BioRuby

後藤　直久
ngoto@gen-info.osaka-u.ac.jp
大阪大学微生物病研究所附属遺伝情報実験センター

自己紹介

- 名前：後藤直久（ごとうなおひさ）
- 所属：大阪大学微生物病研究所附属遺伝情報実験センター　ゲノム情報解析分野　助教
  - 専門
    - バイオインフォマティクス
    - ゲノム情報解析
    - 分子進化

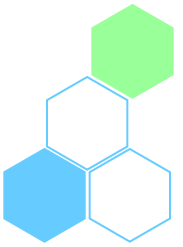

ngoto@gen-info.osaka-u.ac.jp, ng@bioruby.org


http://www.gen-info.osaka-u.ac.jp/~ngoto/


@ngotogenome

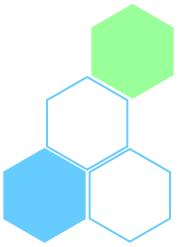

http://jp.linkedin.com/in/ngoto

本日の内容



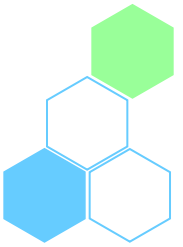
- BioRubyの概要
- 最新のソースに触ってみる
  - バージョン管理ツールGit
  - Git共有リポジトリGitHub
- ~~テストコードの書き方~~

BioRubyとは？



- Ruby言語で書かれたバイオインフォマティクス用ライブラリ+各種ツール
- オープンソース・フリーソフトウェア
- http://bioruby.org/
- http://github.com/bioruby/bioruby

論文が出ました

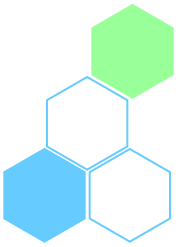


Naohisa Goto, Pjotr Prins, Mitsuteru Nakao, Raoul Bonnal, Jan Aerts, Toshiaki Katayama. (2010) BioRuby: bioinformatics software for the Ruby programming language. Bioinformatics 26(20): 2617-2619.

http://bioinformatics.oxfordjournals.org/content/26/20/2617.abstract

（購読者以外でも全文を読めます）

BioRubyの規模



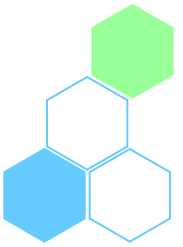
- ファイル数
  - ライブラリ本体: 約 230
  - ユニットテスト: 約 120
  - サンプルスクリプト: 約 70
- 行数(コメント・空行を除く)
  - ライブラリ本体: 約 3万5千行
  - ユニットテスト: 約 2万2千行
- class/moduleの数
  - 約 580 クラス・モジュール
  - 約 2800 メソッド(private除く)

BioRubyの歴史

2000/11/21	BioRubyプロジェクト開始（片山、中尾、奥地)
2001/06/21	バージョン0.1リリース
2001/07/19	Bioinformatics Open Source Conference（デンマーク）ライトニングトーク(奥地)
2001/10/24	バージョン0.3 リリース・CVSレポジトリ開始
2001/11/17	第1回BioRuby宴会（京都）
2001/12/15	バージョン0.3.3 リリース（現存するChangeLogの最初の日付）
2002/02	BioHackathon（南アフリカ）参加（片山）
…	
2005/06-2006/02	IPA未踏ソフトウェアプロジェクト
…	
2005/07/09	第4回関西Ruby勉強会にて発表（後藤・河野）
2006/02/24	バージョン1.0リリース・未踏成果報告会＠品川
…	
2006/12	Phyloinformatics Hackathon（アメリカ）参加（片山、後藤）
2008/2	DBCLS BioHackathon (東京)
2009/3	DBCLS BioHackathon2009 (東京・沖縄)
2009/12/29	バージョン1.4.0 リリース
2010/2	DBCLS BioHackathon2010 (東京)
2010/8	論文アクセプト

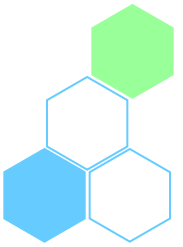
（敬称略）

BioRubyの良いところ

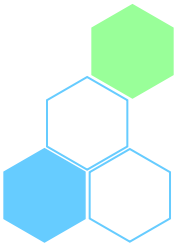


- 継続
  - もうすぐ10周年
- 国際的プロジェクト
  - 海外からも参加
  - 世界のオープンソースバイオ開発者との交流
- 学術的成果：論文
- IT 系コミュニティとの関係
  - Ruby コミュニティ：Ruby勉強会@関西，RubyKaigi
  - IPA 未踏プロジェクト
  - Google Summer of Code

問題点



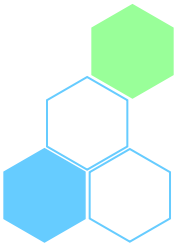
- リリースマネージメント不足
  - 新バージョンのリリース間隔が空いてしまう
- レビューアー不足
  - 新しいコードを試してくれる人があまりいない
  - 他のソフトやデータの準備が必要な場合が多いのも一因
- ドキュメント不足
  - チュートリアルやHowTo が少ない
- 使用例不足
  - ブログなどの記事が少ない
  - サンプルスクリプトも多くない



今後の予定

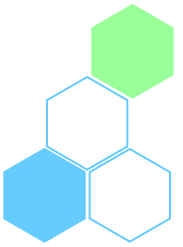
- 新バージョンリリース
  - 今週中には何とか
- プラグイン機能
  - 開発の敷居を下げる
- Ruby 1.9.2 完全対応
- ドキュメントの充実
  - HOWTO, チュートリアルetc.
- 新機能
  - セマンティックウェブ関係
  - 次世代シーケンサー関係

最新のBioRubyに触ってみよう



- BioRubyの開発版の最新のソースはGitHubにある
  - http://github.com/bioruby/bioruby
  - バージョン管理システムGitを使用
- GitHub
  - Git の共有リポジトリを設置できるサーバーのひとつ
    - オープンソースソフトウェアの公開の開発には無料
    - 非公開（指定したユーザー間のみで共有）は有料
- GitHubの「fork」機能を活用
  - GitHub上の他のコードから簡単に「分岐」できる
    - ソースの取得だけならforkしなくてもOKだが
    - BioRubyに取り込んでもらいたい場合はほぼ必須
      - 自分のコードを他の人に見てもらいたい場合も便利

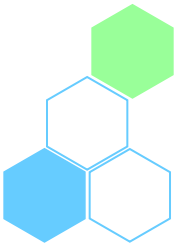
GitHub上でのフォーク(分岐)



- 1. 登録する(とりあえず無料プランでOK)
- 2. http://github.com/bioruby/bioruby にアクセス
- 3.「フォーク (fork) 」ボタンを押す

- これで自分用BioRubyリポジトリの準備完了
  - http://github.com/*****/bioruby
  - (***** は登録したユーザー名)
  - 自分(登録者)と別途指定した共同開発者は書込可能
    - http://github.com/bioruby/bioruby はコアメンバのみ書込可能

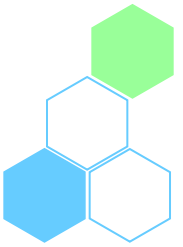
ソースコードのバージョン管理



- バージョン管理システムとは？
  - 追加や変更の履歴を記録してくれるソフト
    - 記録時にその変更に関する説明メモを同時に残すのが通例
      - ファイルとは別にその説明メモも管理してくれるのが普通
  - いつでも昔の状態に戻れる
  - 任意のバージョン間の差分の閲覧もできる

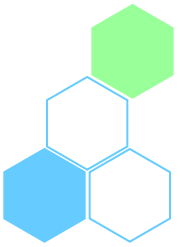
- BioRubyのバージョン管理
  - ～2001年10月(リリースバージョン0.3より前):手動？
  - 2001年10月～2008年8月： CVS
  - 2008年8月～:Git

バージョン管理システムGit



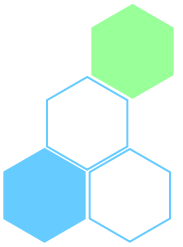
- 「ぎっと」と読む
- 分散型バージョン管理システム
- フリーソフトウェア
- 創始者:リーナス・トーバルズ (Linus Torvalds)
  - Linuxの作者
  - Linuxカーネルのソースコード管理用に開発開始
    - それ以前はBitKeeperという商用ソフトを使っていた
  - 2005年春～7月頃
- 現在のメンテナ: Junio Hamano (濱野 純)
  - 2005年7月頃～現在

インストール



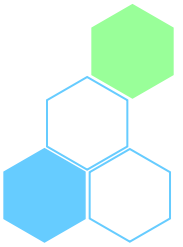
- UNIX / Linux
  - パッケージをインストール、またはソースからビルド
- Mac OS X
  - git-osx-installer
    - http://code.google.com/p/git-osx-installer/
  - または、MacPorts または fink
- Windows
  - msysgit
    - http://code.google.com/p/msysgit/
    - 事前にPuTTYまたはWinSCPをインストールしておくとよい
    - 改行コードの取り扱いに注意
      - インストール時に「Checkout-as-is, commit-as-is」選択が無難

Gitの利点



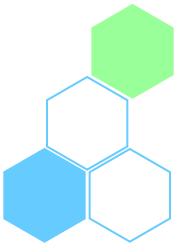
- 速い
  - 巨大なLinuxカーネルソースをストレスなく扱える速さ
- 複数ファイル・ディレクトリ対応
  - CVSはファイル単位の履歴記録だった
- 分散型
  - サーバーや共有ディレクトリ無しでも使用可
    - コミット(変更の記録)にネットワーク接続不要
  - もし壊れても他のリポジトリからの復旧が容易
- 管理用ディレクトリは1レポジトリ1か所のみ (.git)
  - CVS やSubversionは各ディレクトリに再帰的にばらまく
- GitHubなど優れた共有リポジトリの存在

Gitの欠点とその対策



- コマンドがたくさんあってわかりにくい
  - よく使う10個くらいのコマンドだけ覚えておけばOK
- ネット上の情報が新旧混ざっている
  - 古いバージョンの情報が出てくると混乱しがち
    - 急速に発展したソフトなので仕方ない
  - 少なくとも2007年以前の情報は無条件に捨ててください

BioRubyソースの取得



- git clone (または git-clone) コマンド
- (フォークしていない人)

  git clone git://github.com/bioruby/bioruby.git

  (httpしか通らない環境の場合)

  git clone http://github.com/bioruby/bioruby.git

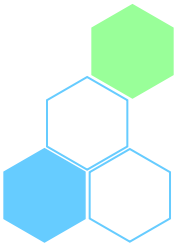
- (フォークした人)

  git clone https://****@github.com/****/bioruby.git

  (**** は登録ユーザー名)

または

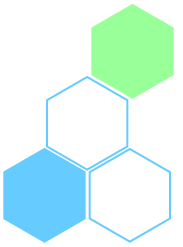
  git clone git://github.com/****/bioruby.git



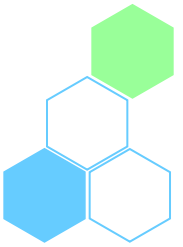
これまでの履歴を見る

- git log コマンド
- 変更されたファイルの情報付きで見たい
  - git log –stat
- パッチ付きで履歴を見たい
  - git log –p
  - 合わせ技もOK: git log –p ––stat

Gitのコミットの指定方法



- 各コミットは16進数40桁のコミットIDを持っている
  - 長いので省略可能
    - (偶然の重複がなければ)先頭5-6ケタ入力すればOK
  - 基本はコミットIDで指定
- 最新のコミットは常に「HEAD」



ソースを変更する

- お好みのエディタを使ってソースを変更

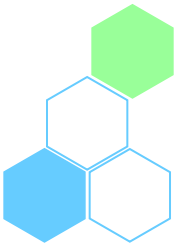
差分を見る

- git diff コマンド

新ファイルの追加・既存ファイル変更の一時記録

- git add コマンド
  - 例: git add sample/my_example.rb
- 新ファイルの場合
  - そのファイルを追加
    - まだ最終確定ではない
- 既存ファイルの場合
  - そのファイルへの変更を一時記録
    - まだ最終確定ではない

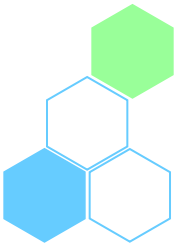
- git add 済みの変更点を見るには
  - git diff --cached



現在の状況を確認

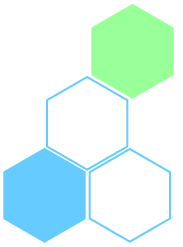
- git status コマンド

変更の記録



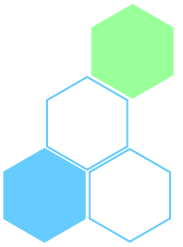
- git commit コマンド
- (git add 済の場合)
  - git commit
    - git add で指定済の変更を記録
- (既存ファイルで git add していない場合)
  - git commit ファイル名...
- (既存ファイルへの全変更を取り込みたい場合)
  - git commit -a
- エディタが起動してコミットメッセージを書き込む

直前のコミットを変更したい



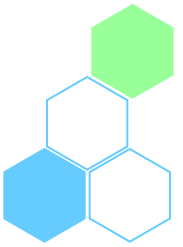
- git commit --amend
  - 典型的にはコミットメッセージのみ変更
  - git commit --amend　前に git add ファイル名 しておくと、そのファイルへの変更も追加で取り込む。
  - 「履歴」自体を改変している

ファイルの削除・名前変更・移動



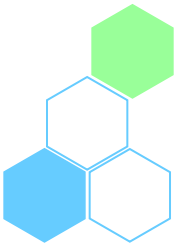
- 削除: git rm コマンド
  - 例: git rm sample/test000.rb
  - まだ最終確定ではない(git add に相当する操作)
  - 確定するには git commit を実行
- ファイル名変更・移動: git mv コマンド
  - 例: git mv sample/test000.rb sample/test001.rb
  - まだ最終確定ではない(git add に相当する操作)
  - 確定するには git commit を実行

変更のキャンセル

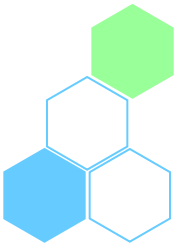


- 間違えたので元の状態に戻したいとき
- git reset コマンド
  - 例: commit をあきらめたのでレポジトリの状態に戻したい
  - git reset --hard HEAD
    - 追加/削除したファイルも元に戻る
    - 実は任意のコミットIDの状態に戻せるが、git reset は**履歴も含めて**完全に「無かった」ことにしてくれるので、生半可に使うと危険

ブランチの作成



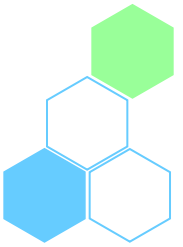
- Git はブランチを簡単に作成できる
  - Git の利点のひとつ
- ブランチの一覧: git branch
  - git branch
    - 最初に作成されるブランチの名前は”master”
- ブランチの作成: git branch コマンド
  - git branch ブランチ名
- ブランチへの移動: git checkout コマンド
  - git checkout ブランチ名
  - ブランチを作成してそこに移動
    - git checkout –b ブランチ名



あるファイルだけ特定の状態にしたい

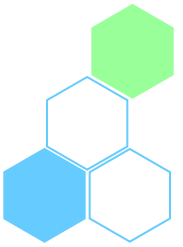
- git checkout コミットID ファイル名...
  - まだ最終確定ではない、つまり後で git commit が必要

ブランチの変更をマージ



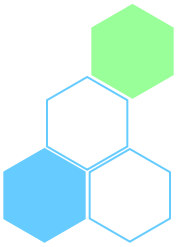
- git merge コマンド
  - git merge ブランチ名
- git の変更の伝播方法
  - “Fast-Forward”
    - 履歴が一直線になる場合
    - 基本的にはこちらだけを使いたい
  - “merge”
    - いわゆるマージ

GitHubに変更を転送



- git push コマンド
  - 例: git push origin master

GitHubの変更を取り込む



- git pull
  - 作業ディレクトリの変更も行う
- git fetch
  - リモートのリポジトリから変更情報を取得するだけ
  - マージは行わない