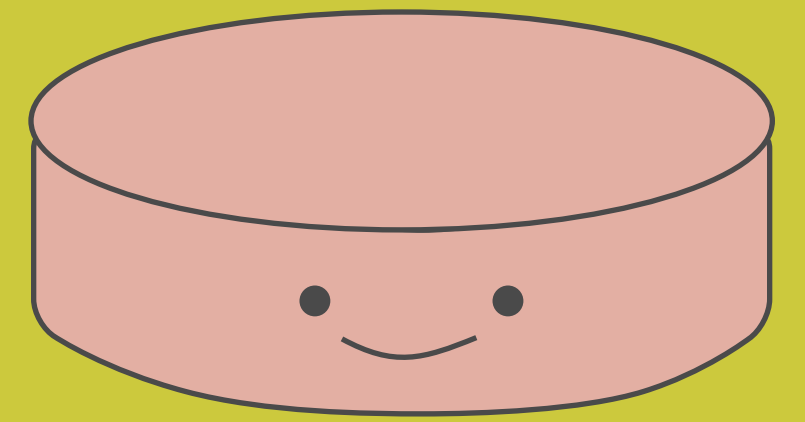
```
~ $ git pull <リモート名> <ブランチ名>  
~ $ git pull origin master
```

```
# 上記コマンドは省略可能  
~ $ git pull
```

```
# これは下記コマンドと同じこと  
~ $ git fetch origin master  
~ $ git merge origin master
```

リモート

リモート
リポジトリ

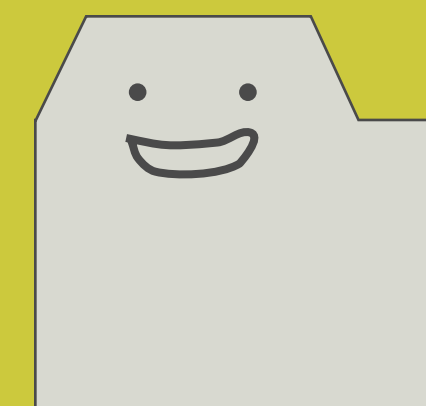
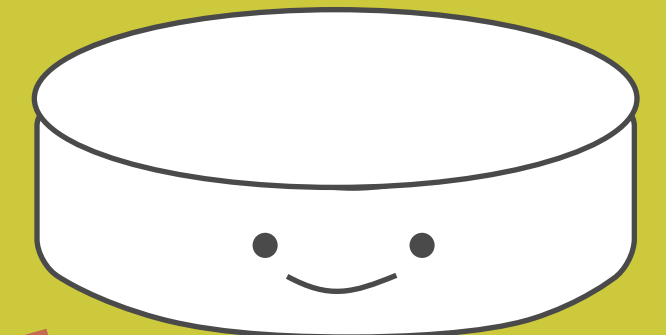


git pull



ローカル

ローカル
リポジトリ



ワークツリー



リモートの詳細情報を表示する

>_ ターミナル

```
~ $ git remote show <リモート名>  
~ $ git remote show origin
```

git remote コマンドより
詳しい情報を表示するよ

表示される情報

- ・ FetchとPushのURL
- ・ リモートブランチ
- ・ git pullの挙動
- ・ git pushの挙動

リモートを変更・削除する

変更する

>_ ターミナル

```
~ $ git remote rename  
    <旧リモート名> <新リモート名>  
~ $ git remote rename  
    tutorial new_tutorial
```

削除する

>_ ターミナル

```
~ $ git remote rm <リモート名>  
~ $ git remote rm new_tutorial
```

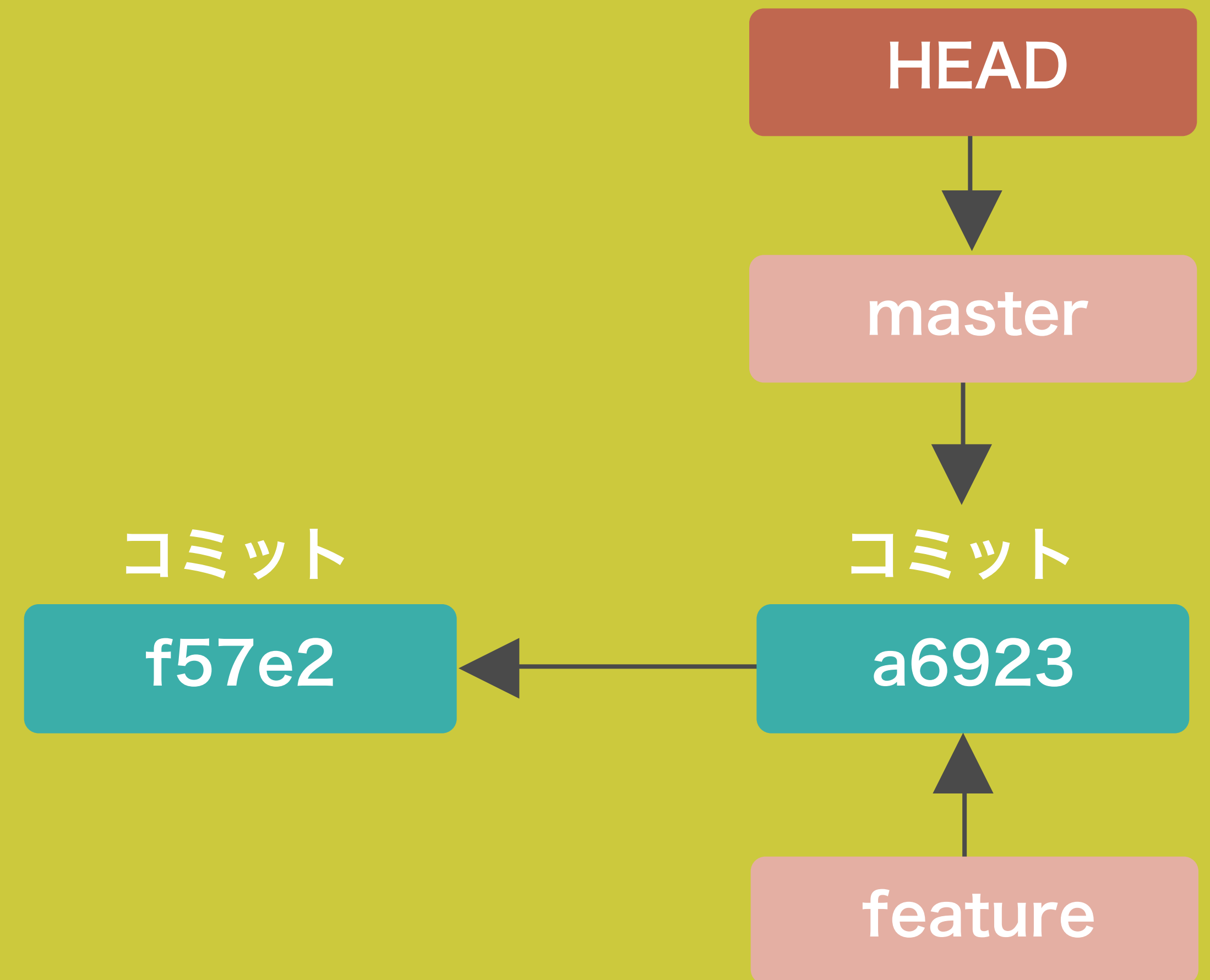
ブランチとマージのコマンド

ブランチを新規追加する

>_ ターミナル

```
~ $ git branch <ブランチ名>  
~ $ git branch feature
```

ブランチを作成するだけで、
ブランチの切り替えまでは
行わないよ。



ブランチの一覧を表示する

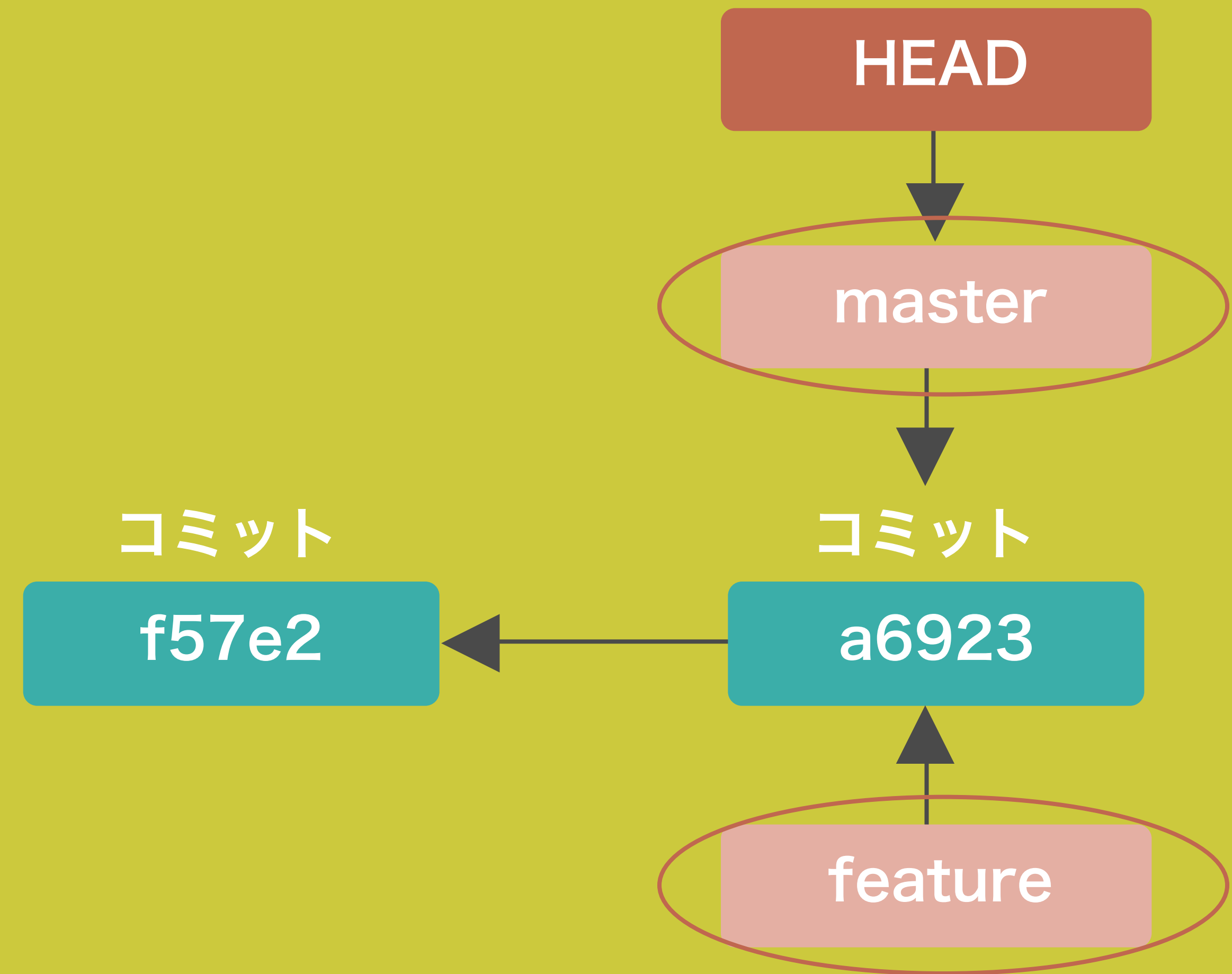
>_ ターミナル

~ \$ git branch

全てのブランチを表示する

~ \$ git branch -a

何のブランチがあるかを確認したい時に使うよ。



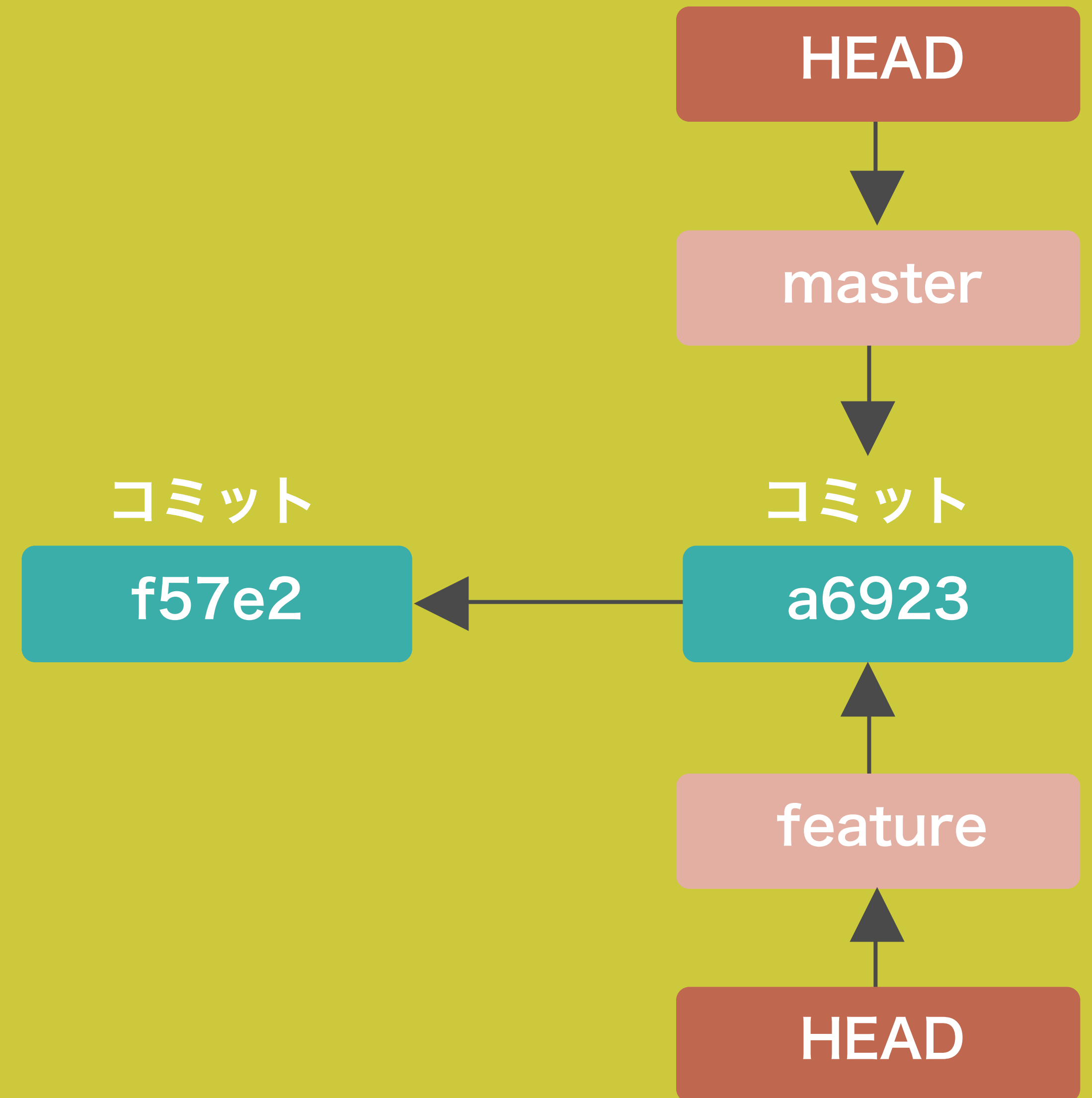
ブランチを切り替える

>_ ターミナル

```
~ $ git checkout <既存ブランチ名>  
~ $ git checkout feature
```

```
# ブランチを新規作成して切り替える  
~ $ git checkout -b <新ブランチ名>
```

-b オプションを付けると
ブランチの作成と切り替えを
一度にしてくれるので楽だよ

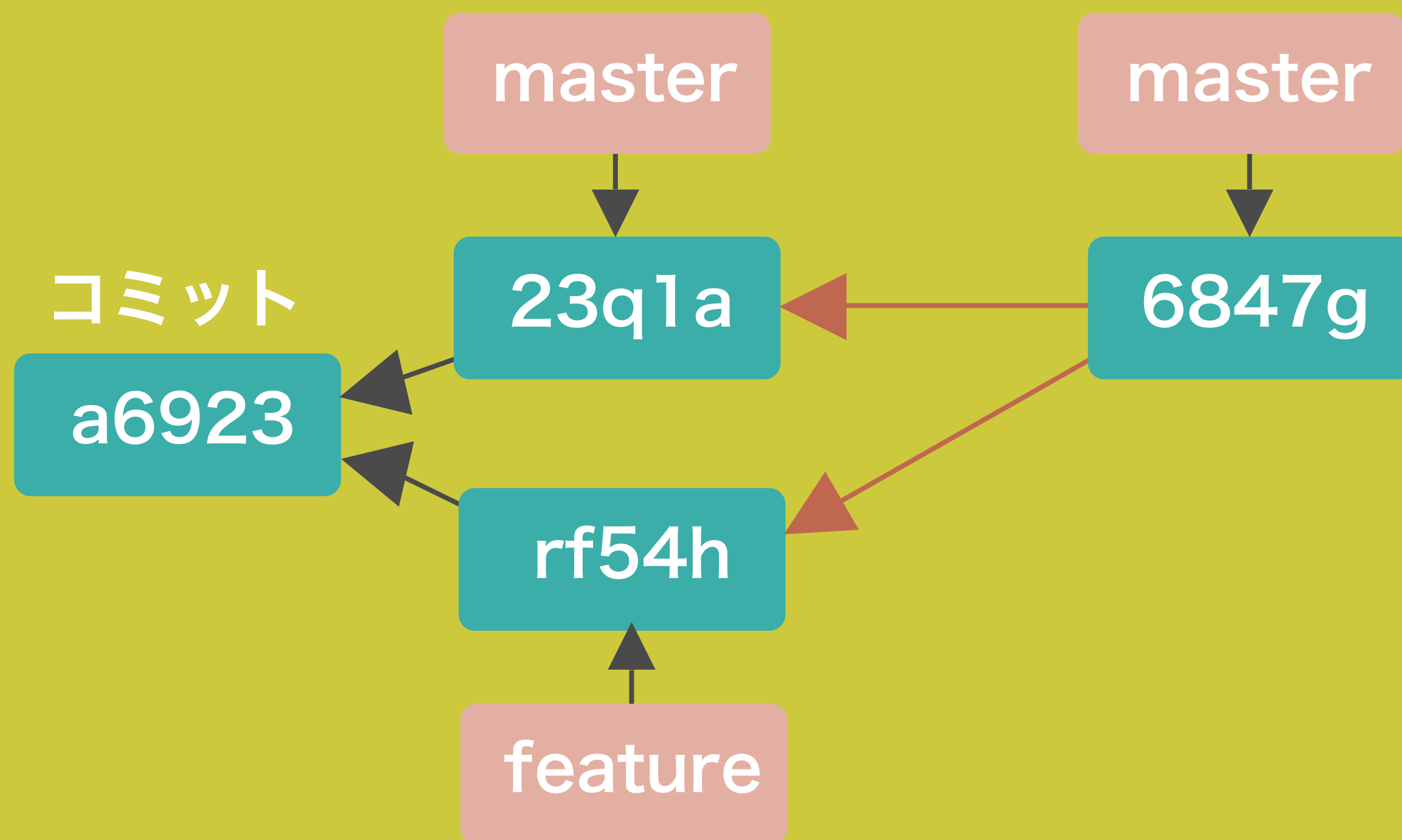


変更履歴をマージする

>_ ターミナル

```
~ $ git merge <ブランチ名>  
~ $ git merge <リモート名/ブランチ名>  
~ $ git merge origin/master
```

作業中のブランチにマージするよ



ブランチを変更・削除する

変更する

>_ ターミナル

```
~ $ git branch -m <ブランチ名>  
~ $ git branch -m new_branch
```

自分が作業しているブランチの名前を変更するよ

削除する

>_ ターミナル

```
~ $ git branch -d <ブランチ名>  
~ $ git branch -d feature
```

強制削除する

```
~ $ git branch -D <ブランチ名>
```

masterにマージされていない
変更が残っている場合削除しないよ

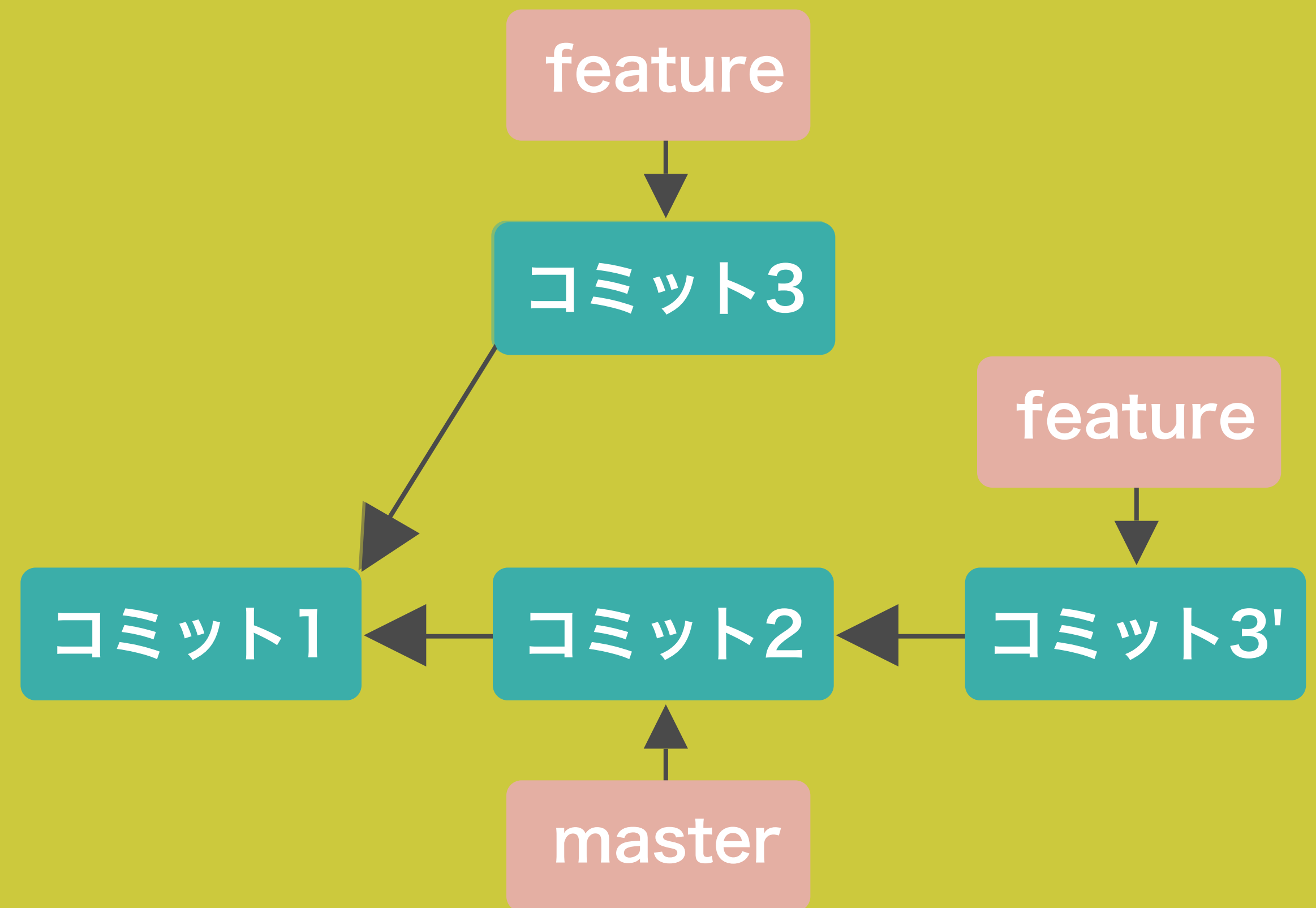
リベースのコマンド

リベースで履歴を整えた形で変更を統合する

>_ ターミナル

~ \$ git rebase <ブランチ名>

ブランチの基点となるコミットを別のコミットに移動するよ



プルのリベース型

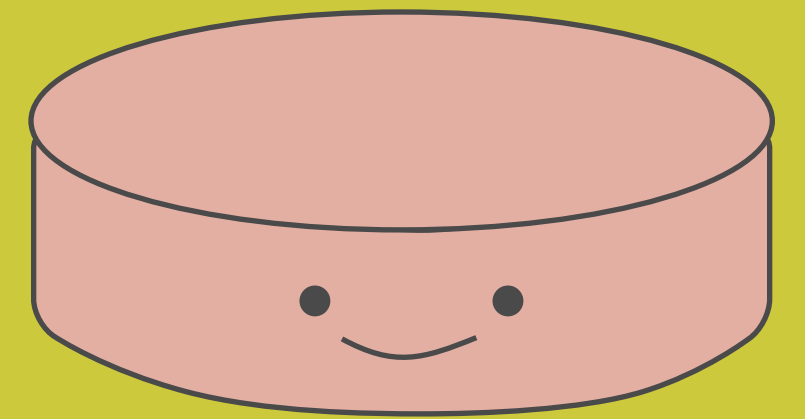
>_ ターミナル

```
~ $ git pull --rebase  
  <リモート名> <ブランチ名>  
~ $ git pull --rebase origin master
```

マージコミットが残らないから、
GitHubの内容を取得したい
だけの時は --rebase を使おう

リモート

リモート
リポジトリ



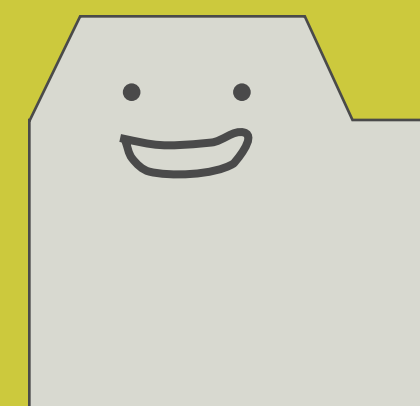
git fetch

ローカル

ローカル
リポジトリ



git rebase



ワークツリー

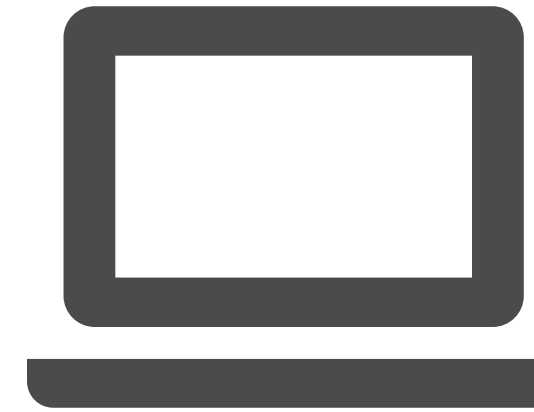
プルをリベース型に設定する

>_ ターミナル

```
~ $ git config --global pull.rebase true  
  
# masterブランチでgit pullする時だけ  
~ $ git config branch.master.rebase  
true
```

--rebase オプションを付けなくても
git pullの挙動がリベース型になるよ

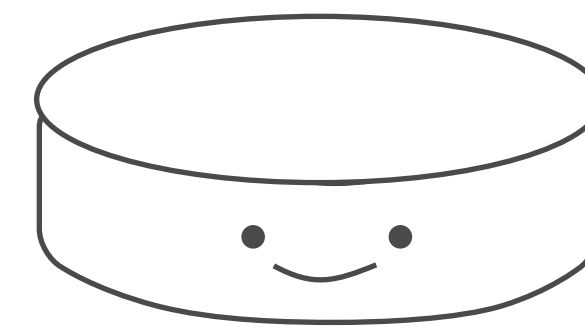
ローカル



~/.gitconfig

~/.config/git/config

--globalを付けると
PC全体の設定になるよ



project/.git/config

ローカルリポジトリ

複数のコミットをやり直す

>_ ターミナル

```
~ $ git rebase -i <コミットID>  
~ $ git rebase -i HEAD~3
```

```
pick gh21f6d ヘッダー修正  
pick 193054e ファイル追加  
pick 84gha0d README修正
```

-i は --interactive の略だよ。
対話的リベースといって、やり取りしながら履歴を変更していくよ。

>_ ターミナル

```
# やり直したいcommitをeditにする  
edit gh21f6d ヘッダー修正  
pick 193054e ファイル追加  
pick 84gha0d README修正
```

```
# やり直したら実行する  
$ git commit --amend
```

```
# 次のコミットへ進む（リベース完了）  
$ git rebase --continue
```

コミットを並び替える、削除する

>_ ターミナル

```
~ $ git rebase -i HEAD~3
```

```
pick gh21f6d ヘッダー修正  
pick 193054e ファイル追加  
pick 84gha0d README修正
```

履歴は古い順に表示されるので
注意してね。git logとは逆だよ。

>_ ターミナル

```
# ①84gha0dのコミットを消す  
# ②193054eを先に適用する  
pick 193054e ファイル追加  
pick gh21f6d ヘッダー修正
```

コミットを削除したり
並び替えたりできるよ。

コミットをまとめる

>_ ターミナル

```
~ $ git rebase -i HEAD~3
```

```
pick gh21f6d ヘッダー修正  
pick 193054e ファイル追加  
pick 84gha0d README修正
```

>_ ターミナル

```
# コミットを1つにまとめる  
pick gh21f6d ヘッダー修正  
squash 193054e ファイル追加  
squash 84gha0d README修正
```

squashを指定するとそのコミットを直前のコミットと一つにするよ

コミットを分割する

>_ ターミナル

```
~ $ git rebase -i HEAD~3
```

```
pick gh21f6d ヘッダー修正
```

```
pick 193054e ファイル追加
```

```
pick 41gha0d READMEとindex修正
```

>_ ターミナル

```
# コミットを分割する
```

```
pick gh21f6d ヘッダー修正
```

```
pick 193054e ファイル追加
```

```
edit 84gha0d READMEとindex修正
```

```
$ git reset HEAD^
```

```
$ git add README
```

```
$ git commit -m 'README修正'
```

```
$ git add index.html
```

```
$ git commit -m 'index.html修正'
```

```
$ git rebase --continue
```


タグ付けのコマンド

タグの一覧を表示する

>_ ターミナル

~ \$ git tag

パターンを指定してタグを表示

\$ git tag -l "201705"

20170501_01

20170501_02

20170503_01

git tag コマンドはアルファベット順にタグを表示するよ

20170501_01

20170501_02

コミット1

コミット2

コミット3

タグを作成する（注釈付きタグ）

>_ ターミナル

```
~ $ git tag -a [タグ名]  
    -m "[メッセージ]"  
~ $ git tag -a 20170520_01  
    -m "version 20170520_01"
```

-a オプションを付けると注釈付きタグを作成するよ。
-m オプションを付けるとエディタを立ち上げずにメッセージを入力できるよ。

- ・ 名前を付けられるよ
- ・ コメントを付けられるよ
- ・ 署名を付けられるよ

20170501_01

20170501_02

コミット1

コミット2

コミット3

タグを作成する（軽量版タグ）

>_ ターミナル

```
~ $ git tag [タグ名]  
~ $ git tag 20170520_01  
  
# 後からタグ付けする  
~ $ git tag [タグ名] [コミット名]  
~ $ git tag 20170520_01 8a6cbc4
```

オプションを付けないと軽量版タグを作成するよ。

・ 名前を付けられるよ

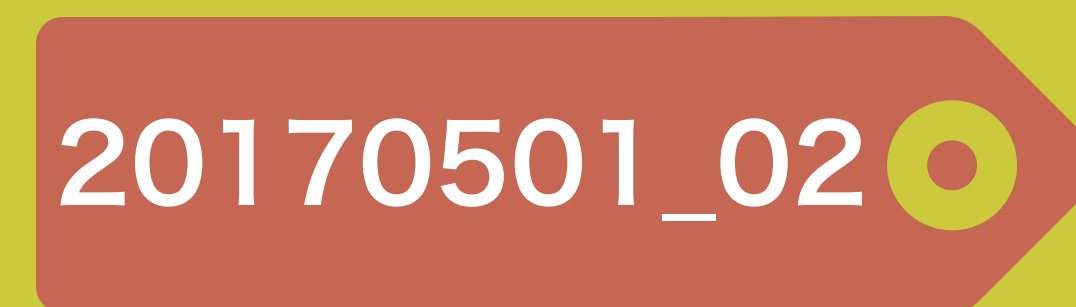
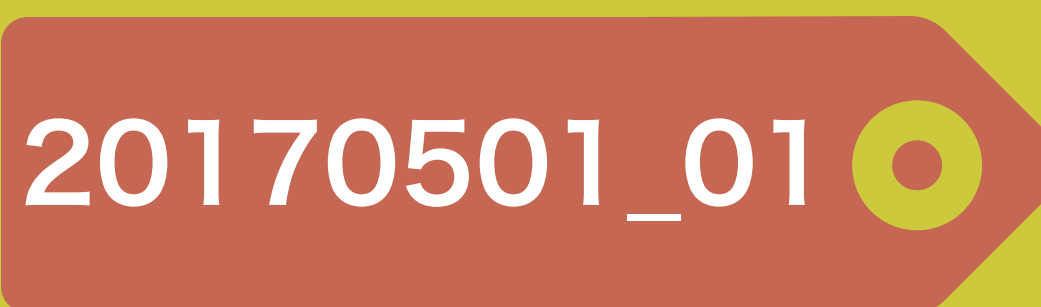
20170501_01

20170501_02

コミット1

コミット2

コミット3



タグのデータを表示する

>_ ターミナル

~ \$ git show [タグ名]

~ \$ git show 20170520_01

タグのデータと関連付けられた
コミットを表示するよ。

- ・ タグ付けした人の情報
- ・ タグ付けした日時
- ・ 注釈メッセージ
- ・ コミット

20170501_01

20170501_02

コミット1

コミット2

コミット3

タグをリモートリポジトリに送信する

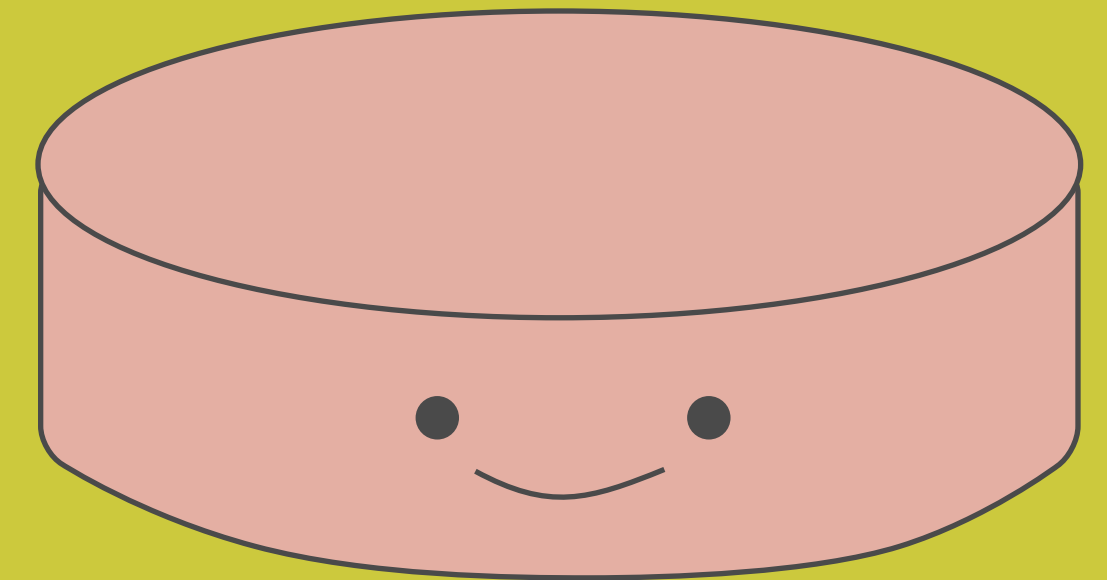
>_ ターミナル

```
~ $ git push [ブランチ名] [タグ名]  
~ $ git push origin 20170520_01
```

```
# タグを一斉に送信する  
~ $ git push origin --tags
```

--tags を付けるとローカルにあってリモートリポジトリに存在しないタグを一斉に送信するよ。

リモートリポジトリ
(GitHub)

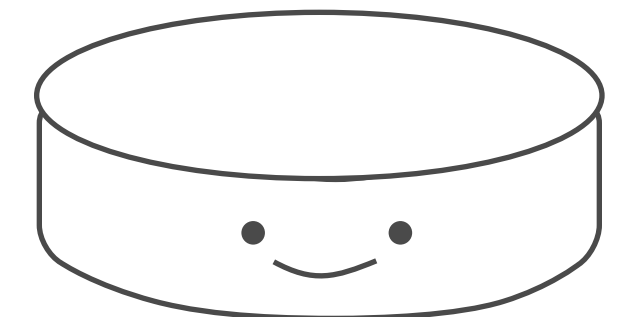


git push origin 20170520_01



ローカル

ローカルリポジトリ



コミット1

20170520_01



スタッシュのコマンド

作業を一次避難する

>_ ターミナル

```
~ $ git stash
```

```
~ $ git stash save
```

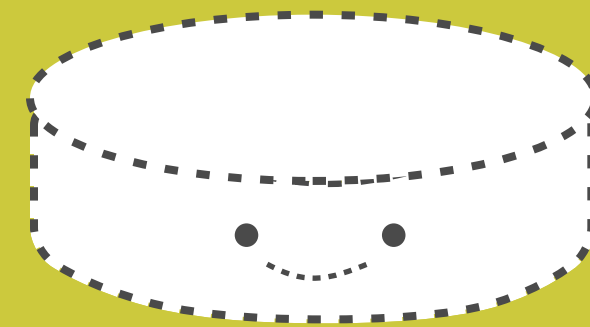
stashは「隠す」という意味だよ

stash

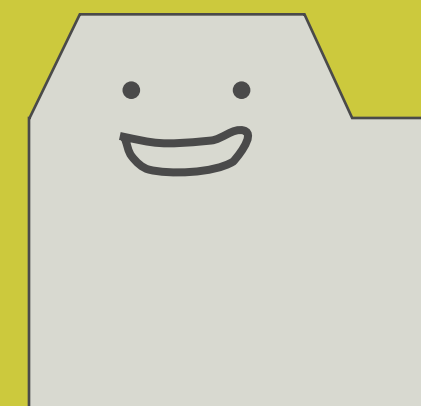
変更分をstashに一時避難する

git stash

ステージ



ワークツリー



top.html

~~変更~~

index.html

~~変更~~

避難した作業を確認する

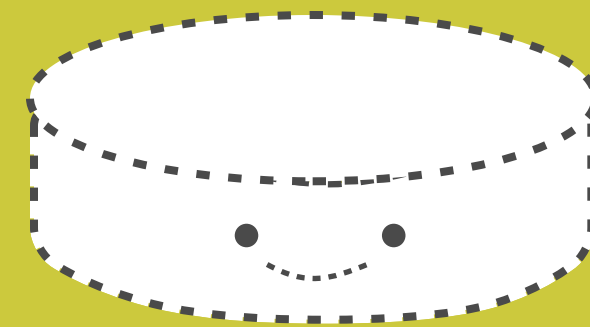
>_ ターミナル

~ \$ git stash list

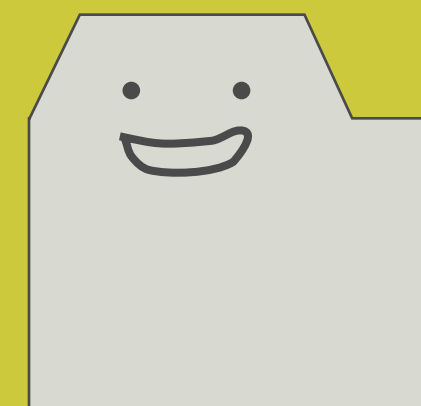
避難した作業の一覧を表示するよ

stash

ステージ



ワークツリー



避難作業1

避難作業2

top.html
~~変更~~

top.html
index.html
~~変更~~

index.html
~~変更~~

避難した作業を復元する

>_ ターミナル

```
# 最新の作業を復元する
~ $ git stash apply
# ステージの状況も復元する
~ $ git stash apply --index

# 特定の作業を復元する
~ $ git stash apply [スタッシュ名]
~ $ git stash apply stash@{1}
```

applyは適用するということだよ

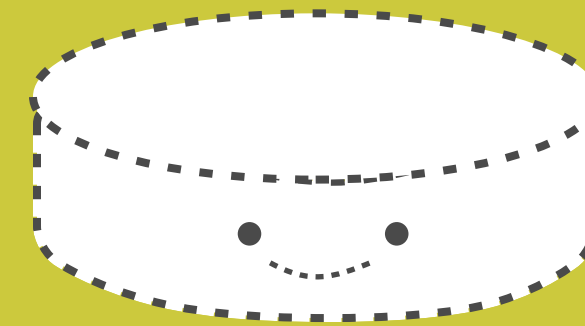
stash

避難作業1
stash@{1}

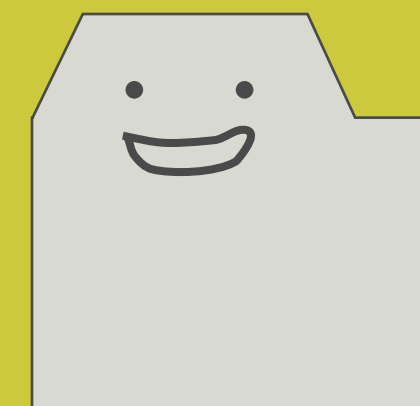
避難作業2
stash@{0}

git stash apply --index

ステージ



ワークツリー



top.html
~~変更~~

top.html
index.html
~~変更~~

index.html
~~変更~~

避難した作業を削除する

>_ ターミナル

最新の作業を削除する
~ \$ git stash drop

特定の作業を削除する
~ \$ git stash drop [スタッシュ名]
~ \$ git stash drop stash@{1}

全作業を削除する
~ \$ git stash clear

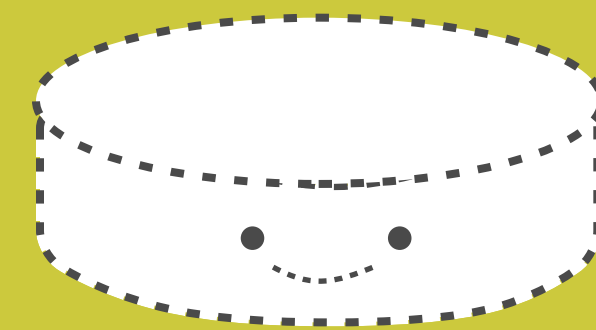
stash

git stash drop

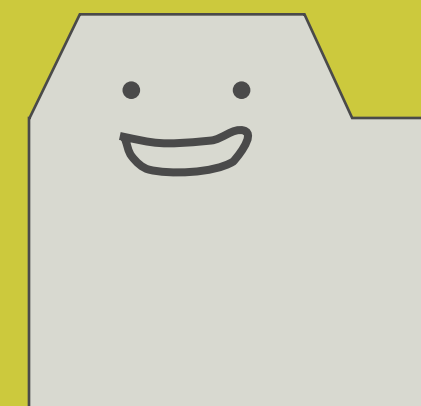
~~避難作業1
stash@{1}~~

~~避難作業2
stash@{0}~~

ステージ



ワークツリー



top.html
~~変更~~

top.html
index.html
~~変更~~

index.html
~~変更~~