東京大学
THE UNIVERSITY OF TOKYO

統計勉強会第一回 12/4

かわの まこと

今日やること

- **Git ～ソースコードをうまく管理しよう～**
- **第一章 ～PythonDe統計のための準備～**

Gitって？

複数人でプロジェクト開発するときにつかう

- プロジェクト開発に必要なのは：
  - コミュニケーション能力
  - 仕様書
  - 共同開発環境

Gitって？

複数人でプロジェクト開発するときにつかう

- プロジェクト開発に必要なのは：
  - コミュニケーション能力
  - 仕様書
  - **共同開発環境 ← ここでGitの登場**

Gitって？

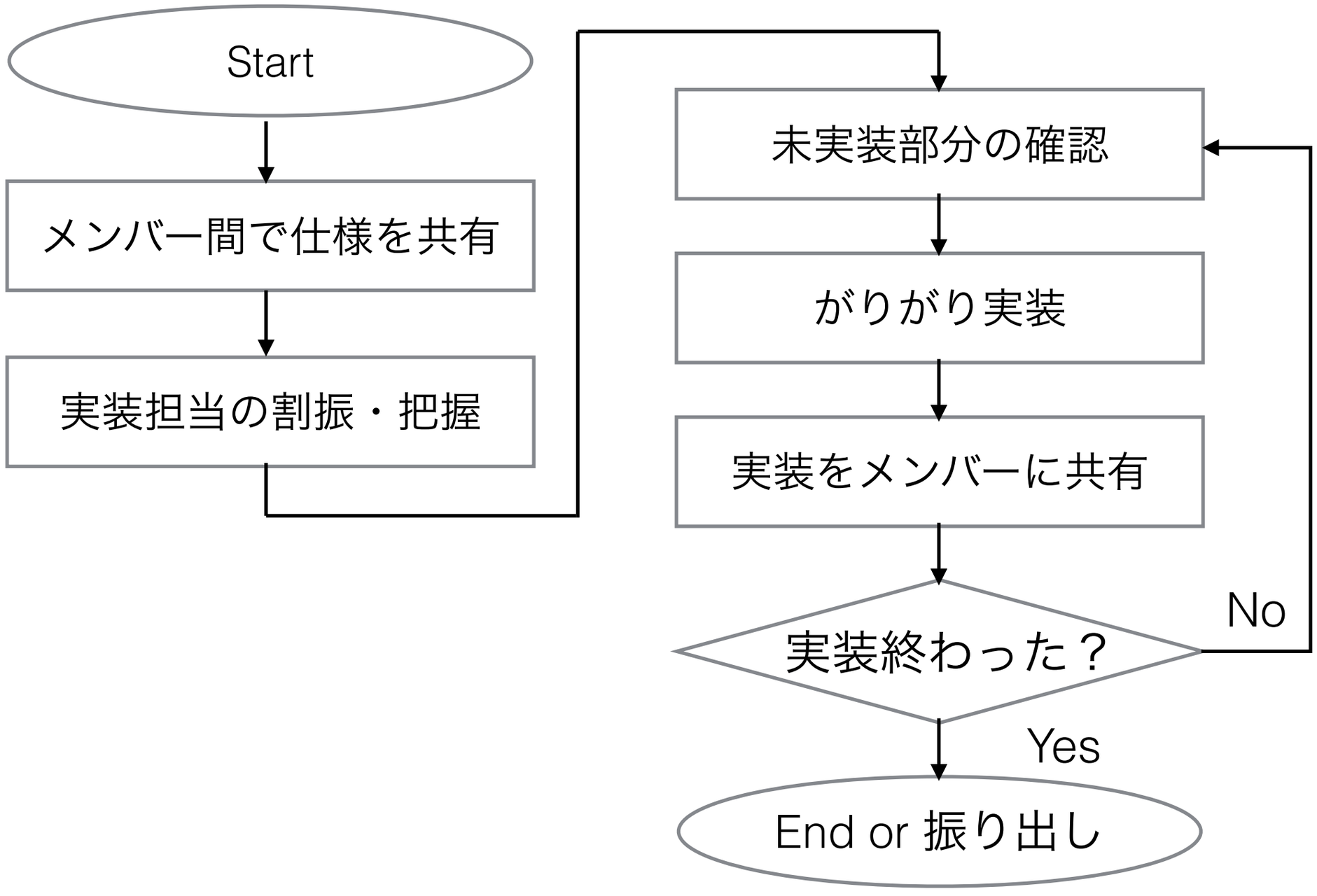
複数人でプロジェクト開発するときにつかう

- プロジェクト開発に必要なのは：
  - コミュニケーション能力
  - 仕様書
  - **共同開発環境 ← ここでGitの登場**

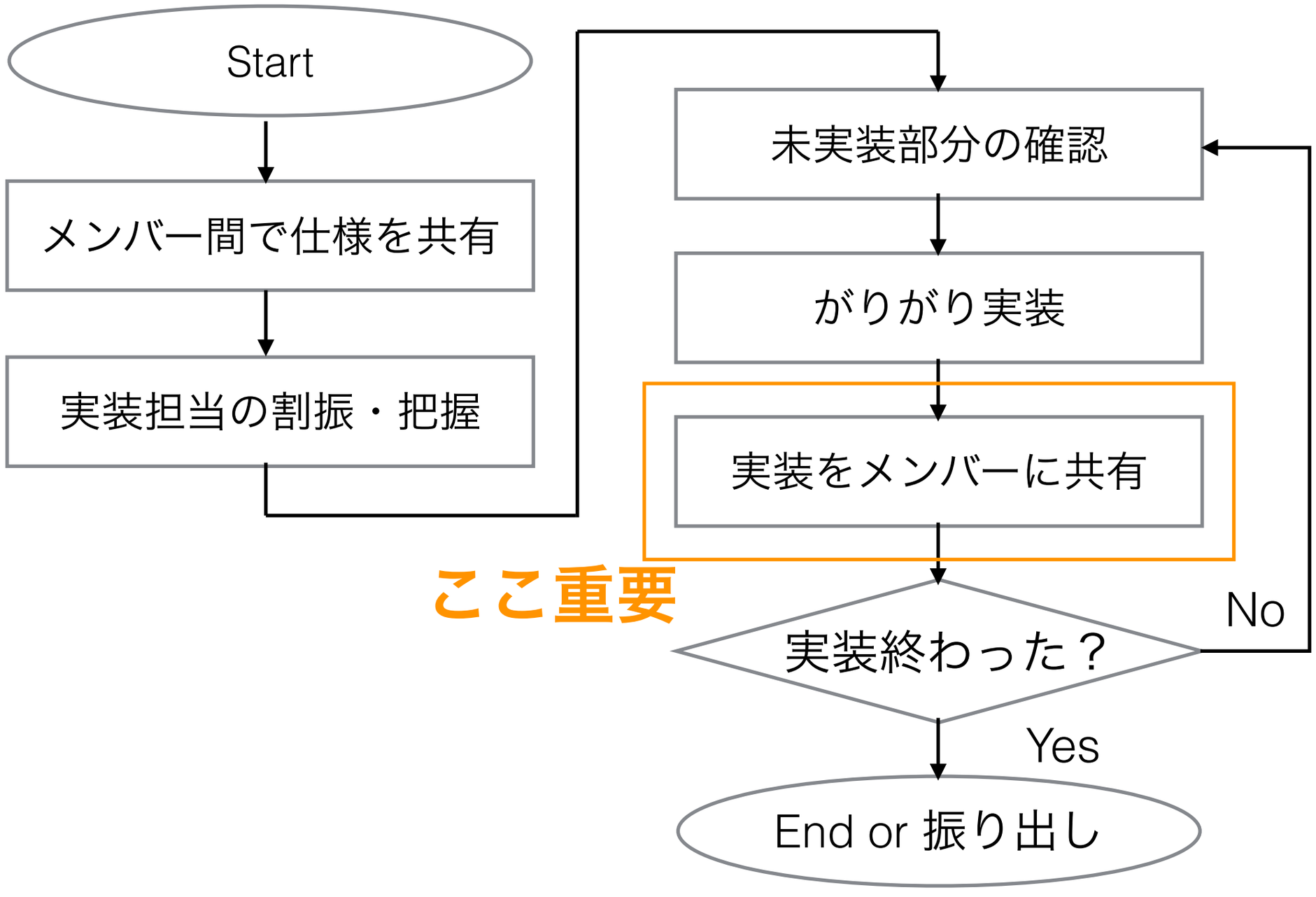
バージョン管理（ソースコードの共有）をして幸せになろう

東京大学 THE UNIVERSITY OF TOKYO

プロジェクト開発の流れ



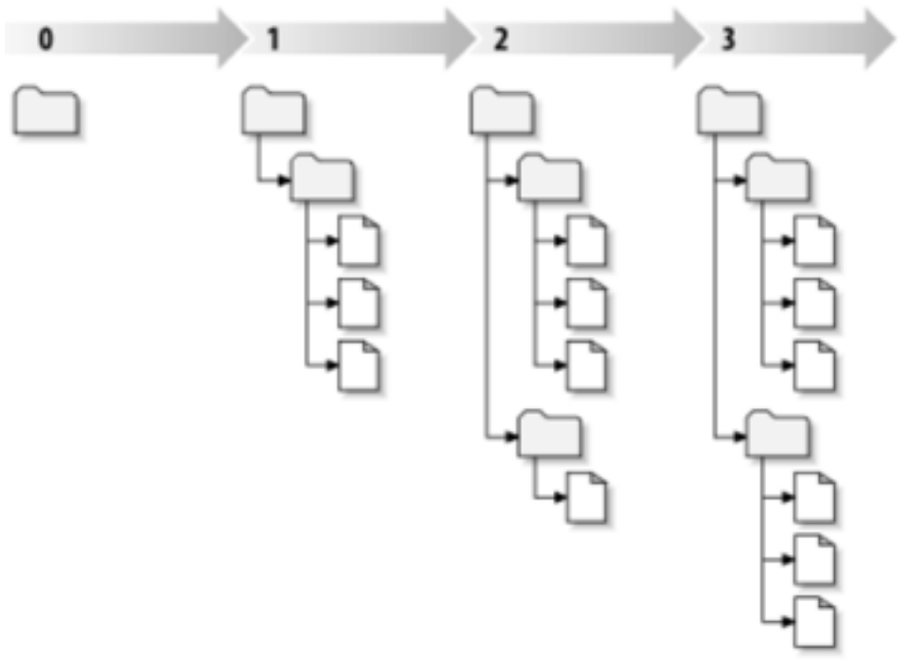
プロジェクト開発の流れ



バージョン管理って

ソースコードの変更ログを残しながら保存

- ファイルの変更点やディレクトリ構造の
  スナップショットをとるイメージ
- サーバに保存しておくとPCが逝っても大丈夫



バージョン管理めんどい

「○○の機能が出来たから添付したソースをプロジェクトに追加しておいて」

- 手間がかかる
- プロジェクトが大きくなるにつれてつらい

「追加・改修したら動かなくなった」

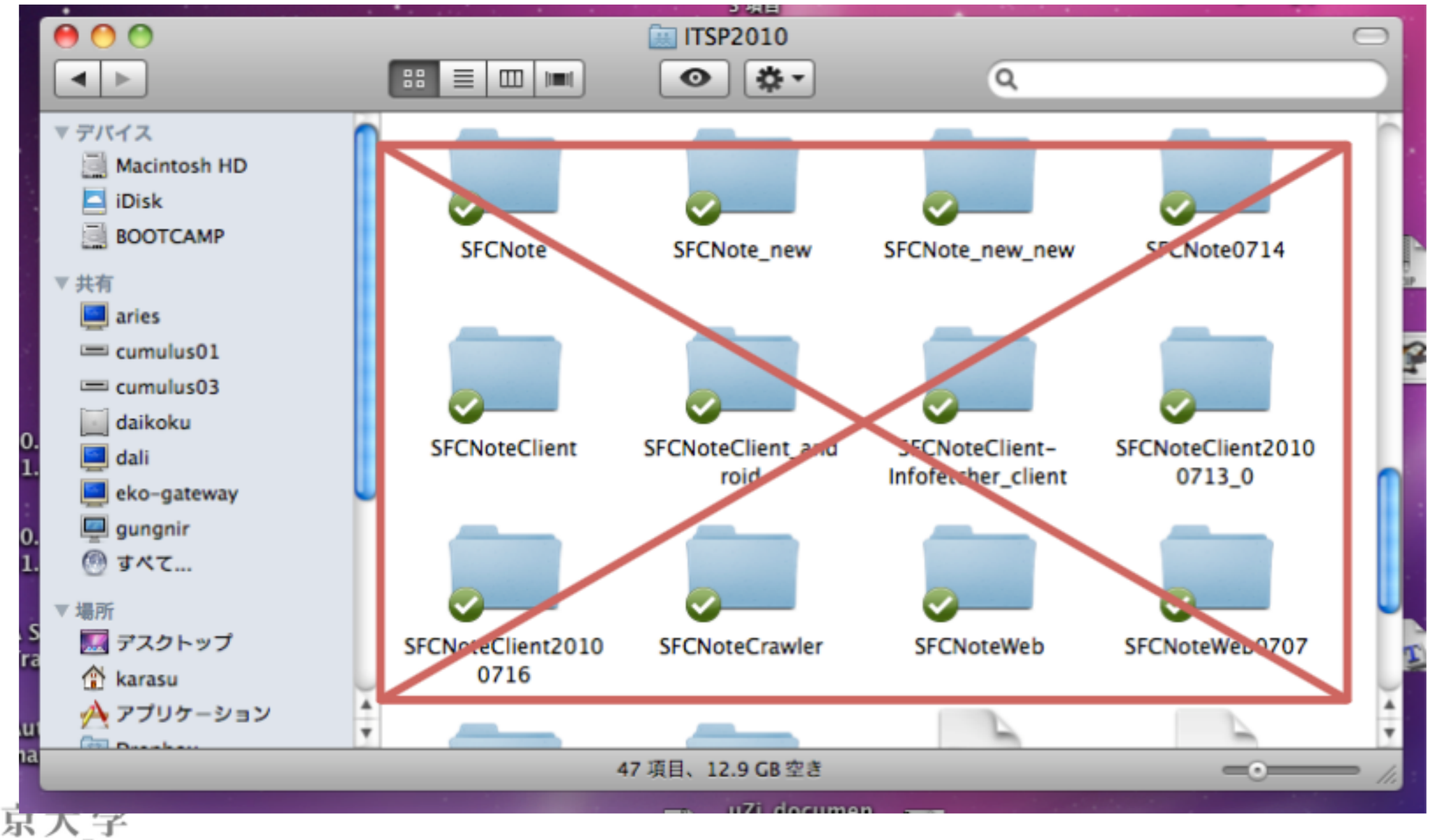
- どこを戻せばいいのかわからなくてつらい

「デモよろしく」

- 作業中で今動かなくてつらい

バージョン管理めんどい

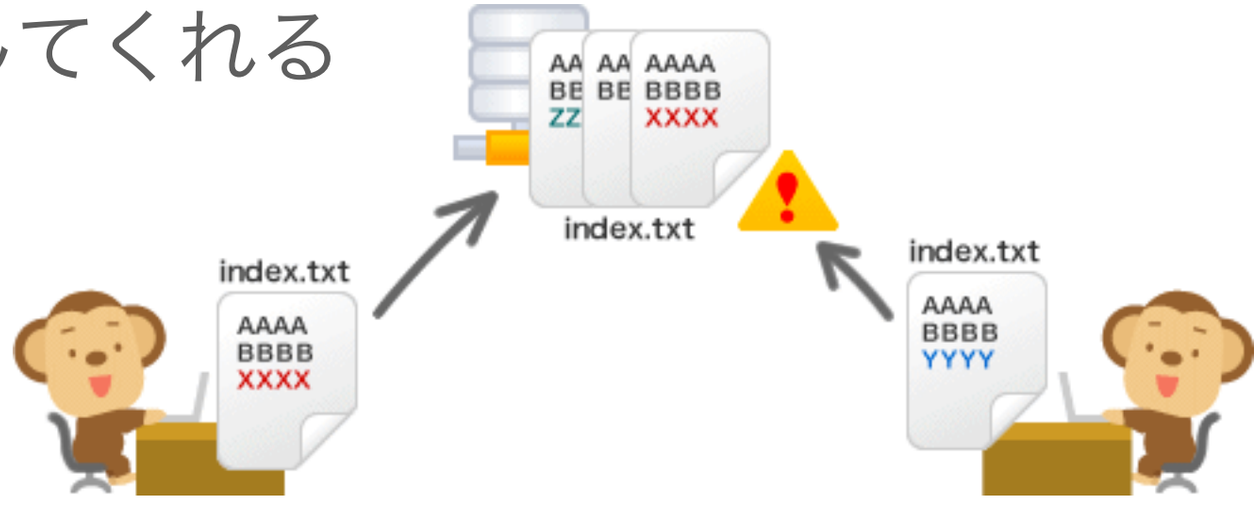
Dropboxでよくね？



結局Gitってなんなん？

ファイルの変更点をタイムスタンプや
ログコメントと共にDBに保存するシステム

- ローカルマシン・遠隔サーバ上のリポジトリ(ファイルDB)にファイルを集約できる
- 複数の人間が編集したファイルを自動的に結合（マージ）してくれる



リポジトリとかマージって何

リポジトリ(Repository)

- ファイル管理を行うプロジェクトディレクトリ
- ローカルに個人用リポジトリがあったり,サーバに共有リポジトリがある

ブランチ(Branch)

- リポジトリの枝
- 安定版・開発版・レガシー版などリポジトリを分岐させる

コミット(Commit)

- ソースコードの変更箇所をリポジトリに保存してバージョニングすること

リポジトリとかマージって何

マージ(Merge)

- 変更ファイルの結合

プッシュ(Push)

- 共有リポジトリに自分のコミットをマージすること

プル(Pull)

- リポジトリからファイルなどをローカルにコピーすること

バージョン管理システムには２種類ある

中央集約型管理システム

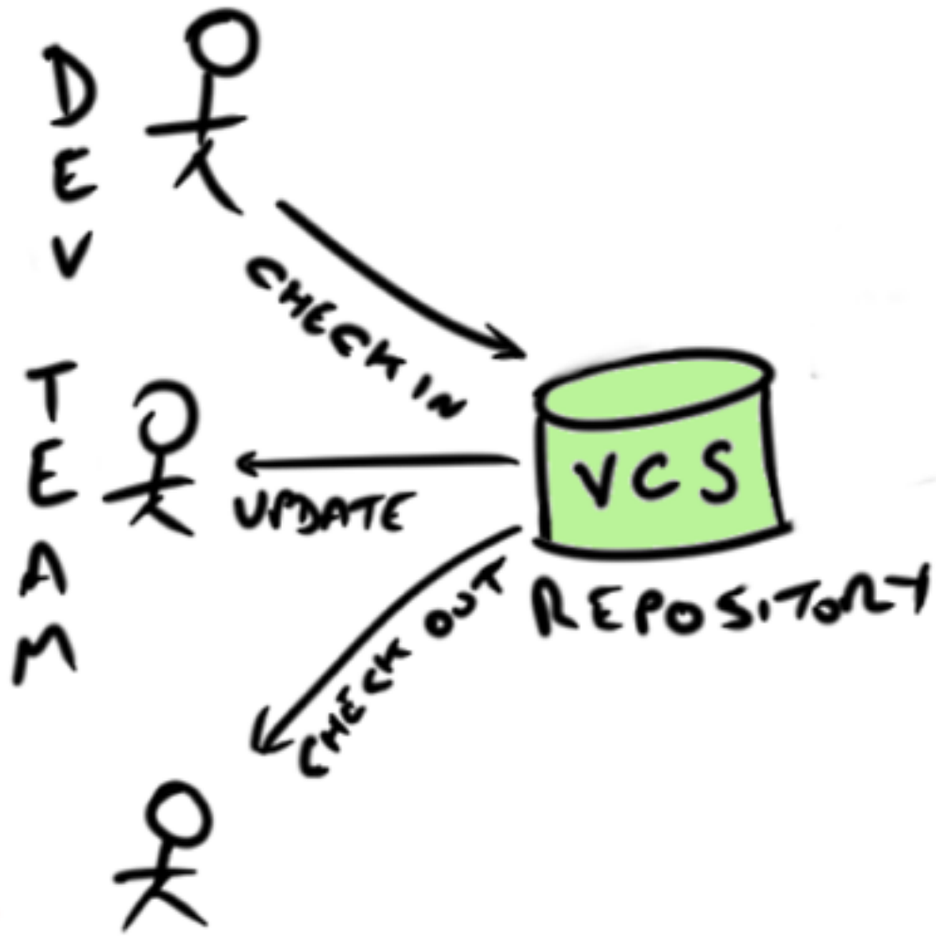
- リポジトリがプロジェクトに１つだけ
- 開発者はリポジトリからローカルにコピーして作業

分散型管理システム（←Gitはこっち）

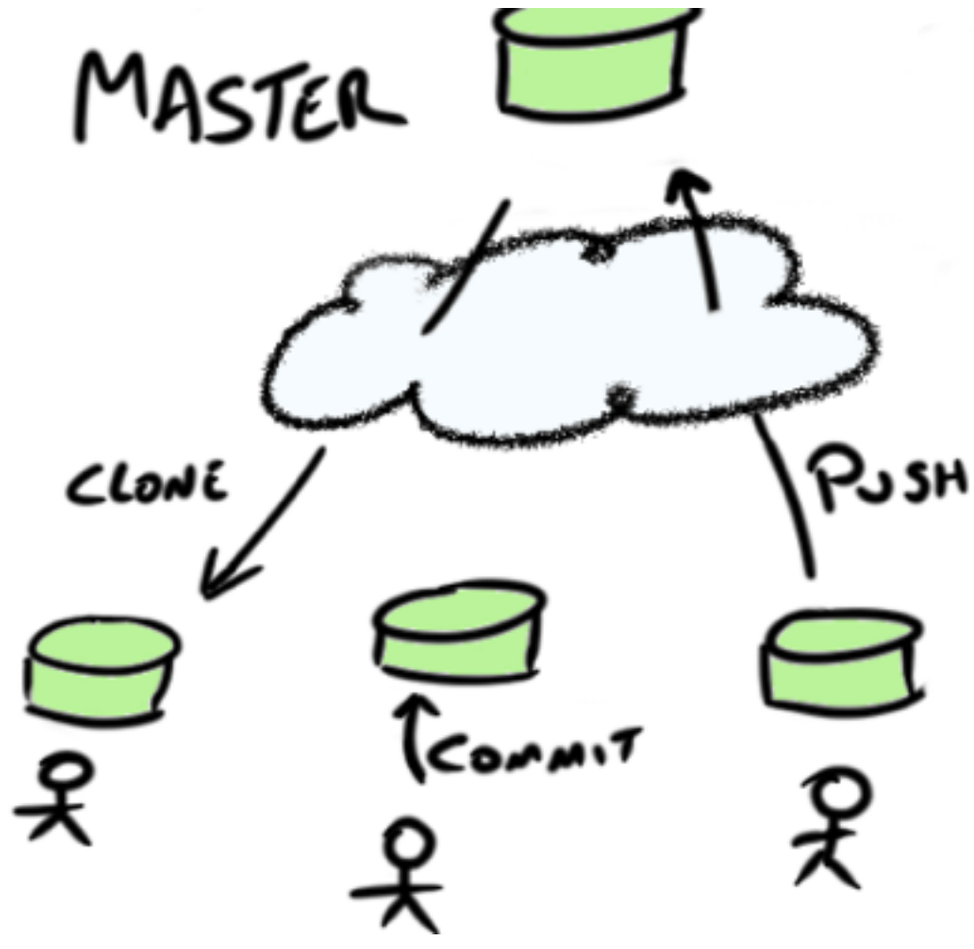
- 開発者がそれぞれでローカル環境にリポジトリをもつ
- 共有リポジトリとローカルリポジトリを同期する

バージョン管理システムには２種類ある

中央集約型管理システム



分散型管理システム(Git)



改めてGitって

Linus / Torvalds氏が開発した分散型バージョン管理

- 彼が開発していたOSのバージョン管理に使うためGit誕生

大規模プロジェクトでも安心の速度で動作

- Google Codeやら某IT企業でも導入されてます

Gitのいいところ

Githubっていう神サービスがある

- https://github.com/
- 無料！

ネットワーク環境がなくてもコミット可

- スタバでドヤる時に
- 中央集約型だとネットワーク環境ないとダメ('ω'乂)

Branch, Reset, Mergeが高速

- ふーん

ここから手動かすよ

インストール

macだし，homebrew入ってるよね？

```
ruby -e "$(curl -fsSL https://raw.github.com/Homebrew/homebrew/go/install)"
```

sudo brew install git で一発！

準備（ローカル編）

ユーザ情報の設定

- ユーザ毎にログイン名や連絡先を登録
- Githubに合わせると楽かもしれない

```
% git config --global user.name “<your_name>”
% git config --global user.email your@mail.address
```

ローカルリポジトリを作成

```
% cd <your_directory>/
% mkdir python_analysis_code
% cd python_analysis_code
% git init
```

さっそく

まずファイルつくろ？

```
% touch README
```

状態statusを見てみよう

```
% git status
```

- Untracked files:にREADMEがあるはず

➡ お初なファイルだからgitも驚いてる

さっそく

ファイルをaddする

```
% git add README
```

statusを見る

```
% git status
```

- Changes to be committed:に入ったはず
- ステージに乗ったとか言うはず
  (今後gitが追跡してくれる)

さっそく

commitする

```
% git commit -m “first commit”
```

- コミットログは短く，わかりやすいものを

statusを見る

```
% git status
```

- nothing to commit, working directory clean とか出てたらOK

ここからちょっと難しい

ブランチ覚えてる？

- 作業を分岐させる

```
% git branch issue1
% git checkout issue1
```

or

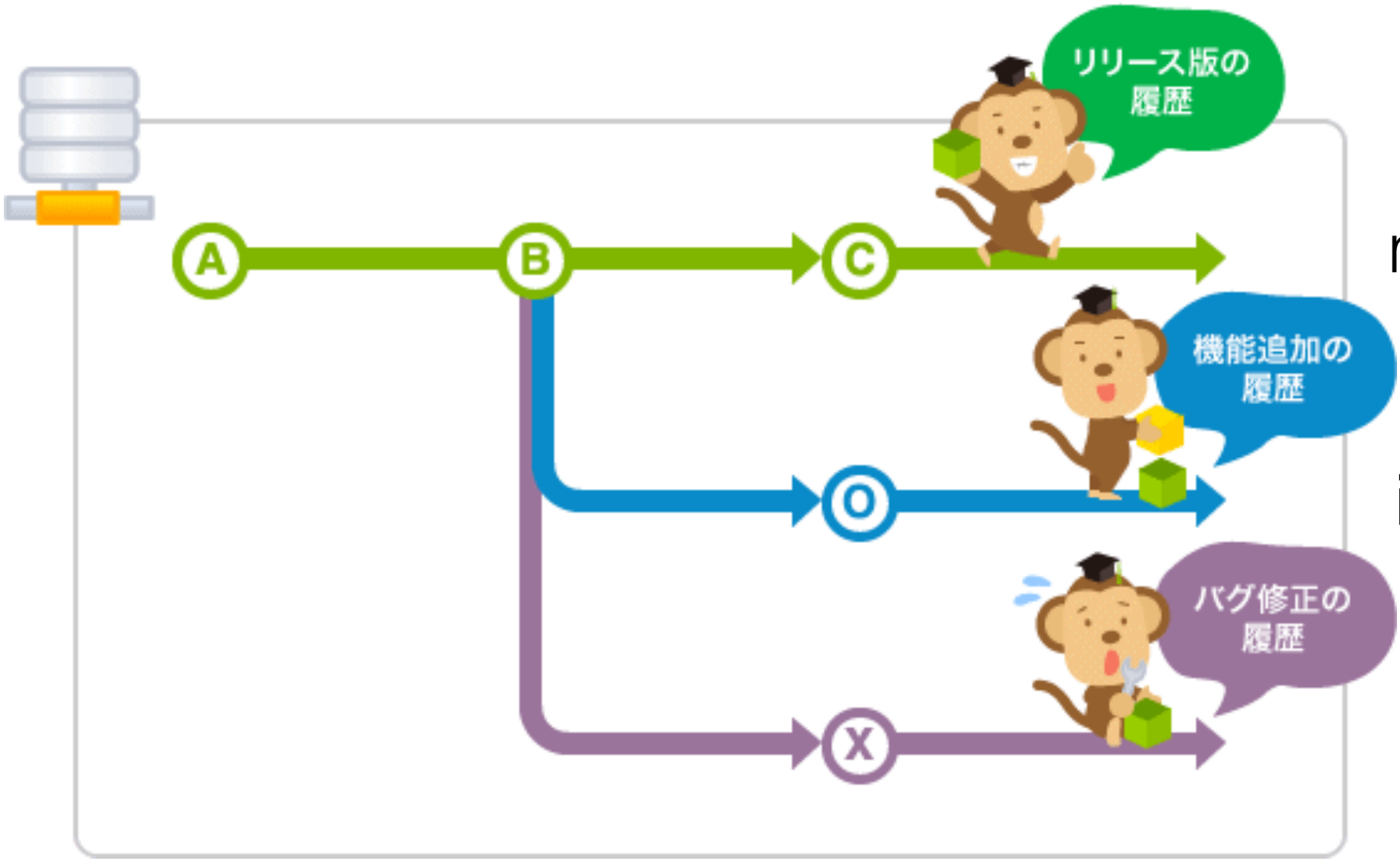
```
% git checkout -b issue1
```

ブランチを見てみよう

```
% git branch
```

- できてたでしょ？ *は今自分がいるブランチ

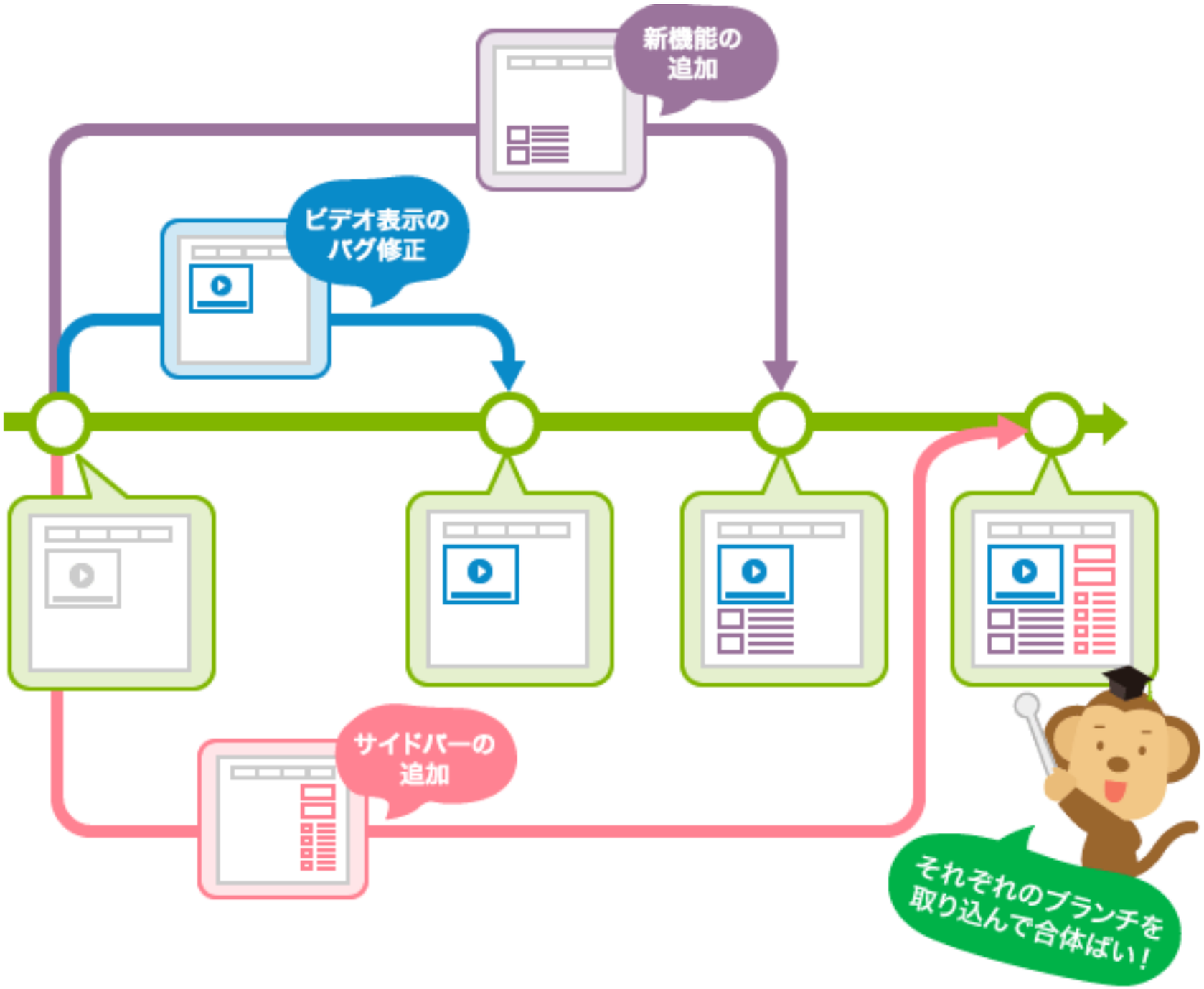
結局ブランチ何



masterブランチ

issue1ブランチ

結局ブランチ何



ブランチ体験

まずブランチを戻そう

```
% git checkout master
```

確認

```
% git branch
```

- ちゃんとmasterに＊ついてる？

ブランチ体験

さっきのREADMEに書こう

```
% vim README
さいこうだよ！かっこいいよ！
```

さっそくadd / commitする

```
% git add README
% git commit -m “READMEに１行追加”
```

そういえば今までどんなことしたっけ？

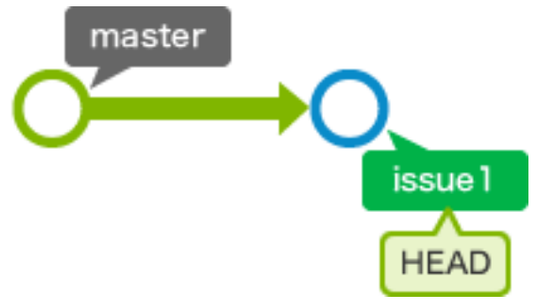
```
% git log
```

ブランチ体験

さっきのブランチに切り替える

- どうやるか覚えてる？

ブランチ体験



さっきのブランチに切り替える

```
% git checkout issue1
```

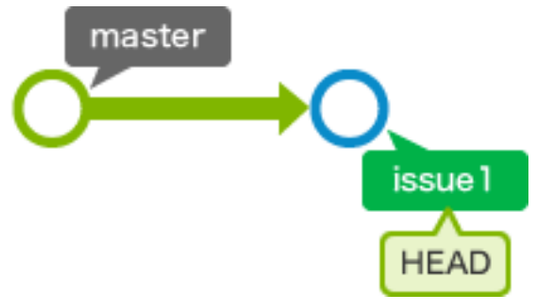
ファイルを更に編集しよう

```
% vim README
さいこうだよ！かっこいいよ！

でもわたしなんかだめだめだよ….
```

終わったらadd / commitしてね

- もうだいじょうぶだよね？

ブランチ体験



さっきのブランチに切り替える

```
% git checkout issue1
```

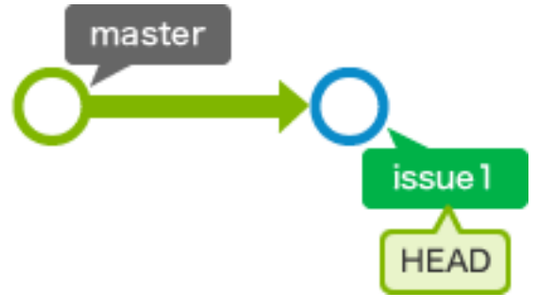
ファイルを更に編集しよう

```
% vim README
さいこうだよ！かっこいいよ！

でもわたしなんかだめだめだよ….
```

終わったらadd / commitしてね

```
% git add README
% git commit -m “返事した”
```

ブランチ体験



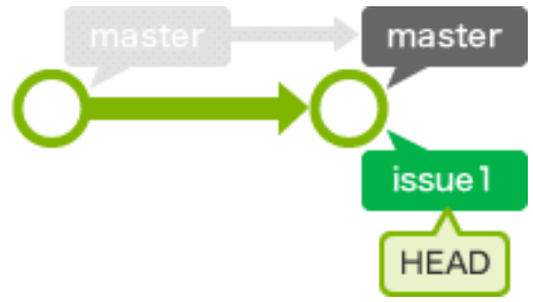
issue1ブランチの変更点をmasterブランチにマージしよう！

...のまえに確認しよう

```
% git checkout master
% less README
```

- 一行しか書いてないよね？

ブランチ体験



ついにマージする時がきた

```
% git merge issue1
```

確認だッ！！

```
% less README
```

- どうなったよ？

ブランチ体験

もうissue1ブランチとはお別れの時(削除)

```
% git branch -d issue1
```

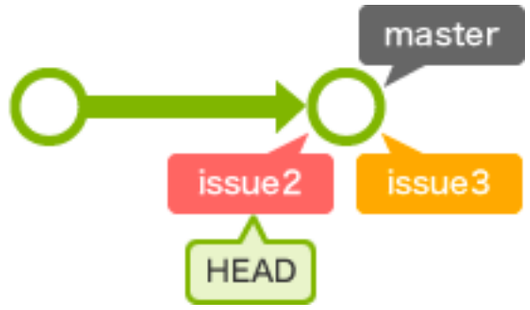
本当にもういないの…？

```
% git branch
```

ブランチ実戦

よくある問題isコンフリクト

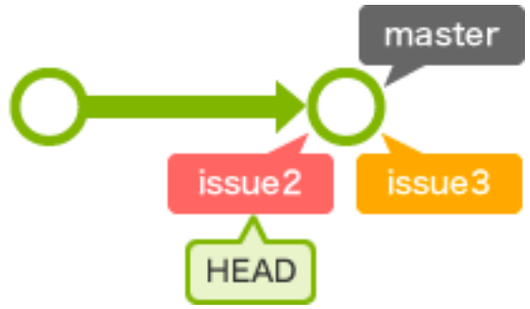
ブランチ実戦



まずブランチ２つ(issue2, issue3)つくろう

- もういけるよね？

ブランチ実戦



まずブランチ２つ(issue2, issue3)つくろう

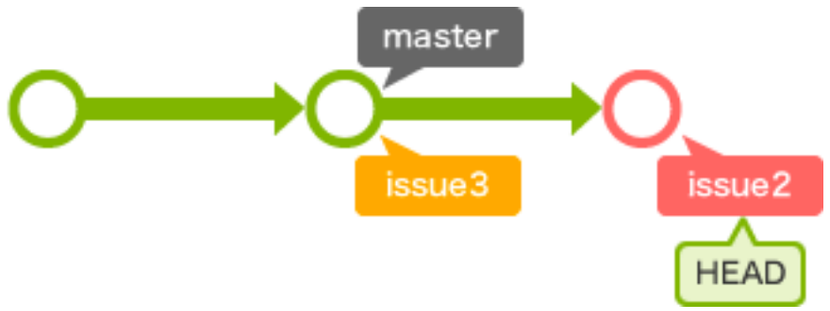
```
% git branch issue2
% git branch issue3
% git checkout issue2
```

一応確認

```
% git branch
```

- issue2に＊ついてるよね

ブランチ実戦



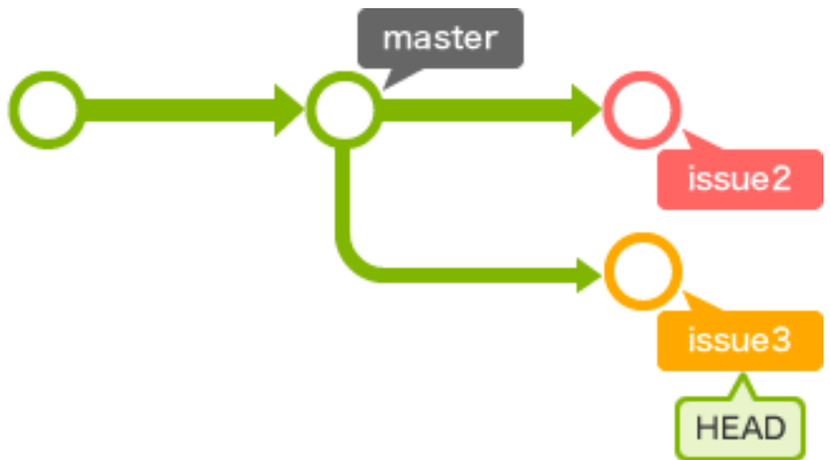
さらにファイルを編集

```
% vim README
さいこうだよ！かっこいいよ！

でもわたしなんかだめだめだよ….

そんなことないよ！僕よりイケてるよ！
```

add / commit しましょ

```
% git add README
% git commit -m “励まされた”
```

ブランチ実戦



issue3ブランチにチェックアウト

```
% git checkout issue3
```

確認してみる

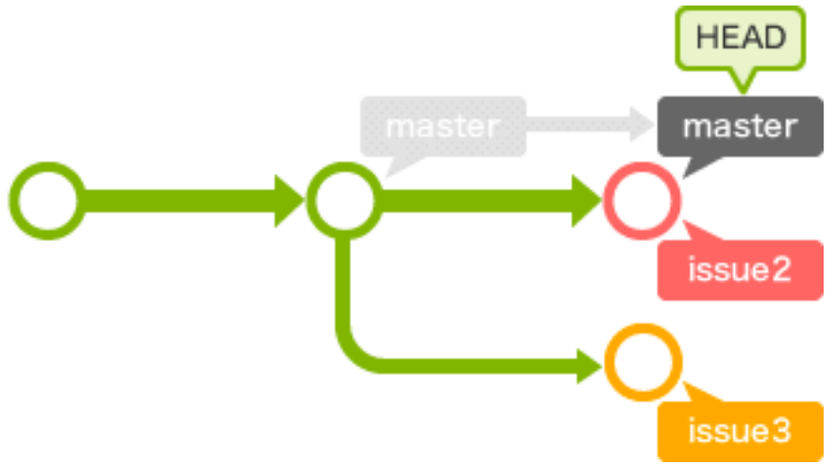
```
% less README
```

ファイルを編集しちゃおう

```
% vim README
さいこうだよ！かっこいいよ！

でもわたしなんかだめだめだよ….

そうかも…ダメかもしれない….
```

```
% git add README
% git commit -m “追い込まれた”
```

ブランチ実戦



issue2ブランチをmasterにマージしちゃおう

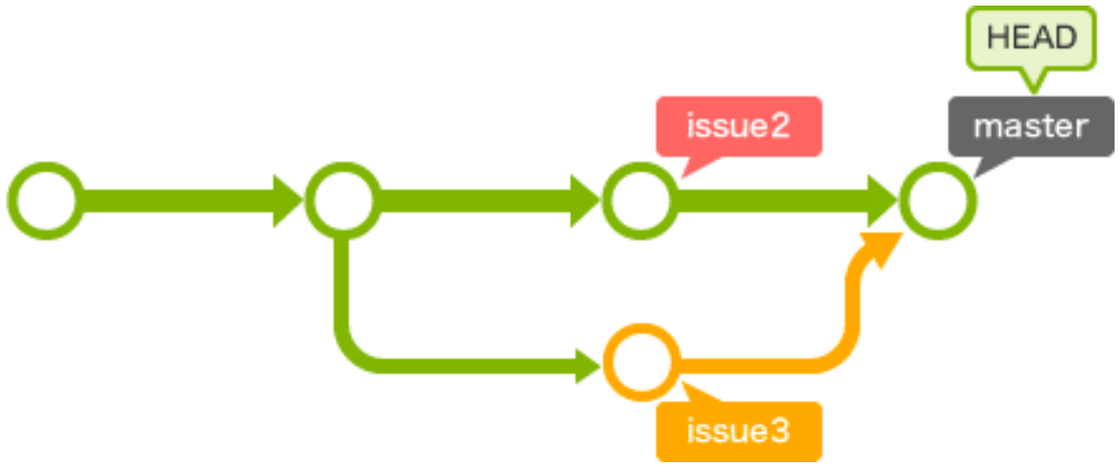
```
% git checkout master
% git merge issue2
```

確認してみる

```
% git log
% less README
```

- masterのREADMEにも励まされたのが入ってるよね

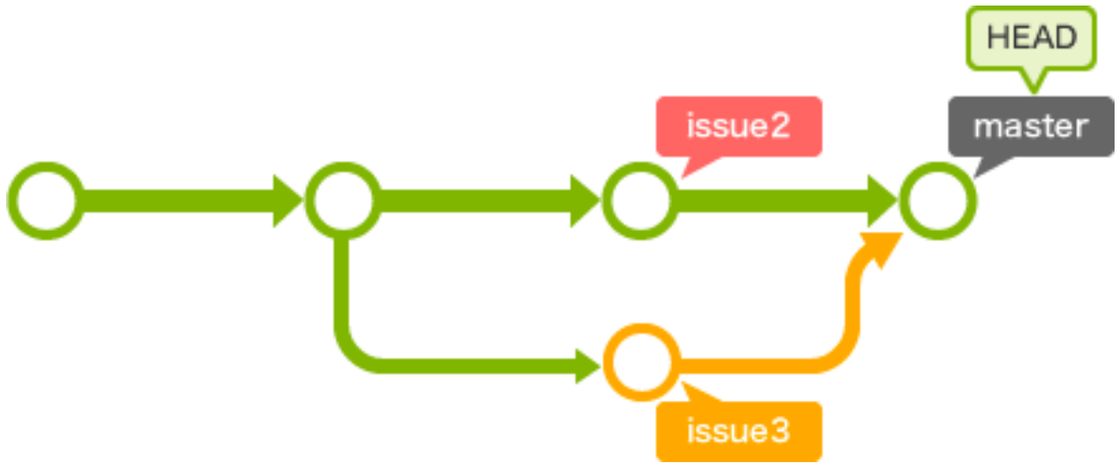
ブランチ実戦



issue3ブランチをmasterにマージしちゃおう

...あれ？３行目のとこどうすんの？

ブランチ実戦



とりあえずやってみよう

```
% git merge issue3
```

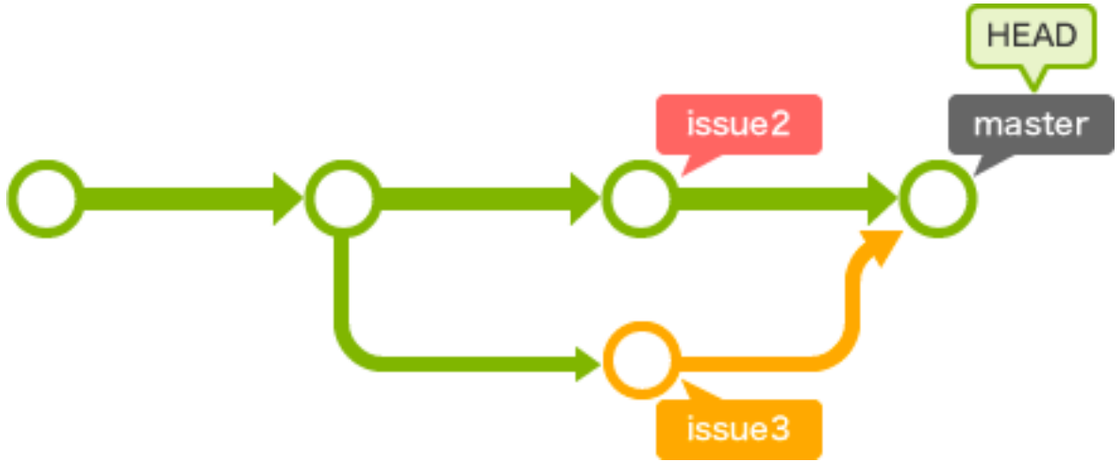
ほらやっぱりコンフリクト(競合)おきた

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

ファイルはどうなったん？

```
% vim README
```

ブランチ実戦



こんな感じだよね

さいこうだよ！かっこいいよ！

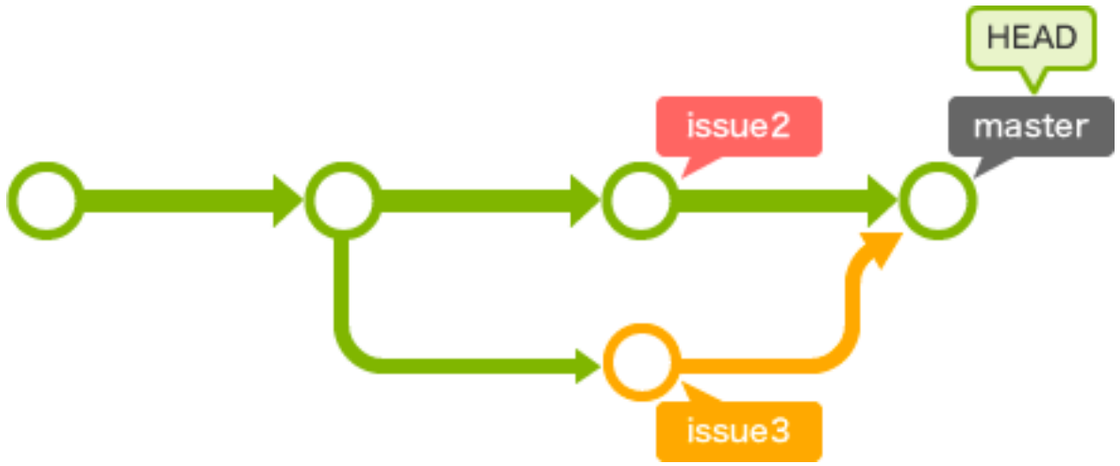
でもわたしなんかだめだめだよ….

<<<<<<< HEAD
そんなことないよ！僕よりイケてるよ！ ←もともとあったやつ

=======
そうかも…ダメかもしれない…. ←マージしようとした奴にあったやつ

>>>>>>> issue3

ブランチ実戦



ここは直すしかない

さいこうだよ！かっこいいよ！

でもわたしなんかだめだめだよ….

そんなことないよ！僕よりイケてるよ！

そうかも…ダメかもしれない….

コンフリクトも修正したし，コミット

```
% git add README
% git commit -m “issue3ブランチをマージ”
```

Gitの動きはだいたいわかったよね？

【参考】サルでもわかるGit入門 http://www.backlog.jp/git-guide/

【Special Thanks】karasuさん できる!プロジェクト管理

そういえばGithubって？

何回か出てきたサーバの共有レポジトリを無料で作れちゃうすごいサービス

- https://github.com

詳しいことはまた今度.

ちなみに，GithubはPublicにしかできないので,

もし誰にも見られたくない，っていうのであれば,

Bitbucketがオススメです.

まとめ

Gitを使えばかわいいは作れる！

【参考】サルでもわかるGit入門 http://www.backlog.jp/git-guide/

【Special Thanks】karasuさん できる!プロジェクト管理

東京大学 THE UNIVERSITY OF TOKYO

つかれたから休憩

このあとは統計だよ

Python for Data Analysis

What Is This Book About?

Pythonでデータを操作・処理・整理・理解するための基礎を教えてくれる

- 実践的な科学計算
- 手法の解説はしないよ

Why Python for Data Analysis?

近年アプリ・学問において科学計算で使われるようになってきた

- データ分析・計算・可視化で優れてる
- CythonとかでCやC++とのインターフェースが用意され，コンパイルも速くなってきてる
- RやMATLABと違って科学計算だけでなくProductにも使える
- JavaやC++には速度負けるけど，プログラミングの時間は短縮できるし，トレードオフでしょ(いけるならCとかのほうがいい)

Essential Python Libraries

NumPy

- 配列とか高速に使える

pandas

- データ構造とかをよしなに扱える

matplotlib

- グラフとか可視化最強

IPython

- リッチなインタラクティブコンソール

SciPy

- いろいろな科学計算が入ってる

ここからオリジナル

おすすめコンソール

iterm2

- http://iterm2.com/
- Preference>Profiles>Command
  - /bin/zsh

~/.zshrcを作成，編集すると幸せになれる

- 痒いところに手が届くようになるよ
- 他人のを参考にするといい
  - https://gist.github.com/mollifier/4979906/
  - http://shepherdmaster.hateblo.jp/entry/20110924/1316881131

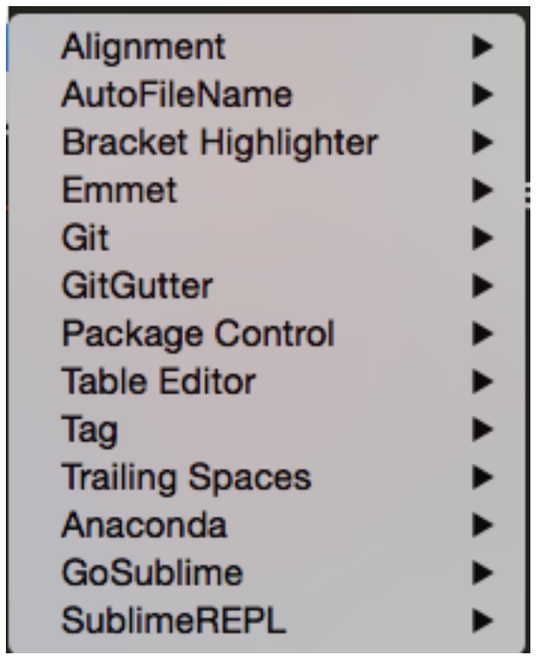
おすすめエディタ

SublimeText3

- http://www.sublimetext.com/3/

設定とかプラグイン

- http://qiita.com/mattuso/items/259f60b3d3bc46bf983c



pyenv

pythonには

- 2系3系と言われるバージョンがある
- いろいろなディストリビューションがある

使い分けちゃおう！っていうのがpyenv

pyenv インストール諸々

インストール

```
% sudo brew install pyenv
```

設定(~/.zshrc追記)

```
if which pyenv > /dev/null; then eval "$(pyenv init -)"; fi
export PYENV_ROOT=/usr/local/opt/pyenv
```

使い方

```
% pyenv install 2.7.8
% pyenv install anaconda-2.1.0
```

```
% pyenv global anaconda-2.1.0
% pyenv local anaconda-2.1.0
```

ライブラリのインストール

pip使えば世界は救われる

```
% pip install numpy scipy pandas ipython
```

matplotlibはちょっとだけ特殊

```
% brew install freetype
% brew install libpng # 確かかなり重要

% pip install matplotlib
```

※sudoは適宜つけてくだしあ