

コラボレイティブ開発特論

Table of Contents

Part 1: ガイダンスとモダンな道具達	1
第1章 ガイダンス	1
第2章 コラボレイティブ開発の道具達	5
Part 2: Git/GitHub演習（別資料）	6
第3章 Git/GitHub演習	7
Part 3: Sinatra/Heroku	7
第4章 SinatraでWebアプリを作ろう	7
Part 3: Ruby on Rails/Heroku	11
第5章 Ruby on Railsアプリの開発	11
第6章 DBを使うアプリの開発と継続的統合	17
補足資料	22
補足資料	22

Part 1: ガイダンスとモダンな道具達

第1章 ガイダンス

連絡事項

連絡事項(1)

1. 資料等の入手先

- GitHubの下記リポジトリにまとめておきます
 - https://github.com/ychubachi/collaborative_development
- 資料は随時updateするので、適宜、最新版をダウンロードしてください

連絡事項(2)

1. 仮想環境（Vagrant）

- 各自のPCに仮想環境をインストールしておいてください
- インストールするソフトウェア
- Git
 - <https://gitforwindows.org/>
- VirtualBox
 - <https://www.virtualbox.org/>
- Vagrant

- <https://www.vagrantup.com/>

授業の全体像

学習の目的

- ビジネスアプリケーションを構築するための基礎力
- 分散型PBLを実施する上で必要となる知識やツールの使い方
- これら活用するための自己組織的なチームワーク

学習の目標

- 分散ソフトウェア開発のための道具を学ぶ
 - 開発環境（Ruby）, VCSとリモートリポジトリ（GitHub）
 - テスト自動化, 継続的インテグレーション, PaaS
- これらのツールの 設計思想 に対する本質理解

前提知識と到達目標

1. 前提とする知識
 - 情報系の学部レベルで基礎的な知識を持っていること
2. 最低到達目標
 - 授業で取り上げる各種ツールの基本的な使い方を身につける
3. 上位到達目標
 - 授業で取り上げる各種ツールの高度な使い方に習熟する.

授業の形態

1. 対面授業
 - 担当教員による講義・演習
2. 個人演習
 - 個人によるソフトウェア開発
3. グループ演習
 - グループによるソフトウェア開発

授業の方法

講義・演習・課題

1. 講義
 - ツールの説明
 - ツールの使い方
2. 演習
 - 個人でツールを使えるようになる
 - グループでツールを使えるようになる

成績評価

1. 課題

- 。個人でソフトウェアを作る
- 。グループでソフトウェアを作る

2. 評価の方法

- 。課題提出と実技試験

3. 評価の観点

- 。分散PBLで役に立つ知識が習得できたかどうか

==== 自己紹介

===== 自己紹介

===== 名前

- 。中鉢 欣秀（ちゅうばち よしひで）

===== 出身地

- 。宮城県仙台市

===== 肩書

- 。産業技術大学院大学 産業技術研究科 \ 情報アーキテクチャ専攻 准教授

===== 連絡先

E-Mail

yc@aiit...

Facebook

ychubachi

Twitter

ychubachi（あんまり使ってない）

Skype

ychubachi（あんまり使ってない）

===== 学歴

1991年	4月	慶應義塾大学環境情報学部 入学
1995年	10月	同大大学院 政策・メディア研究科
		修士課程 入学
1997年	10月	同大大学院 政策・メディア研究科
		後期博士課程 入学

2004年	10月	同大大学院 政策・メディア研究科
		後期博士課程 卒業
		学位：博士（政策・メディア）

===== 職歴

1997年	10月	合資会社ニューメリック設立
		社長就任
2005年	4月	独立行政法人科学技術振興機構
		PD級研究員
		（長岡技術科学大学）
2006年	4月	産業技術大学院大学 産業技術研究科
		情報アーキテクチャ専攻 准教授

===== 起業経験

===== 社名

- 。合資会社ニューメリック

===== 設立

- 。1997年

===== 資本金

- 。18万円

===== 起業の背景

===== 設立当時の状況

- 。Windows 95が普及（初期状態でインターネットは使えなかった）
- 。後輩のやっていたベンチャーの仕事を手伝って面白かった

===== 会社設立の理由

- 。「やってみたかった」から
- 。少しプログラムがかければ仕事はいくらでもあった
- 。後輩にそそのかされた・笑

===== 起業から学んだこと

- 。実プロジェクトの経験

- 。使える技術
- 。お金は簡単には儲からない

===== 教育における関心事

===== 情報技術産業の変化

- 。情報技術のマーケットが変化
- 。ユーザ・ベンダ型モデルの終焉

===== モダンなソフトウェア開発者

- 。新しいサービスの企画から，ソフトウェアの実装まで何でもこなせる開発者
- 。このような人材の育成方法

「学びの共同体」になろう

共に学び、共に教える「場」

- ・ 教室に集うメンバーで 学びの共同体 になろう
- ・ 困った時には助けを求める
- ・ 他人に教えること = 学び

チーム演習での問題解決（理想の流れ）

1. 困った時はメンバーに聞く
 2. わからなかったらチーム全員で考える
 3. それでもダメなら他のチームに相談
 4. 講師・コーチに尋ねるのは最終手段！
 5. ...となるのが理想
- 。授業の進め方などの質問は遠慮無く聞いてください

共同体になるためにお互いを知ろう

- ・ 皆さんの自己紹介
 - 。名前（可能であれば所属も）
 - 。どんな仕事をしているか（あるいは大学で学んだこと）
 - 。この授業を履修した動機

第2章 コラボレイティブ開発の道具達

モダンなソフトウェア開発とは

ソフトウェア開発のための方法・言語・道具

授業で取り上げる範囲

/figures/FLT_framework.pdf

1. 取り上げること
 - 。良い道具には設計思想そのものに方法論が組み込まれている

- 世界中の技術者の知恵が結晶した成果としてのOSSのツール

2. 取り扱わないこと

- 方法論そのものについてはアジャイル開発特論で学ぶ
- プログラミングの初歩については教えない

Scrumするためのモダンな道具たち

モダンな開発環境の全体像

/figures/tools.pdf

1. 仮想化技術（Virtualization）

- WindowsやMacでLinux上でのWebアプリケーション開発を学ぶことができる
- HerokuやTravis CI等のクラウドでの実行や検査環境として用いられている

2. ソーシャルコーディング（Social Coding）

- LinuxのソースコードのVCSとして用いられているGitを学ぶ
- GitはGitHubと連携することでOSS型のチーム開発ができる

enPiT仮想化環境

1. 仮想環境にインストール済みの道具

- エディタ（Emacs/Vim）
- Rubyの実行環境
- GitHub, Heroku, Travis CIと連携するための各種コマンド（github-connect.sh, hub, heroku, travis）
- PostgreSQLのクライアント・サーバーとDB
- 各種設定ファイル（.bash_profile, .gemrc, .gitconfig）
- その他

2. 仮想化環境の構築用リポジトリ（参考）

- [ychubachi/vagrant_enpit](https://github.com/ychubachi/vagrant_enpit)

enPiT仮想化環境にログイン

1. 作業内容

- 前の操作に引き続き、仮想化環境にSSH接続する

2. コマンド

```
vagrant up
vagrant ssh
```

Part 2: Git/GitHub演習（別資料）

第3章 Git/GitHub演習

Git/GitHub演習の解説と演習

Git/GitHub演習について

1. Git/GitHub演習

GitとGitHubにとことん精通しよう

2. 演習資料

[ychubachi/github_practice: Git/GitHub演習](#)

Part 3: Sinatra/Heroku

第4章 SinatraでWebアプリを作ろう

クラウド環境のアカウント・設定

GitHub/Herokuのアカウントを作成

1. GitHub

- [Join GitHub · GitHub](#)

2. Heroku

- [Heroku - Sign up](#)

仮想環境の準備から起動

Port Forwardの設定(1)

1. 説明

- Guest OSで実行するサーバに, Host OSからWebブラウザでアクセスできるようにしておく
- 任意のエディタでVagrantfileの「config.vm.network」を変更
- 任意のエディタでVagrantfileを変更

Port Forwardの設定(2)

1. 変更前

```
# config.vm.network "forwarded_port", guest: 80, host: 8080
```

2. 変更後

```
config.vm.network "forwarded_port", guest: 3000, host: 3000
config.vm.network "forwarded_port", guest: 4567, host: 4567
```

Sinatraアプリケーションの作成

Sinatraを使った簡単なWebアプリケーション

1. Sinatraとは？
 - Webアプリケーションを作成するDSL
 - Railsに比べ簡単に、学習曲線が緩やか
 - 素早くWebアプリを作ってHerokuで公開してみよう
2. 参考文献
 - [Sinatra](#)
 - [Sinatra: README](#)

Sinatraアプリ用リポジトリを作成する

- Sinatraアプリを作成するため、GitHubで新しいリポジトリを作る
 - 名前は「sinatra_enpit」とする
 - できたらcloneする

Sinatraアプリを作成する(1)

- エディタを起動し、次のスライドにある「hello.rb」のコードを入力

1. コマンド

```
emacs hello.rb
git add hello.rb
git commit -m 'Create hello.rb'
```

Sinatraアプリを作成する(2)

- Sinatraアプリ本体のコード（たった4行！）

1. コード: **hello.rb**

```
require 'sinatra'

get '/' do
  "Hello World!"
end
```


Sinatraアプリを起動する

- hello.rbをrubyで動かせば、サーバが立ち上がる
 - vagrantのport forwardを利用するため、「-o」オプションを指定する
1. コマンド

```
ruby hello.rb -o 0.0.0.0
```

Sinatraアプリの動作確認

- **Host OS** のWebブラウザで、http://localhost:4567 にアクセスする
 - 「Hello World!」が表示できれば成功
- アクセスできない場合は **Vagrantfile** のPort Forwardの設定を見直すこと

Sinatraのマニュアル

- <http://sinatrarb.com/intro-ja.html>

アプリをGitHubにpushする

- GitHubにコードをpushしよう
1. コマンド

```
git add .  
git commit -m 'My Sinatra App'  
git push
```

HerokuでSinatraを動かす

Sinatraアプリのディプロイ

- SinatraアプリをHerokuで動作させてみよう
- Webアプリは世界中からアクセスできるようになる
- WebアプリをHeroku（などのアプリケーションサーバ）に 設置することを配備（Deploy）と言う

SinatraアプリをHerokuで動かせるようにする

- SinatraアプリをHerokuで動作させるには、（少ないものの）追加の設定が必要
 - 次スライドを見ながら、エディタを用いて、

次の2つのファイルを作成する

ファイル名	内容
config.ru	Webアプリサーバ（Rack）の設定

ファイル名	内容
Gemfile	他のメンバーやHeroku/Travis CIとで、
	Gemのバージョンを揃える

追加するコード

1. コード: **config.ru**

```
require './hello'
run Sinatra::Application
```

2. コード: **Gemfile**

```
source 'https://rubygems.org'
gem 'sinatra'
```

関連する**Gem**のインストール

- **Gemfile** の中身に基づき、必要なGem（ライブラリ）をダウンロードする
 - **Gemfile.lock** というファイルができる
 - このファイルもcommitの対象に含める

1. コマンド

```
bundle install
```

アプリを**GitHub**に**push**する

- Herokuで動かす前に、commitが必要

1. コマンド

```
git add .
git commit -m 'Add configuration files for Heroku'
git push
```

Herokuにアプリを作る

1. Herokuでの操作

- Heroku にログインする
- 新しいアプリを作る
- GitHubと連携させる
- 手動でデプロイする

- 以降、GitHubにpushするとHerokuにも自動でデプロイされる

演習課題

演習課題4-1

1. Sinatraアプリの作成

- Sinatraアプリを作成して、Herokuで動作させなさい
- SinatraのDSLについて調べ、機能を追加しなさい
- コミットのログは詳細に記述し、どんな作業を行ったかが他の人にも分かるようにしなさい
- 完成したコードはGitHubにもpushしなさい

演習課題4-2 (1)

1. Sinatraアプリの共同開発

- グループメンバーでSinatraアプリを開発しなさい
- 代表者がGitHubのリポジトリを作成し他のメンバーを Collaborators に追加する
 - 他のメンバーは代表者のリポジトリをcloneする
- どんな機能をもたせるかをチームで相談しなさい
 - メンバーのスキルに合わせて、できるだけ簡単なもの（DBは使わない）

演習課題4-2 (2)

1. Sinatraアプリの共同開発（続き）

- 慣れてきたらGitHub Flowをチームで回すことを目指す
 - ブランチを作成し、Pull Requestを送る
 - 他のメンバー（一人以上）からレビューを受けたら各自でマージ
- GitHubのURLとHerokuのURLを提出
 - <http://goo.gl/forms/p1SXNT2grM>

Part 3: Ruby on Rails/Heroku

第5章 Ruby on Railsアプリの開発

Ruby on Railsアプリの生成と実行

RoRを使ったWebアプリケーション

1. Ruby on Rails（RoR）とは？

- Webアプリケーションを作成するためのフレームワーク

2. 参考文献

- [Ruby on Rails](#)

rails_enpit アプリを作成する

- rails は予め、仮想化環境にインストールしてある
- rails new コマンドを用いて、RoRアプリの雛形を作成する
 - コマンドは次スライド

rails_enpit を作成するコマンド

```
rails new ~/rails_enpit --database=postgresql
cd ~/rails_enpit
git init
git create
git add .
git commit -m 'Generate a new rails app'
git push -u origin master
```

GemfileにJS用Gemの設定

- GemfileにRails内部で動作するJavaScriptの実行環境を設定する
 - 当該箇所のコメントを外す
1. 変更前

```
# gem 'therubyracer', platforms: :ruby
```

2. 変更後

```
gem 'therubyracer', platforms: :ruby
```

Bundle installの実行

- Gemfileを読み込み、必要なgemをインストールする
 - rails new をした際にも、 bundle install は実行されている
 - therubyracer と、それが依存しているgemで まだインストールしていないものをインストール
1. コマンド

```
git commit -a -m 'Run bundle install'
```

Gemfile設定変更のコミット

- ここまでの内容をコミットしておこう
1. コマンド

```
git add .
git commit -m 'Edit Gemfile to enable the rubyracer gem'
git push -u origin master
```

データベースの作成

- rails_enpitアプリの動作に必要なDBを作成する
- DatabasesはHerokuで標準のPostgreSQLを使用する
 - RoR標準のsqliteは使わない
- enPiT仮想環境にはPostgreSQLインストール済み

PostgreSQLにDBを作成

1. 開発で利用するDB

rails_enpit_development	開発作業中に利用
rails_enpit_test	テスト用に利用
(rails_enpit_production)	(本番環境用)

- 本番環境用DBは **Heroku**でのみ 用いる

2. コマンド

```
createdb rails_enpit_development
createdb rails_enpit_test
```

PostgreSQLクライアントのコマンド

1. クライアントの起動

- **psql** コマンドでクライアントが起動

2. psqlクライアントで利用できるコマンド

Backslashコマンド	説明
l	DBの一覧
c	DBに接続
d	リレーションの一覧
q	終了

Rails serverの起動

- 次のコマンドでアプリケーションを起動できる
1. コマンド

```
bundle exec rails server -b 0.0.0.0
```

Webアプリの動作確認

- Host OSのWebブラウザで、 `http://localhost:3000` にアクセスして確認
- 端末にもログが表示される
- 確認したら、端末でCtrl-Cを押してサーバを停止する

Controller/Viewの作成

Hello Worldを表示するController

- HTTPのリクエストを処理し、Viewに引き渡す
 - MVC構造でいうControllerである
- `rails generate controller` コマンドで作成する
 1. コマンド

```
bundle exec rails generate controller welcome
```

生成されたControllerの確認とコミット

- `git status`コマンドでどのようなコードができたか確認

```
git status
```

- Controllerのコードを作成した作業をコミット

```
git add .  
git commit -m 'Generate the welcome controller'
```

Hello Worldを表示するView

- HTML等で結果をレンダリングして表示する
 - erbで作成するのが一般的で、内部でRubyコードを動作させることができる
- `app/views/welcome/index.html.erb` を（手動で）作成する
 - コードは次スライド

Hello Worldを表示するViewのコード

1. `index.html.erb`

```
<h2>Hello World</h2>
<p>
  The time is now: <%= Time.now %>
</p>
```

作成した**View**の確認とコミット

- `git status`コマンドで変更内容を確認

```
git status
```

- Viewのコードを作成した作業をコミット

```
git add .
git commit -m 'Add the welcome view'
```

routeの設定

- Routeとは？
- HTTPのリクエスト（URL）とコントローラを紐付ける設定
 - ここでは **root** へのリクエスト（**GET /**）を

welcome コントローラの **index** メソッドに紐付ける

- `config/routes.rb` の当該箇所をアンコメント

```
root 'welcome#index'
```

- `bundle exec rake routes` コマンドで確認できる

routes.rbの設定変更の確認

- `routes.rb` は既にトラッキングされているので、`git diff`コマンドで変更内容を確認できる

```
git diff
```

- `routes.rb`を変更した作業をコミット

```
git add .
git commit -m 'Edit routes.rb for the root controller'
```

ControllerとViewの動作確認

- 再度, **rails server** でアプリを起動し, 動作を確認しよう
- Webブラウザで **http://localhost:3000/** を開く
 1. コマンド

```
bundle exec rails server -b 0.0.0.0
```

GitHubへのPush

- ここまでの作業で, controllerとviewを1つ備えるRoRアプリができた
 - 作業が一区切りしたので, GitHubへのpushもしておく
 - 一連の作業を **git log** コマンドで確認してみると良い
1. コマンド

```
git push
```

Herokuにデプロイする

RoRをHerokuで動かす

- 作成しとRoRアプリをHerokuで動作させよう
 1. Getting Started
- [Getting Started with Rails 4.x on Heroku](#)

Heroku用設定をGemfileに追加

- **Gemfile** に **rails_12factor** を追加する
- Rubyのバージョンも指定しておく
- **Gemfile** を変更したら必ず **bundle install** すること
 1. **Gemfile** に追加する内容

```
gem 'rails_12factor', group: :production  
ruby '2.2.5'
```

productionを除いたbundle install

- **rails_12factor** は開発時には利用しない
 - **Gemfile** では「**group: production**」を指定してある
- 次のコマンドでproduction以外のGemをインストール


```
bundle install --without production
```

- このオプションは記憶されるので、2回目以降 `--without production` は不要

デプロイ前にGitにコミット

- Herokuにコードを送るには、gitを用いる
 - 従って、最新版をcommitしておく必要がある
- commitし、まずはGitHubにpushしておく

1. コマンド

```
git commit -a -m 'Set up for Heroku'
git push
```

- 2行目: pushする先はorigin (=GitHub) である

Herokuアプリの作成とデプロイ

- `heroku` コマンドを利用してアプリを作成する

1. コマンド

```
heroku create
git push heroku master
```

- 1行目: `heroku create` で表示されたURLを開く
- 2行目: `git push` はherokuのmasterを指定。デプロイすると、Herokuからのログが流れてくる

演習課題

演習課題6

1. RoRアプリの作成

- ここまでの説明に従い、Herokuで動作するRoRアプリ（`rails_enpit`）を完成させなさい

第6章 DBを使うアプリの開発と継続的統合

DBとScaffoldの作成

Scaffold

1. Scaffoldとは

- [https://www.google.co.jp/search?q=scaffold&client=ubuntu&hs=PiK&channel=fs&hl=ja&source=lnms&tbn=isch&sa=X&ei=smUdVKaZKY7s8AXew4LwDw&ved=0CAgQ_AUoAQ&biw=1195&bih=925\[scaffold](https://www.google.co.jp/search?q=scaffold&client=ubuntu&hs=PiK&channel=fs&hl=ja&source=lnms&tbn=isch&sa=X&ei=smUdVKaZKY7s8AXew4LwDw&ved=0CAgQ_AUoAQ&biw=1195&bih=925[scaffold)

- Google 検索]

2. RoRでは、MVCの雛形のこと

- CRUD処理が全て自動で実装される

Scaffoldの生成方法

1. コマンド

```
git checkout -b books
bundle exec rails generate scaffold book title:string author:string
```

- 多くのコードが自動生成されるので、branchを切っておくと良い
 - 動作が確認できたらbranchをマージ
 - うまく行かなかったらbranchごと削除すれば良い

routeの確認

- Scaffoldの生成で追加されたルーティングの設定を確認

1. コマンド

```
bundle exec rake routes
```

- `git diff` でも確認してみよう

DBのMigrate

1. migrateとは

- Databaseのスキーマ定義の更新
- Scaffoldを追加したり、属性を追加したりした際に行う

2. コマンド

```
bundle exec rake db:migrate
```

参考：Migrateの取り消しの方法

- DBのmigrationを取り消したいときは次のコマンドで取り消せる

```
bundle exec rake db:rollback
```

- 再度、migrateすれば再実行される

```
bundle exec rake db:migrate
```

参考：Scaffold作成の取り消しの方法

1. コマンド

```
git add .  
git commit -m 'Cancel'  
git checkout master  
git branch -D books
```

- 1～2行目：自動生成されたScaffoldのコードをbranchに一旦コミット
- 3行目：masterブランチに移動
- 4行目：branchを削除（**-D** オプション使用）

動作確認

1. 動作確認の方法

- Webブラウザで <http://localhost:3000/books> を開く
- CRUD処理が完成していることを確かめる

2. コマンド

```
bundle exec rails server
```

完成したコードをマージ

1. ブランチをマージ

- 動作確認できたので、**books** branchをマージする
- 不要になったブランチは、**git branch -d** で削除する

2. コマンド

```
git add .  
git commit -m 'Generate books scaffold'  
git checkout master  
git merge books  
git branch -d books
```

Herokuにデプロイ

1. デプロイ

- ここまでのアプリをデプロイする
- herokuにあるdbもmigrateする
- Webブラウザで動作確認する

2. 設定ファイル(Procfile)

```
release: bundle exec rake db:migrate
web: bundle exec rails server -p $PORT
```

RoRアプリのテスト

テストについて

1. ガイド
 - [A Guide to Testing Rails Applications — Ruby on Rails Guides](#)

テストの実行

1. テストコード
 - Scaffoldはテストコードも作成してくれる
 - テスト用のDB (`rails_enpit_test`) が更新される
2. コマンド

```
bundle exec rake test
```

Travis CIとの連携

Travis CIのアカウント作成

1. アカウントの作り方
 - 次のページにアクセスし、画面右上の「Sign in with GitHub」のボタンを押す
 - [Travis CI - Free Hosted Continuous Integration Platform for the Open Source Community](#)
 - GitHubの認証ページが出るので、画面下部にある緑のボタンを押す
 - Travis CIから確認のメールが来るので、確認する

Travisの設定

1. 設定ファイルの変更
 - まず、Rubyのバージョンを指定する
 - 変更の際はYAMLのインデントに注意する
2. `.travis.yml` を書き換える

```
language: ruby
rvm:
- 2.2.5
```

Travis用DB設定ファイルと作成

- RubyのVersionなど
- テストDB用の設定ファイルを追加する

1. `.travis.yml`

```
language: ruby
rvm:
- 2.2.5
services: postgresql
bundler_args: "--without development --deployment -j4"
cache: bundler
before_script:
- cp config/database.travis.yml config/database.yml
- bundle exec rake db:create
- bundle exec rake db:migrate
script: bundle exec rake test
```

2. `config/database.travis.yml`

```
test:
  adapter: postgresql
  database: travis_ci_test
  username: postgres
```

GitHubとTravis CI連携

1. 説明

- ここまでの設定で、GitHubにpushされたコードはTravis CIでテストされるようになった。
- GitHubにプッシュしてWebブラウザでTravis CIを開いて確認する

2. コマンド

```
git add .
git commit -m 'Configure Travis CI'
git push
```

CI通過後のHerokuへの自動deploy

1. HerokuへのDeploy

- テストが通れば、自動でHerokuに配備されるように、Herokuに設定を追加する

演習課題

演習課題7-1

1. rails_enpit の拡張

- Viewを変更
 - welcomeコントローラのviewから、booksコントローラのviewへのリンクを追加する etc
- Scaffoldの追加
 - 任意のScaffoldを追加してみなさい
 - DBのmigrationを行い、動作確認しなさい
- Herokuへの配備
 - Travis経由でHerokuへdeployできるようにする

補足資料

補足資料

Vagrant関連

Vagrantの補足

1. 仮想環境とのファイル共有
 - Guest OS内に **/vagrant** という共有フォルダがある
 - このフォルダはHost OSからアクセスできる
 - 場所はVagrantfileがあるフォルダ
 - 共有したいファイル（画像など）をここに置く

Git関連

Gitの補足

1. 元いたbranchに素早く戻る方法

```
git checkout other_branch # masterで
# 編集作業とcommit
git checkout - # masterに戻る
```

2. Git blame

- だれがどの作業をしたかわかる（誰がバグを仕込んだのかも）
 - [Using git blame to trace changes in a file · GitHub Help](#)

バイナリのコンフリクト(1)

- git mergeでバイナリファイルがコンフリクトした場合、ファイルはgit merge実行前のままとなります^[1]。

- 以下を実行してコンフリクトが発生したとします。

```
git checkout master
git merge branch_aaa
```

バイナリのコンフリクト(2)

- そのままにしたいとき(=masterを採用) は

```
git checkout --ours <binaryfile> #明示的な実行は不要
git add <binaryfile>
git commit
```

- branch_aaaのファイルを採用したいときは

```
git checkout --theirs <binaryfile>
git add <binaryfile>
git commit
```

Hubコマンドについて

- enPiT環境にはHubコマンドが仕込んである
 - [github/hub](#)
- 通常のGitの機能に加えて、GitHub用のコマンドが利用できる
 - コマンド名は「git」のまま（エイリアス設定済み）
- 確認方法

```
git version
alias git
```

GitHub関連

GitHubの補足(1)

1. Issue

- 課題管理（ITS: Issue Tracking System）
- コミットのメッセージでcloseできる
 - [Closing issues via commit messages · GitHub Help](#)

2. Wiki

- GitHubのリポジトリにWikiを作る
 - [About GitHub Wikis · GitHub Help](#)

GitHubの補足(2)

1. GitHub Pages

- 特殊なブランチを作成すると、Webページが構築できる
 - [GitHub Pages](#)

Heroku関連

Herokuの補足

1. HerokuのアプリのURL確認

```
heroku apps:info
```

2. `rails generate` などが動かない

```
spring stop
```

Travis CI関連

Travis CIの補足

1. Status Image

- README.mdを編集し、Travisのテスト状況を表示するStatus Imageを追加する
- [Travis CI: Status Images](#)

2. Deploy後、自動で heroku の db:migrate

- 次のURLの「Running-commands」の箇所を参照
 - [Heroku Deployment - Travis CI](#)

Sinatraでテストを実行可能に

- `Gemfile` に `rake` を追加する

```
gem 'rake'
```

- `Rakefile` を作成する


```
task :default => :test

require 'rake/testtask'

Rake::TestTask.new do |t|
  t.pattern = ".*_test.rb"
end
```

TODO travxs setup のトラブル

- 次のようなトラブルが発生することがある
 - [TravisとGitHubのリポジトリの同期・Issue #17](#)
 - [楽天APIサンプルのfork・Issue #18](#)
- Travis CIからHerokuにデプロイするのではなく, HerokuからGitHubを監視させるようにしたほうがよいかも . . .

[1] [git mergeでバイナリファイルがコンフリクトした場合・Issue #6](#)