

# Git入門

SecPrj Intro-phase

# Gitとは

**Git**とは、**オープンソースの分散バージョン管理システム**の一つ。複数の開発者が共同で一つのソフトウェアを開発する際などに、ソースコードや**ドキュメント**などの編集履歴を統一的に管理するのに用いられる。

Gitとは - IT用語辞典 e-Words <https://e-words.jp/w/Git.html>



**GitHub**



**GitBucket**

GitLab



**Bitbucket**

# ソフトウェアのバージョン管理

Gitがない世界

ファイル名	最終変更日
app.py	2021/03/01
app_ver2.py	2021/05/20
app_ver3.py	2021/05/10
app_latest.py	2021/06/02
app_latest2.py	2021/05/30



で、結局どれが  
最新なんすか？

# ソフトウェアのバージョン管理

## Gitがない世界

ファイル名	最終変更日
app.py	2021/03/01
app_ver2.py	2021/05/20
app_ver3.py	2021/05/10
app_latest.py	2021/06/02
app_latest2.py	2021/05/30



で、結局どれが  
最新なんすか？

## Gitがある世界

ファイル名	最終変更日
.git/ app.py	2021/06/02 2021/06/02

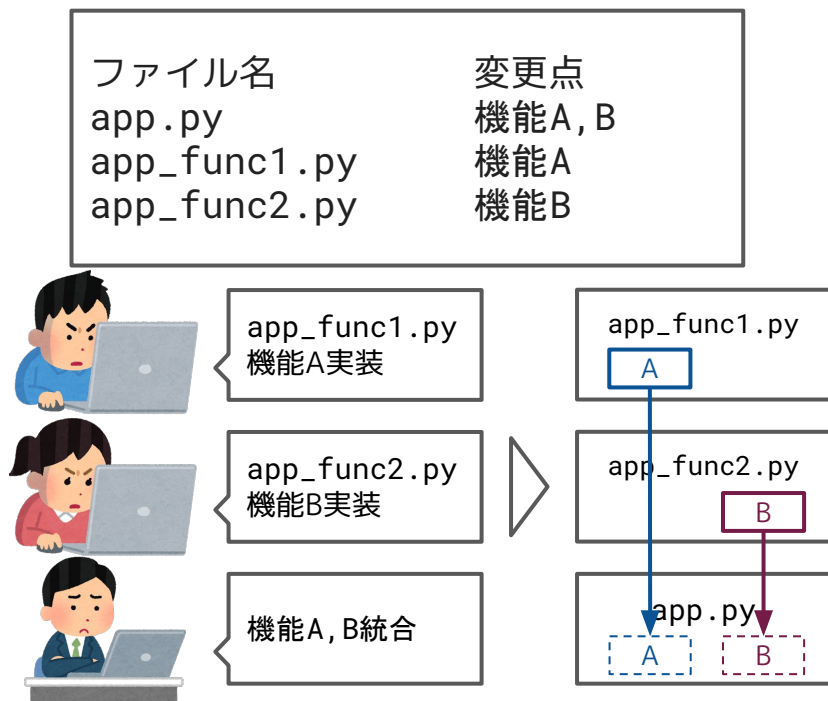
## 変更履歴（コミットログ）



任意の変更点を  
打ち消したり  
戻したりできる

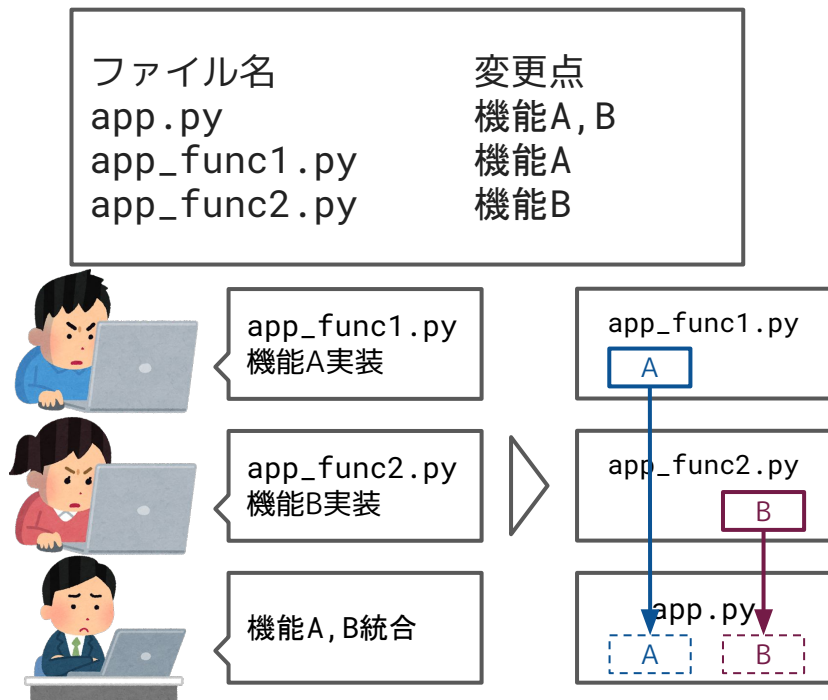
# 複数人での作業

Gitがない世界

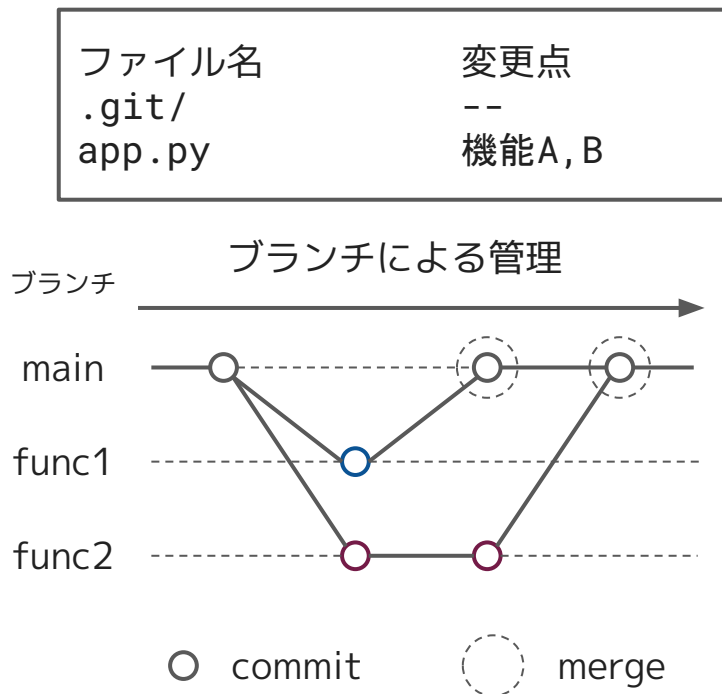


# 複数人での作業

Gitがない世界



Gitがある世界

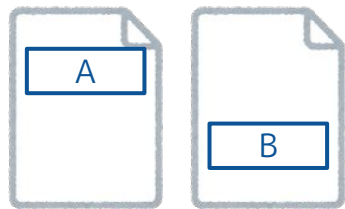


# Gitの基本的な操作

- ステージング (add)
- コミット (commit)
- ステータス (status)
- ブランチ (branch)
  - 移動 (switch)
  - マージ (merge)
- プッシュ (push)
- プル (pull)
- クローン (clone)

# Gitでコミットする

コミットツリー



未コミットの変更点A,B  
があるファイル群

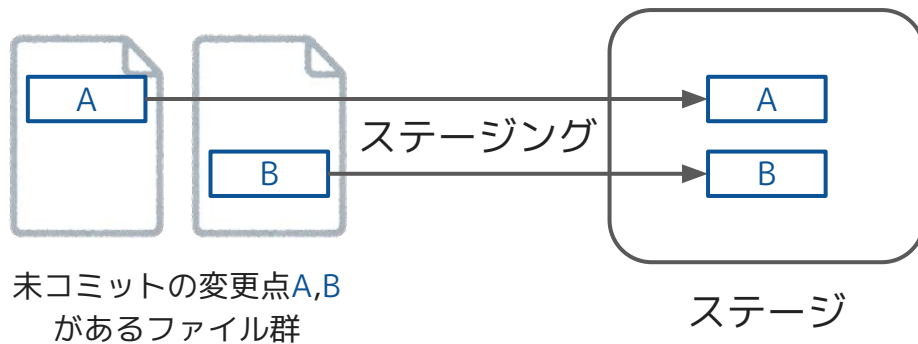


ステージ

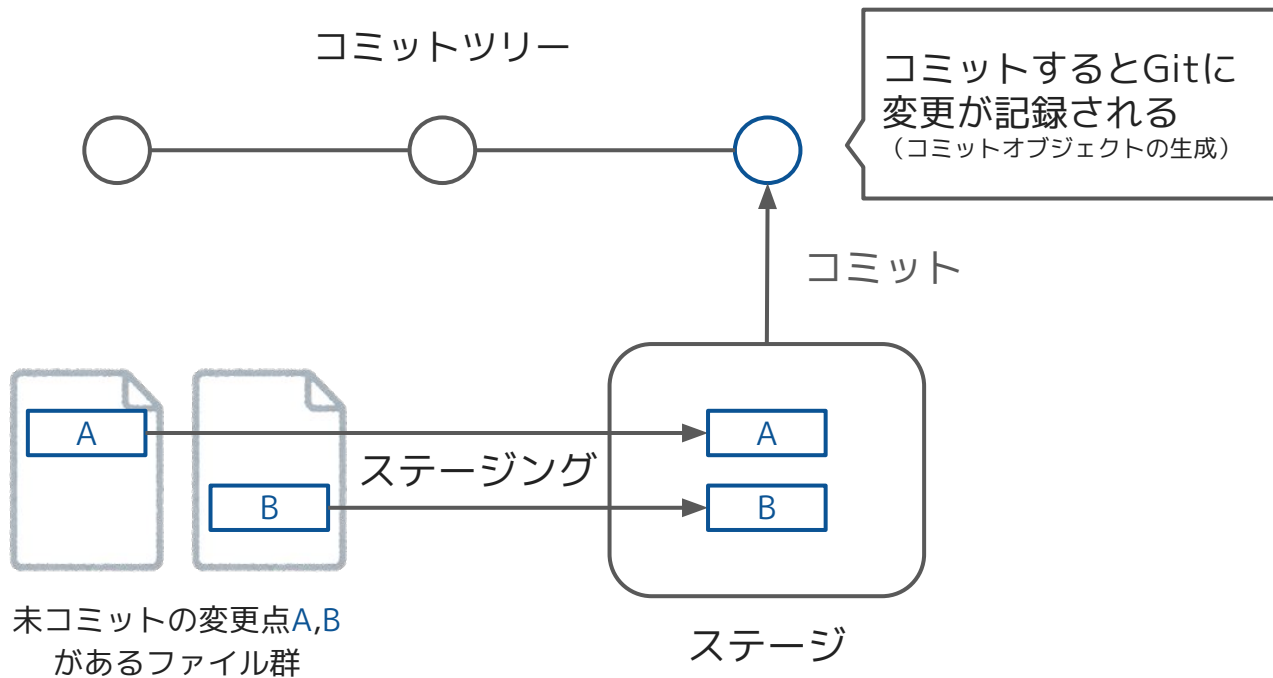


# Gitでコミットする

コミットツリー



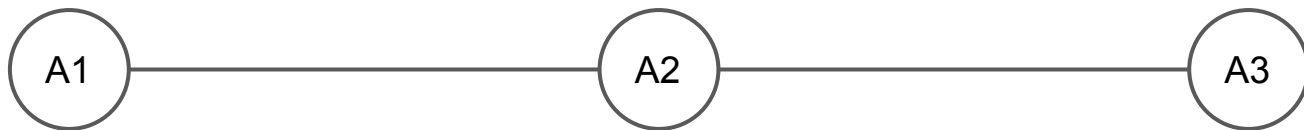
# Gitでコミットする



# Gitのコミット（コミットオブジェクト）

コミットはスナップショット（その時点のディレクトリ／ファイルのコピー）

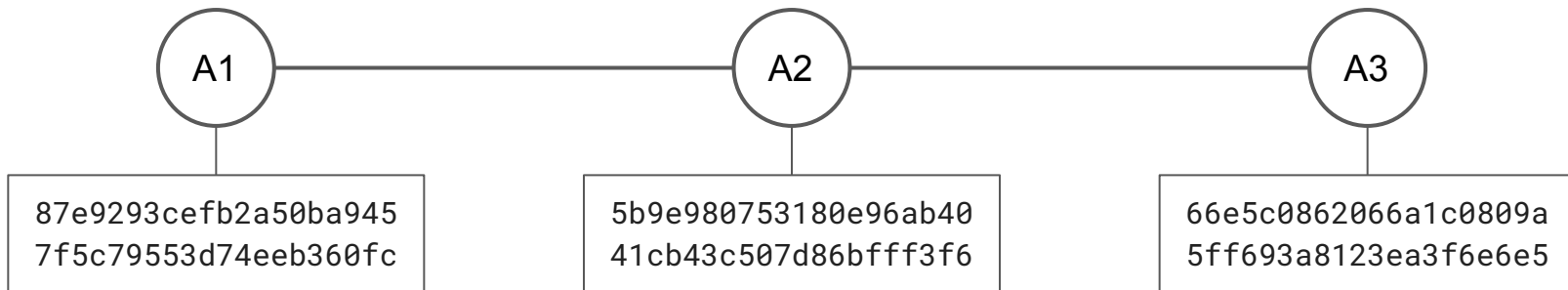
コミットしたファイルと各種情報を持つオブジェクトがコミットオブジェクト



# Gitのコミット（コミットオブジェクト）

コミット（コミットオブジェクト）にはそれぞれコミットハッシュとよばれるSHA1ハッシュ（40文字）が振られる

コミットハッシュでコミット（コミットオブジェクト）を一意に指定できる

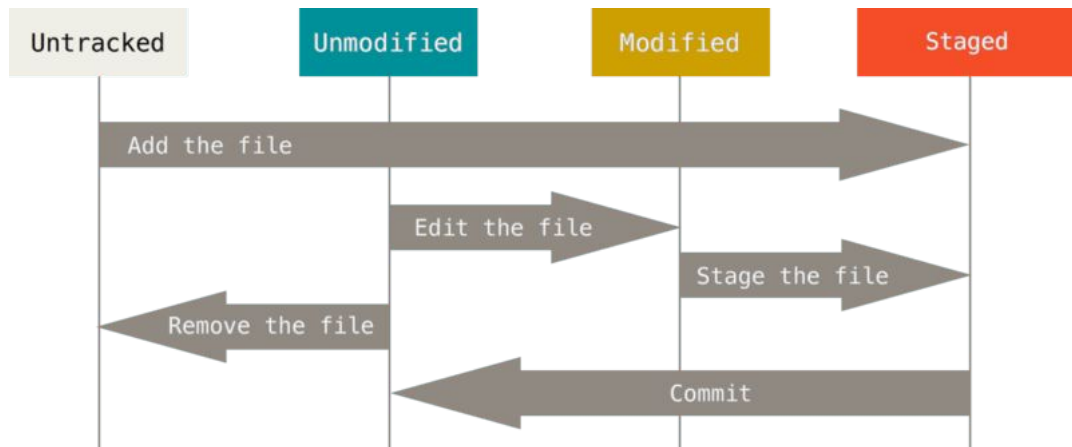


# Gitにおけるファイルのステータス

ファイルには4つの状態がある

Untracked（追跡されてない） / Unmodified（変更なし） /

Modified（変更あり / ステージングされていない） / Staged（ステージングされている）

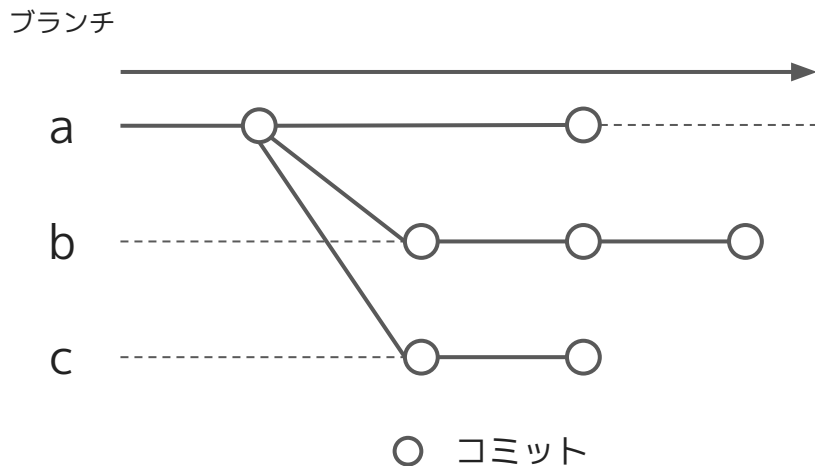


# Gitでブランチを使う

ブランチはいわば「世界線」。あるブランチは他ブランチの変更に影響されない

ex) 関数Aを実装した世界線 / 関数Bを実装した世界線

実装する機能やバージョンごとにブランチを切るのが一般的



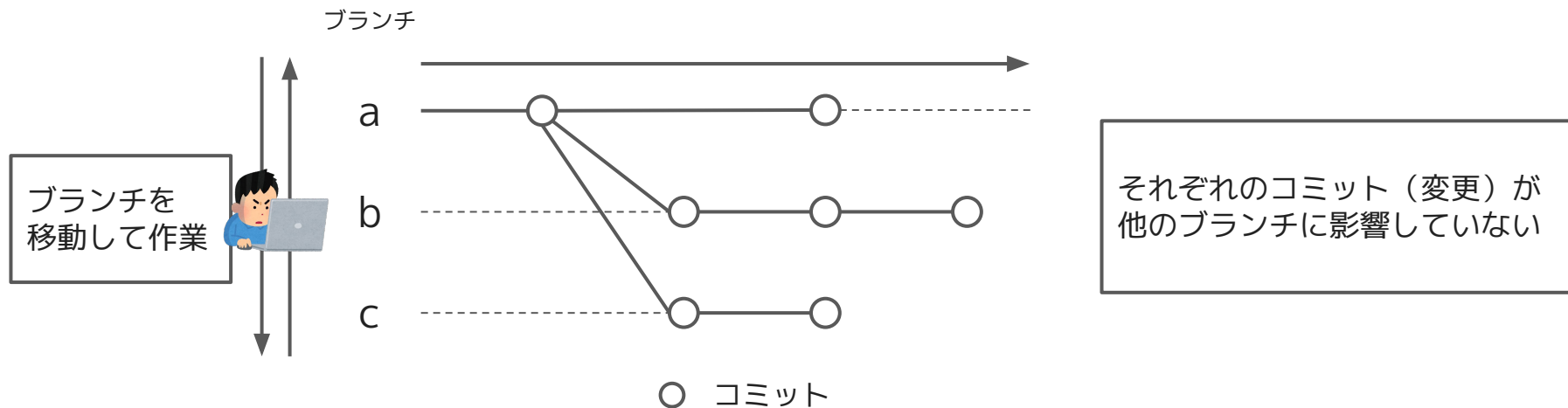
それぞれのコミット（変更）が  
他のブランチに影響していない

# Gitでブランチを使う

ブランチはいわば「世界線」。あるブランチは他ブランチの変更に影響されない

ex) 関数Aを実装した世界線 / 関数Bを実装した世界線

実装する機能やバージョンごとにブランチを切るのが一般的



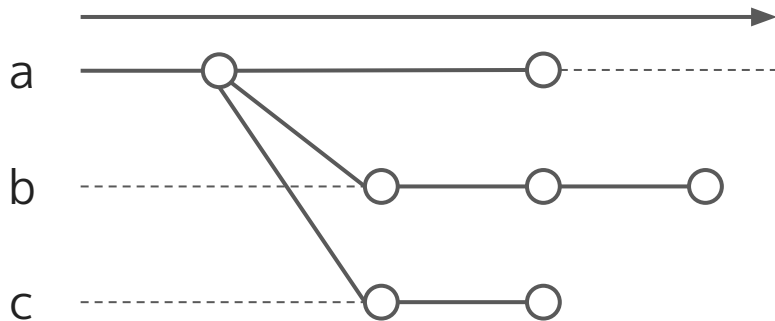
# Gitでブランチを使う

ブランチはいわば「世界線」。あるブランチは他ブランチの変更に影響されない

ex) 関数Aを実装した世界線 / 関数Bを実装した世界線

実装する機能やバージョンごとにブランチを切るのが一般的

ブランチ



ブランチを  
割振って作業



それぞれのコミット（変更）が  
他のブランチに影響していない

○ コミット



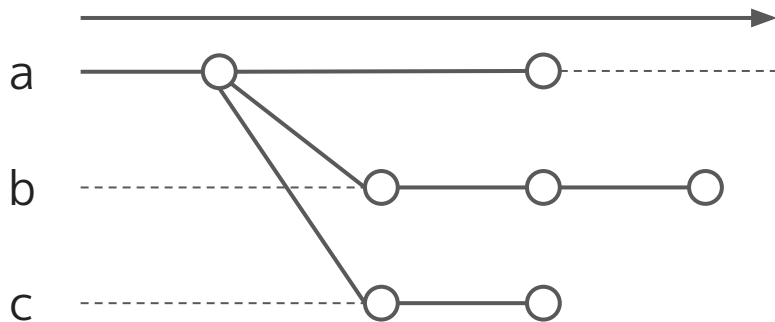
# Gitでブランチを使う

ブランチはいわば「世界線」。あるブランチは他ブランチの変更に影響されない

ex) 関数Aを実装した世界線 / 関数Bを実装した世界線

実装する機能やバージョンごとにブランチを切るのが一般的

ブランチ



実装する機能  
によって分離

A

B

○ コミット

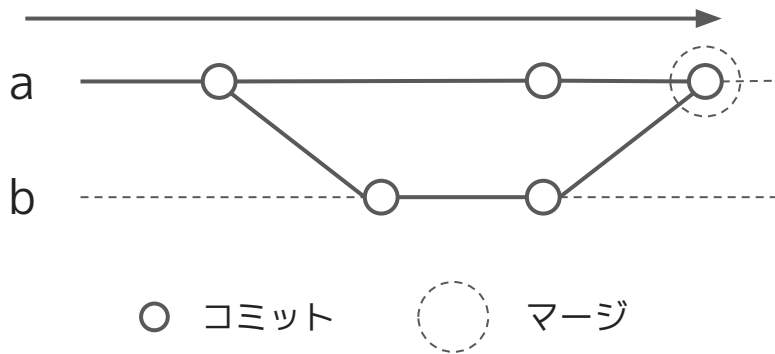
それぞれのコミット（変更）が  
他のブランチに影響していない

# ブランチをマージする

あるブランチの変更と別のブランチを統合／反映する

反映は基本自動で行われるが、変更の重複点のみコンフリクトを起こす

コンフリクトは手動で解消する必要がある。

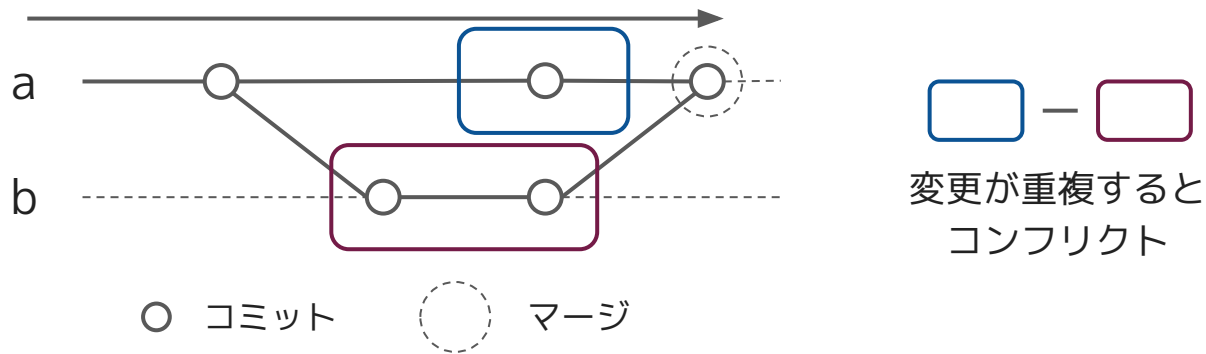


# ブランチをマージする

あるブランチの変更と別のブランチを統合／反映する

反映は基本自動で行われるが、変更の重複点のみコンフリクトを起こす

コンフリクトは手動で解消する必要がある。

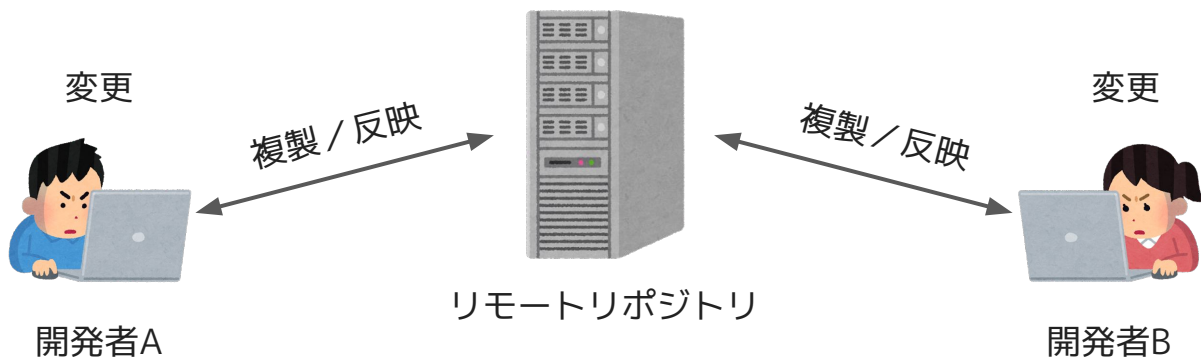


# リポジトリ

Git管理されているディレクトリをリポジトリ (repository) と呼ぶ

ローカルリポジトリ - 手元 (ローカル) にあるリポジトリ

リモートリポジトリ - インターネット上などのサーバ上にあるリポジトリ



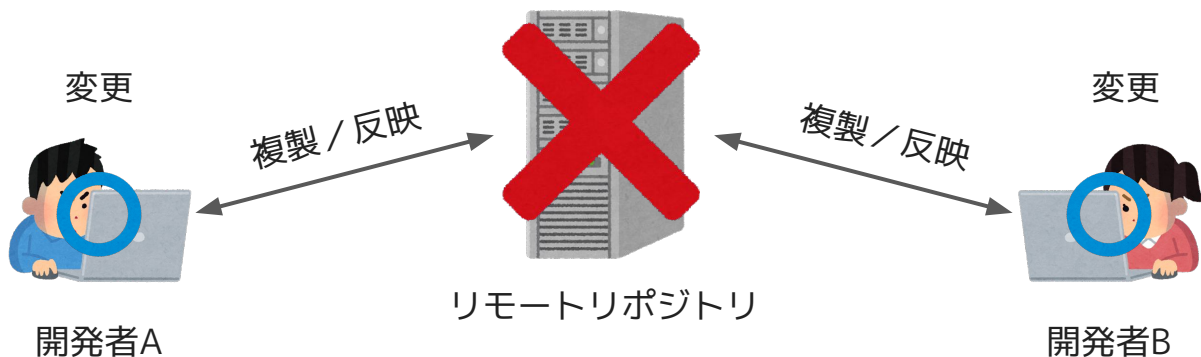
# リポジトリ

Git管理されているディレクトリをリポジトリ (repository) と呼ぶ

ローカルリポジトリ - 手元 (ローカル) にあるリポジトリ

リモートリポジトリ - インターネット上などのサーバ上にあるリポジトリ

複製がどこかにあればリモートリポジトリが死んでもある程度は復元可能



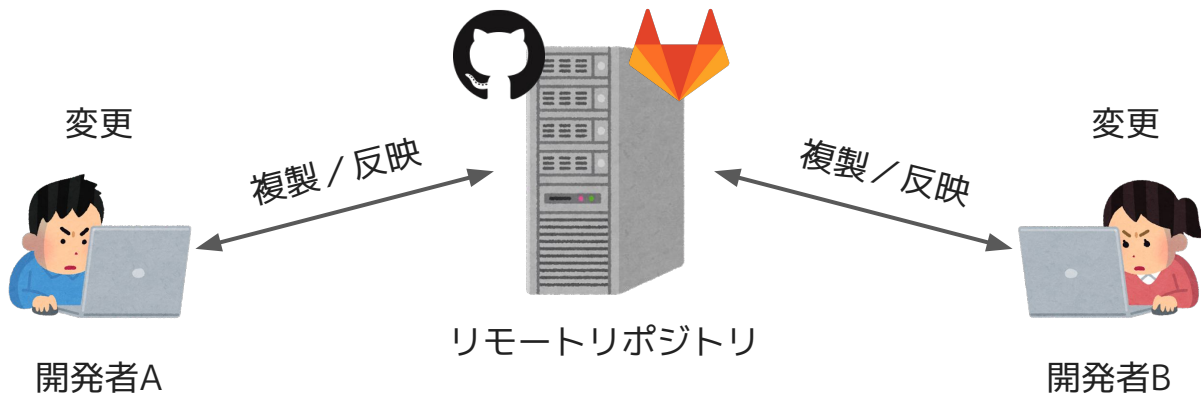
# リポジトリ

Git管理されているディレクトリをリポジトリ (repository) と呼ぶ

ローカルリポジトリ - 手元 (ローカル) にあるリポジトリ

リモートリポジトリ - インターネット上などのサーバ上にあるリポジトリ

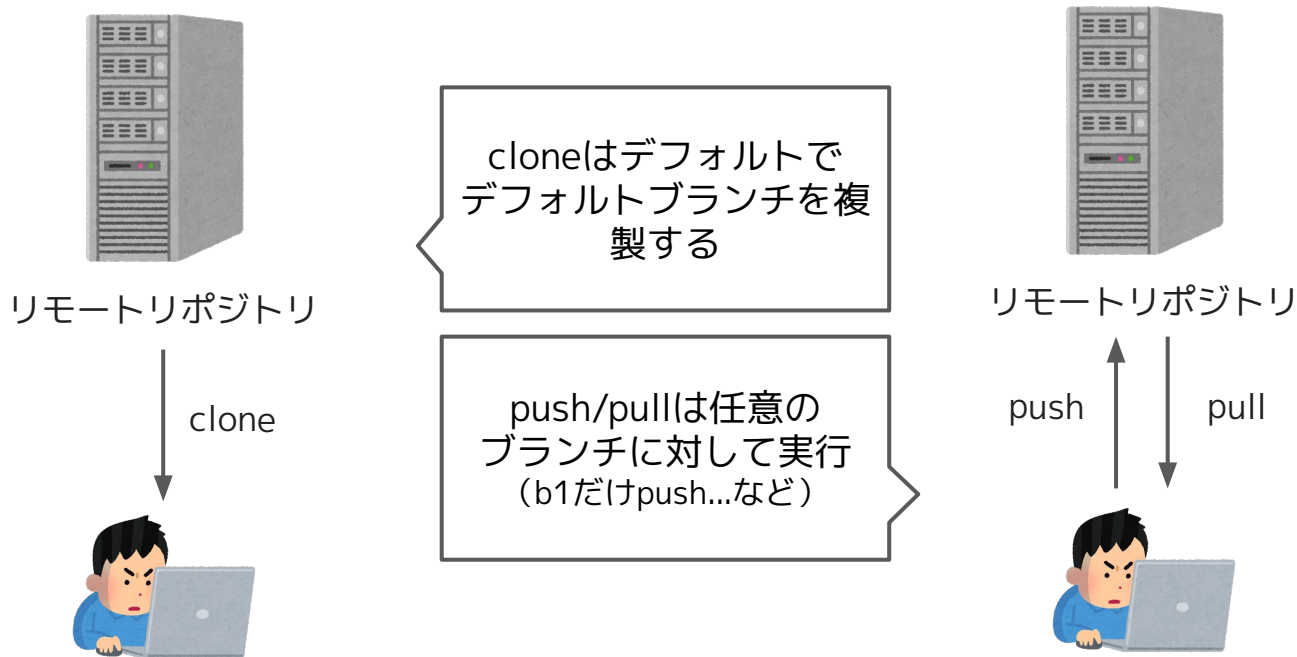
GitHubやGitLabなどはリモートリポジトリの実装の1つ



# リモートリポジトリへの操作

リポジトリを複製（Clone）

ローカルの変更反映／リモートの変更反映



# ハンズオン1

ローカルリポジトリを作成する



# リポジトリを初期化する

ディレクトリにあるファイルはそのまま、  
カレントディレクトリをGit管理下に置く

```
$ git init
```

.git/ が生成され、既にあるファイルは  
追跡されていない未コミットファイルとなる

```
$ git status
```

```
vagrant@ubuntu-focal:~$ mkdir git-test
vagrant@ubuntu-focal:~$ cd git-test/
vagrant@ubuntu-focal:~/git-test$ echo "hello" > hello.txt
vagrant@ubuntu-focal:~/git-test$ ls
hello.txt
vagrant@ubuntu-focal:~/git-test$ git init
Initialized empty Git repository in /home/vagrant/git-test/.git/
vagrant@ubuntu-focal:~/git-test$ ls
hello.txt
vagrant@ubuntu-focal:~/git-test$ ls -a
.  ..  .git  hello.txt
vagrant@ubuntu-focal:~/git-test$ git status
On branch master

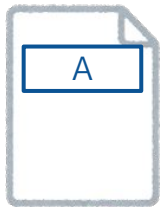
No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
      hello.txt

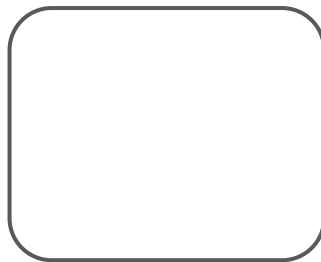
nothing added to commit but untracked files present (use "git add" to track)
vagrant@ubuntu-focal:~/git-test$
```

# Gitでコミットする

コミットツリー



未コミットの変更点A  
があるファイル



ステージ

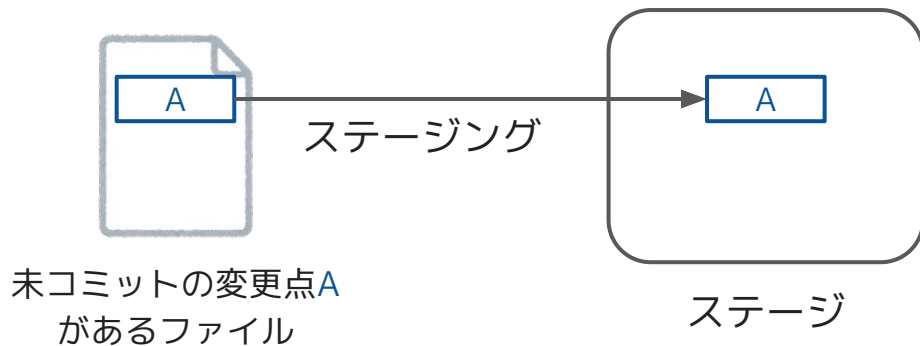
```
$ git diff
```

Untrackedなファイルは差分表示されない

```
vagrant@ubuntu-focal:~/git-test$ git diff  
vagrant@ubuntu-focal:~/git-test$
```

# Gitでコミットする

コミットツリー



```
$ git add <filename>
```

```
vagrant@ubuntu-focal:~/git-test$ git add hello.txt
vagrant@ubuntu-focal:~/git-test$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   hello.txt

vagrant@ubuntu-focal:~/git-test$
```

# Gitでコミットする

ユーザ情報の設定

コミットユーザの情報として利用される

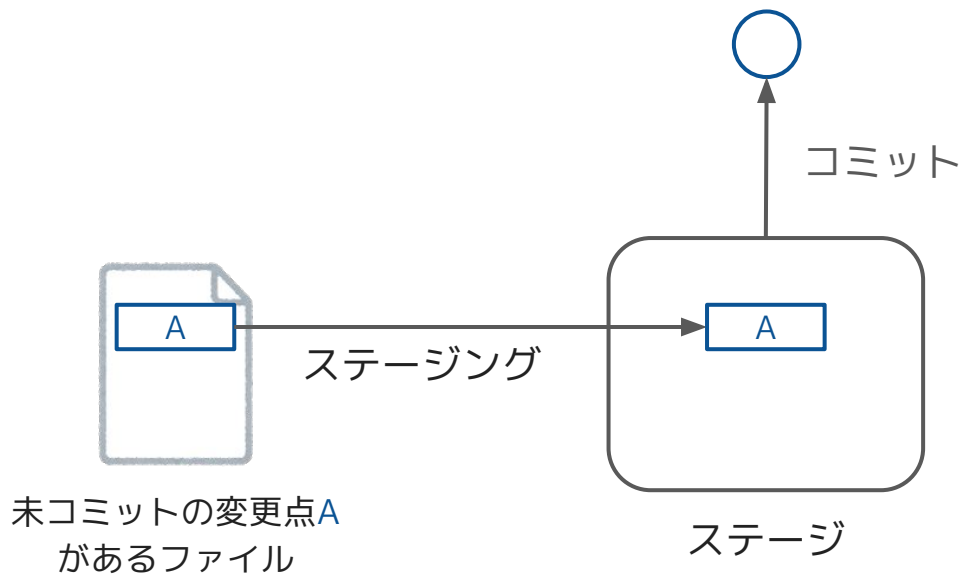
```
$ git config --global user.email "your email"
```

```
$ git config --global user.name "your name"
```

```
$ git config --global core.editor "nano"
```

# Gitでコミットする

コミットツリー



```
$ git commit
```

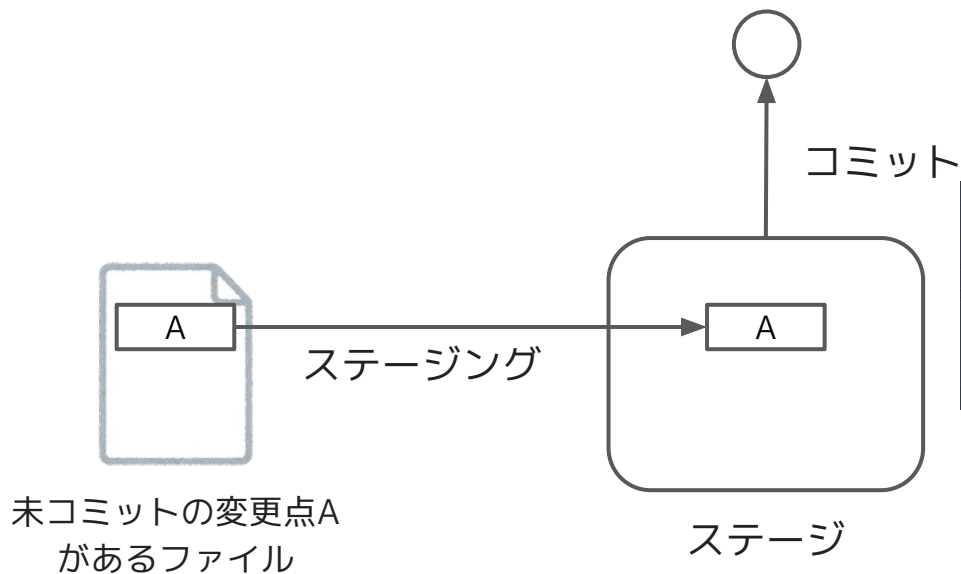
or

```
$ git commit -m "..."
```

```
vagrant@ubuntu-focal:~/git-test$ git commit -m "first commit"
[master (root-commit) 546c781] first commit
1 file changed, 1 insertion(+)
create mode 100644 hello.txt
vagrant@ubuntu-focal:~/git-test$
```

# Gitでコミットする

コミットツリー



\$ git log

```
vagrant@ubuntu-focal:~/git-test$ git log
commit c123cd0c78a1f3266855245cb3027ab306f92a5d (HEAD -> master)
Author: silmin <silvestmint@gmail.com>
Date: Tue Jul 20 15:40:00 2021 +0000

    first commit
vagrant@ubuntu-focal:~/git-test$
```

## Trackedなファイルであれば差分が表示される

```
vagrant@ubuntu-focal:~/git-test$ echo "world" >> hello.txt
vagrant@ubuntu-focal:~/git-test$ git diff
diff --git a/hello.txt b/hello.txt
index ce01362..94954ab 100644
--- a/hello.txt
+++ b/hello.txt
@@ -1,2 @@
 hello
+world
vagrant@ubuntu-focal:~/git-test$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   hello.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

# Gitでブランチを使う

ブランチを確認する

```
$ git branch
```

```
vagrant@ubuntu-focal:~/git-test$ git branch  
* master
```





# Gitでブランチを使う

ブランチを作成する

```
$ git branch b1
```

作成したカレントブランチの情報を引き継ぐ



```
vagrant@ubuntu-focal:~/git-test$ git branch
* master
vagrant@ubuntu-focal:~/git-test$ git branch b1
vagrant@ubuntu-focal:~/git-test$ git branch
b1
* master
vagrant@ubuntu-focal:~/git-test$
```

# Gitでブランチを使う

カレントブランチを移動する

master → b1



```
$ git switch b1
```

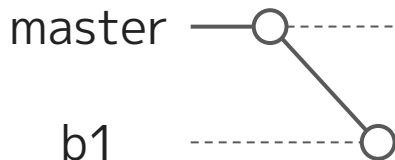
```
vagrant@ubuntu-focal:~/git-test$ git switch b1
Switched to branch 'b1'
vagrant@ubuntu-focal:~/git-test$ git branch
* b1
  master
```

\$ git switch -c b1 で作成&移動も可能

# Gitでブランチを使う

b1で何かを変更し、コミットする

このコミットはmasterに続く形になる  
(b1はmasterから切られているので)



-任意の編集-

```
$ git add <filename>
```

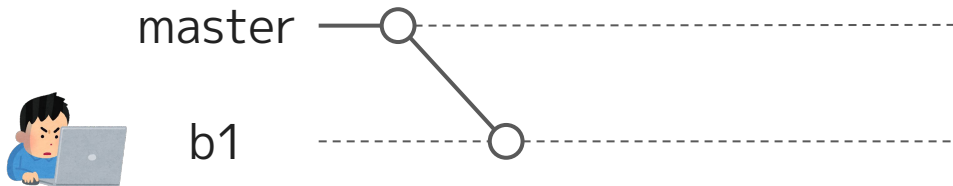
```
$ git commit
```

```
vagrant@ubuntu-focal:~/git-test$ echo "b1 hello" >> hello.txt
vagrant@ubuntu-focal:~/git-test$ git add hello.txt
vagrant@ubuntu-focal:~/git-test$ git commit -m "b1 say hello"
[b1 39cf598] b1 say hello
1 file changed, 2 insertions(+)
```

# Gitでブランチを使う

b1で何かを変更し、コミットする

このコミットはmasterに続く形になる  
(b1はmasterから切られているので)



```
$ git log
```

```
vagrant@ubuntu-focal:~/git-test$ git log
commit 39cf5984753f080addf56868870cb05191e1c816 (HEAD -> b1)
Author: silmin <silvestmint@gmail.com>
Date: Tue Jul 20 15:41:51 2021 +0000

    b1 say hello

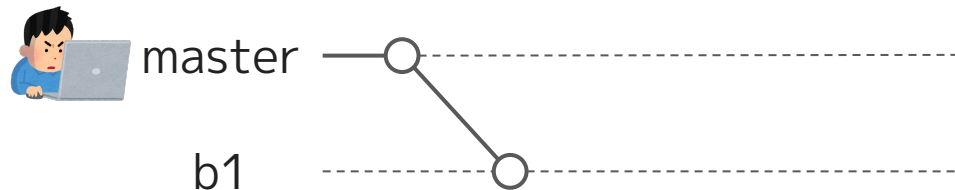
commit c123cd0c78a1f3266855245cb3027ab306f92a5d (master)
Author: silmin <silvestmint@gmail.com>
Date: Tue Jul 20 15:40:00 2021 +0000

    first commit
```

# Gitでブランチを使う

カレントブランチをmasterに移動

1つめのコミットしかみえない



```
$ git switch master
```

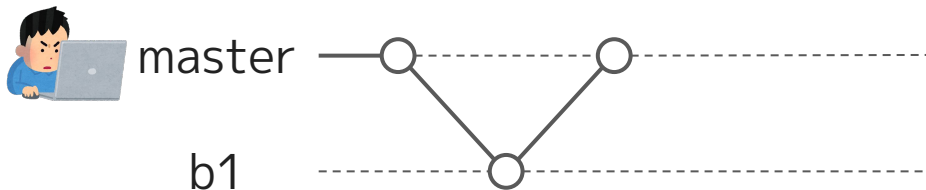
```
vagrant@ubuntu-focal:~/git-test$ git switch master
Switched to branch 'master'
vagrant@ubuntu-focal:~/git-test$ git log
commit c123cd0c78a1f3266855245cb3027ab306f92a5d (HEAD -> master)
Author: silmin <silvestmint@gmail.com>
Date: Tue Jul 20 15:40:00 2021 +0000

    first commit
```

# Gitでブランチを使う

b1ブランチをmasterブランチにマージ

変更がb1でコミットした変更が反映されている `$ git merge b1`



```
vagrant@ubuntu-focal:~/git-test$ git log
commit 39cf5984753f080addf56868870cb05191e1c816 (HEAD -> master, b1)
Author: silmin <silvestmint@gmail.com>
Date: Tue Jul 20 15:41:51 2021 +0000

    b1 say hello

commit c123cd0c78a1f3266855245cb3027ab306f92a5d
Author: silmin <silvestmint@gmail.com>
Date: Tue Jul 20 15:40:00 2021 +0000

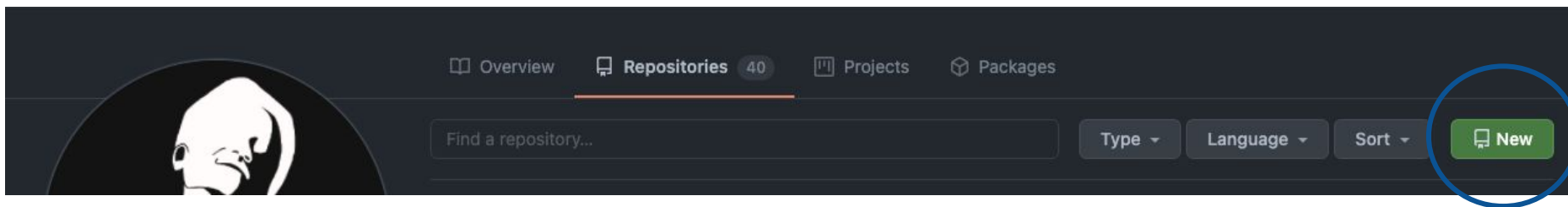
    first commit
```

# ハンズオン 2

リモートリポジトリの作成 / Clone / Push / Pull

# リモートリポジトリのクローン, プッシュ, プル

まず自分のGitHubページで新しいリポジトリを作る






# リモートリポジトリのクローン, プッシュ, プル

## Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner \*

Repository name \*

 silmin

 /

Great repository names are short and memorable. Need inspiration? How about [supreme-pancake?](#)

Description (optional)

☒  **Public**

Anyone on the internet can see this repository. You choose who can commit.

☐  **Private**

You choose who can see and commit to this repository.

**Initialize this repository with:**

Skip this step if you're importing an existing repository.

☐ **Add a README file**

This is where you can write a long description for your project. [Learn more.](#)

☐ **Add .gitignore**

Choose which files not to track from a list of templates. [Learn more.](#)

☐ **Choose a license**

A license tells others what they can and can't do with your code. [Learn more.](#)


Create repository

## Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner \*

Repository name \*

 silmin

 / 

git-test ✓

Great repository names are short and memorable. Need inspiration? How about [supreme-pancake?](#)

Description (optional)

☒  **Public**

Anyone on the internet can see this repository. You choose who can commit.

☐  **Private**

You choose who can see and commit to this repository.

**Initialize this repository with:**

Skip this step if you're importing an existing repository.

☒ **Add a README file**

This is where you can write a long description for your project. [Learn more.](#)

☐ **Add .gitignore**

Choose which files not to track from a list of templates. [Learn more.](#)

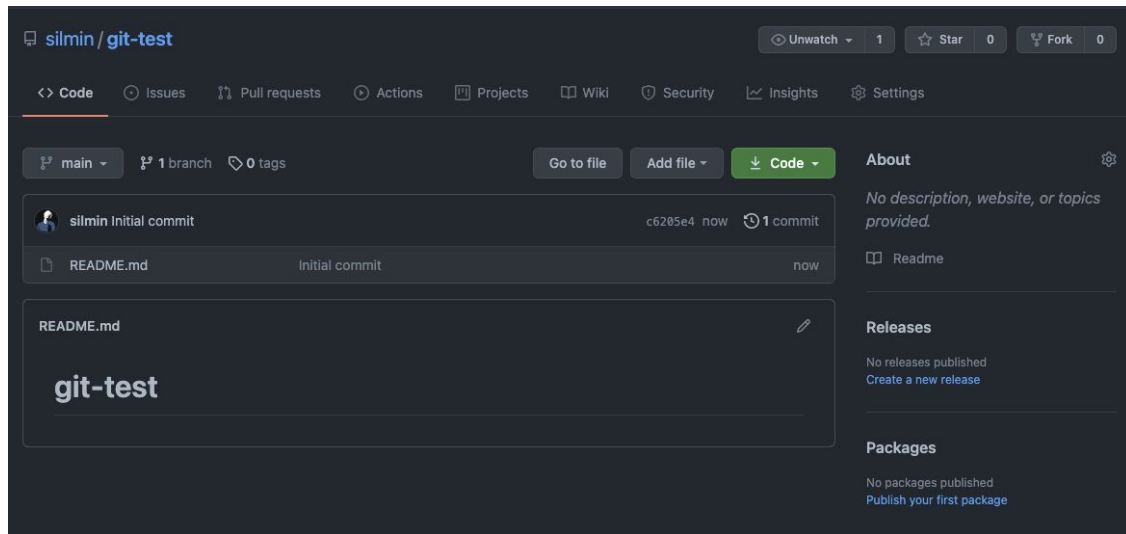
☐ **Choose a license**

A license tells others what they can and can't do with your code. [Learn more.](#)

This will set  **main** as the default branch. Change the default name in your [settings](#).

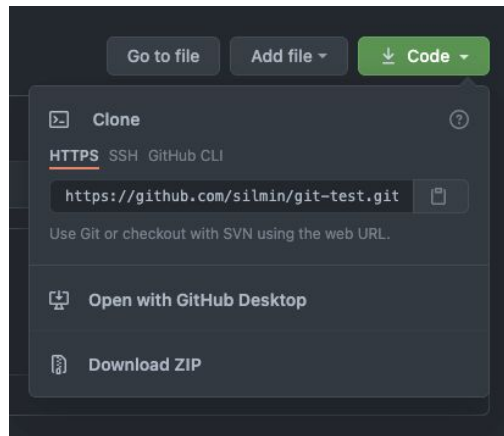
Create repository

# リモートリポジトリのクローン, プッシュ, プル



最初のコミットがされた状態でリモートリポジトリが生成される

# リモートリポジトリのクローン, プッシュ, プル



```
$ git clone https://github.com/silmin/git-test.git
```

```
▶ git clone https://github.com/silmin/git-test
Cloning into 'git-test'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.
```

# リモートリポジトリのクローン, プッシュ, プル

cloneしてきたリポジトリに変更点を加えて, コミットする

```
▶ cd git-test
~/git-test ♪ main
▶ ls
README.md
~/git-test ♪ main
▶ nv README.md
~/git-test ♪ main*
▶ git diff
diff --git a/README.md b/README.md
index e87f6b7..4d37b8d 100644
--- a/README.md
+++ b/README.md
@@ -1,1,3 @@
-# git-test
\ No newline at end of file
+# git-test
+
+hello
```

```
~/git-test ♪ main*
▶ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")

~/git-test ♪ main*
▶ git add README.md

~/git-test ♪ main*
▶ git commit -m "say hello"
[main 041056d] say hello
1 file changed, 3 insertions(+), 1 deletion(-)
```

# リモートリポジトリのクローン, プッシュ, プル

```
~/git-test ʘ main
▶ git log
commit 041056d67cc7ac16af81b35b16f25a40dff4316c (HEAD -> main, origin/main, origin/HEAD)
Author: silmin <silvestmint@gmail.com>
Date:   Wed Jul 21 00:20:52 2021 +0900

    say hello

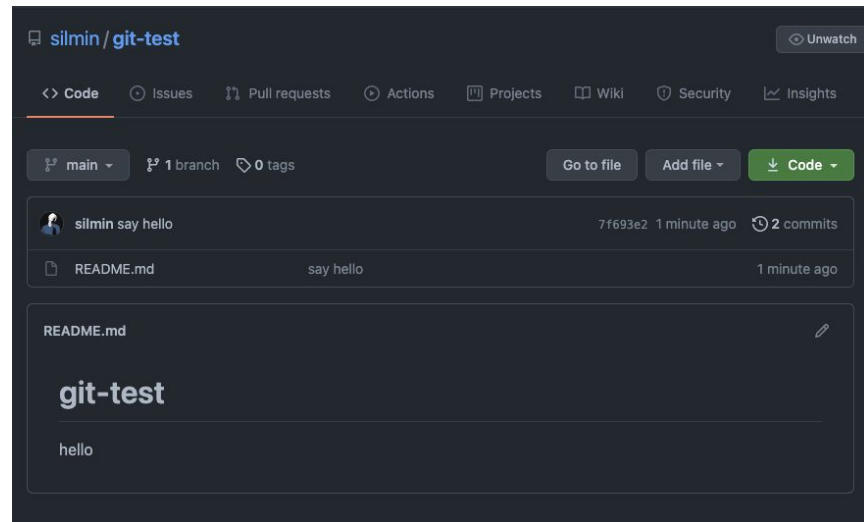
commit ccf32a37d3054034998668411af0f508daec7369
Author: silmin <32303837+silmin@users.noreply.github.com>
Date:   Wed Jul 21 00:18:47 2021 +0900

    Initial commit
```

# リモートリポジトリのクローン, プッシュ, プル

```
$ git push origin main
```

```
~/git-test P main ↑  
▶ git push origin main  
Enumerating objects: 5, done.  
Counting objects: 100% (5/5), done.  
Writing objects: 100% (3/3), 257 bytes | 257.00 KiB/s, done.  
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0  
To https://github.com/silmin/git-test  
ccf32a3..041056d  main -> main
```



helloが追加されている

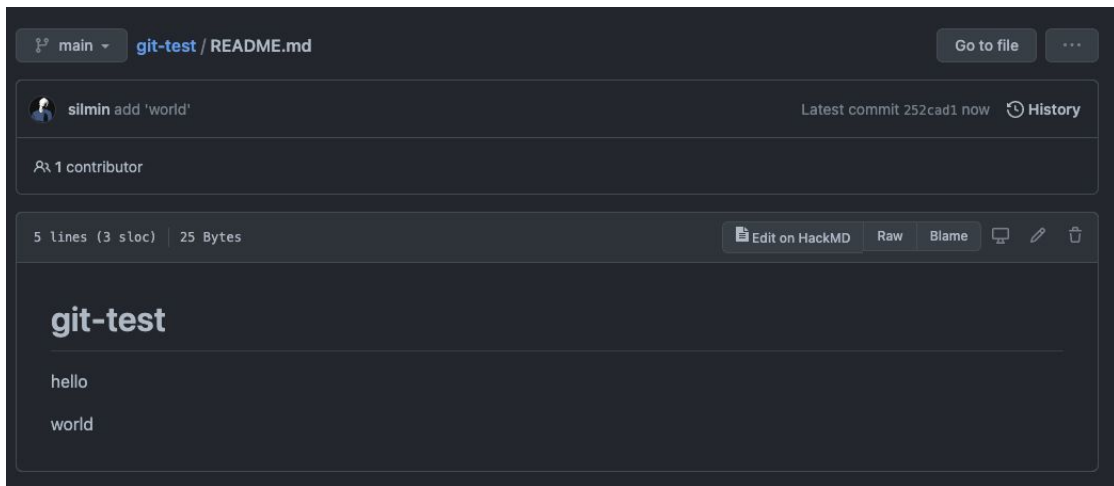
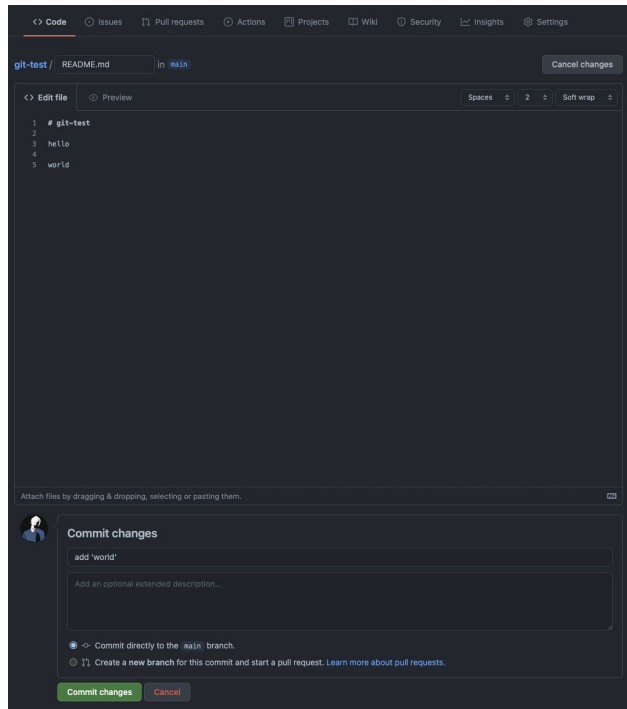
ブランチ

```
$ git push origin main
```

リモートリポジトリ

# リモートリポジトリのクローン, プッシュ, プル

WebUIから直接変更してみる





# リモートリポジトリのクローン, プッシュ, プル

リモートでの変更をローカルにプルして反映

```
~/git-test % main ↓  
▶ git pull  
Updating 041056d..252cad1  
Fast-forward  
 README.md | 2 ++  
 1 file changed, 2 insertions(+)
```

```
~/git-test % main  
▶ git log  
commit 252cad19c88548e7e2cd36d0b349dfa71853722d (HEAD -> main, origin/main, origin/HEAD)  
Author: silmin <32303837+silmin@users.noreply.github.com>  
Date:   Wed Jul 21 00:25:58 2021 +0900  
  
    add 'world'  
  
commit 041056d67cc7ac16af81b35b16f25a40dff4316c  
Author: silmin <silvestmint@gmail.com>  
Date:   Wed Jul 21 00:20:52 2021 +0900  
  
    say hello  
  
commit ccf32a37d3054034998668411af0f508daec7369  
Author: silmin <32303837+silmin@users.noreply.github.com>  
Date:   Wed Jul 21 00:18:47 2021 +0900  
  
    Initial commit
```

ハンズオンおわり

# gitconfigについて

それぞれのリポジトリにはその設定ファイルとして `.git/config` がある  
全てのリポジトリに適用されるグローバルな設定 `~/.gitconfig` もある

```
1 [core]
2     repositoryformatversion = 0
3     filemode = true
4     bare = false
5     logallrefupdates = true
6     ignorecase = true
7     precomposeunicode = true
8 [remote "origin"]
9     url = https://github.com/silmin/git-test
10    fetch = +refs/heads/*:refs/remotes/origin/*
11 [branch "main"]
12     remote = origin
13     merge = refs/heads/main
```

さっきGitHubで作った  
リポジトリの`.git/config`

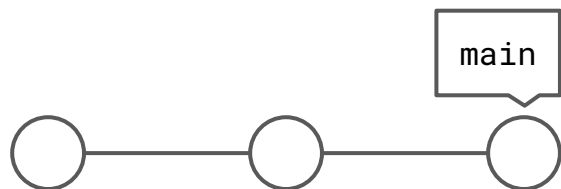
originはリモートリポジトリに  
つけられた名前


# Gitのブランチ

ブランチの実態はただのコミットへの参照（ポインタ）

ブランチの数  $\equiv$  ブランチポインタの数

ブランチのポインタは、それぞれ常にブランチの先頭コミットを参照している



 ブランチポインタ

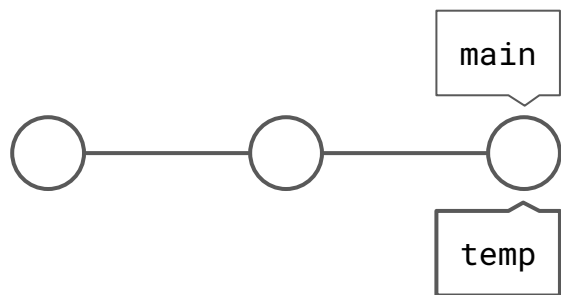
 カレントブランチ（HEAD）

# Gitのブランチ

ブランチの実態はただのコミットへの参照（ポインタ）

ブランチの数 ≡ ブランチポインタの数

ブランチのポインタは、それぞれ常にブランチの先頭コミットを参照している



```
$ git switch -c temp
```

 ブランチポインタ

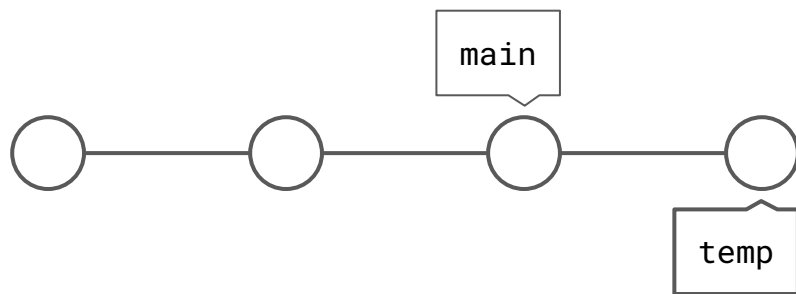
 カレントブランチ（HEAD）

# Gitのブランチ


ブランチの実態はただのコミットへの参照（ポインタ）


ブランチの数 ≡ ブランチポインタの数

ブランチのポインタは、それぞれ常にブランチの先頭コミットを参照している



```
$ git commit ...
```

 ブランチポインタ

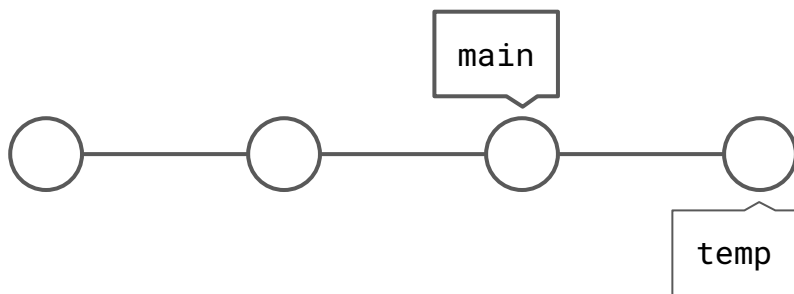
 カレントブランチ（HEAD）

# Gitのブランチ


ブランチの実態はただのコミットへの参照（ポインタ）


ブランチの数 ≡ ブランチポインタの数

ブランチのポインタは、それぞれ常にブランチの先頭コミットを参照している



```
$ git switch main
```

 ブランチポインタ

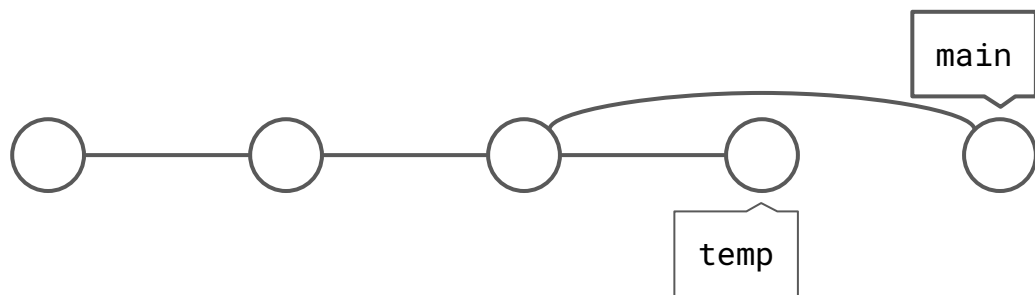
 カレントブランチ（HEAD）

# Gitのブランチ

ブランチの実態はただのコミットへの参照（ポインタ）

ブランチの数 ≡ ブランチポインタの数

ブランチのポインタは、それぞれ常にブランチの先頭コミットを参照している



```
$ git commit ...
```

 ブランチポインタ

 カレントブランチ（HEAD）

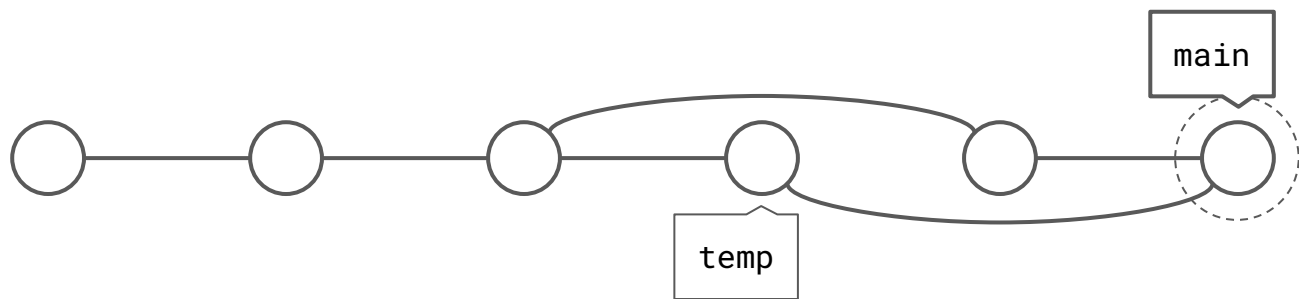


# Gitのブランチ

ブランチの実態はただのコミットへの参照（ポインタ）

ブランチの数 ≡ ブランチポインタの数

ブランチのポインタは、それぞれ常にブランチの先頭コミットを参照している

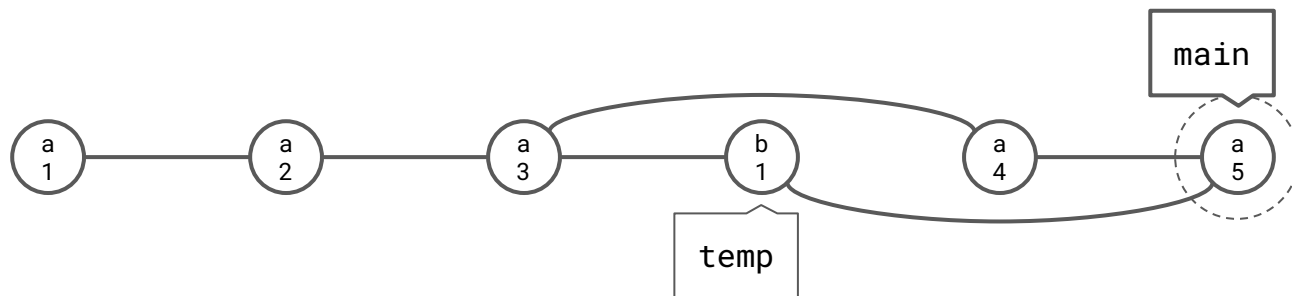
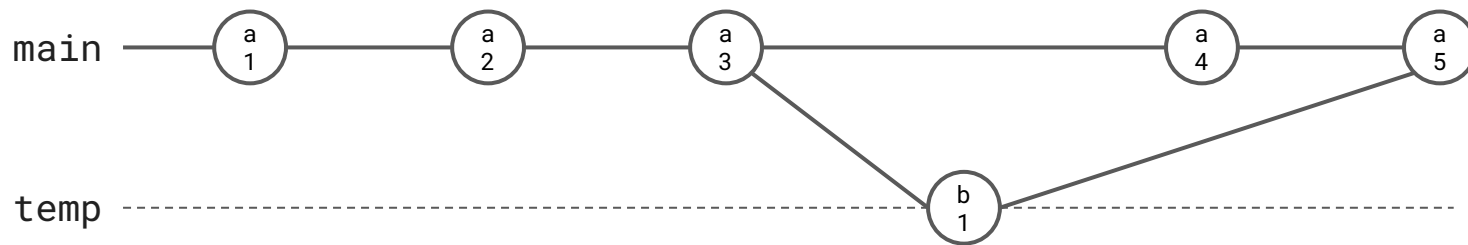


```
$ git merge temp
```

 ブランチポインタ

 カレントブランチ（HEAD）

# Gitのブランチ



- ブランチポインタ
- カレントブランチ (HEAD)