Git の使い方

森 立平

目標

- バージョン管理システム Git を使えるようになる。

1　バージョン管理システムとは

バージョン管理システムはある程度の規模のプログラム (もっと一般にテキストファイル) を書く上では必須のツールです。今までプログラムを書いてきて、既にある程度動作するプログラムに大きな変更を加えたくなったことがあると思います。そのような場合に一旦別ファイルにコピーしてからプログラムを編集した経験があるかもしれません。バージョン管理システムを使えばそのようなことをする必要はありません。過去のバージョンをいつでも見ることができます。またチームでプログラムの開発をする場合には一つのファイルを複数の開発者が書き換えた場合に可能な限り自動的にマージをしてくれます。この授業では最近のバージョン管理システムの中でも最もよく使用されている Git を使います。以下 Git の最小限の説明をしますが、より詳しく知りたい人は Pro Git `https://progit.org/` を読むとよいでしょう。

2　Git の基本的な使い方

2.1　初期設定

ホームディレクトリに `.gitconfig` というファイルを作って以下のように書いてください。

```
[user]
        name = 名前
        email = メールアドレス
[push]
        default = simple
```

ここで **名前**はあなたの名前の英語表記で置き換えてください。また **メールアドレス**はあなたのメールアドレスで置き換えてください。

2.2　リポジトリを作る (演習では使わないが知っておいた方がよい)

Git はリポジトリという単位でファイル群を管理します。まず Git で管理したいディレクトリに移動します。既に作成したファイルがあっても、空のディレクトリであっても構いません。そして

```
git init
```

とコマンドを実行します。これで現在のディレクトリ以下を Git で管理できるようになりました。この Git で管理されているディレクトリのことをリポジトリと呼びます。このディレクトリには `.git` というディレクトリが作成されています (`ls -A` で確認できます)。このディレクトリは Git が情報を保存するためのディレクトリであり、設定ファイルである `.git/config` 以外を編集することはありません。

2.3 管理するファイルを追加する

リポジトリを作っただけでは Git はどのファイルを管理すればよいのか分かりません。

`git add` **ファイル名**

というコマンドで Git で管理したいファイルを指定します。ここで **ファイル名**は Git で管理したいファイル名に置き換えてください。たとえば `hello.scala` というファイルを Git で管理したいときは

```
git add hello.scala
```

としてください。これで Git は「`hello.scala` というファイルの変更を管理すればよいのだ」ということを理解します。しかしまだ Git は `hello.scala` の内容を読んではいません。次に紹介する `git commit` でファイルの内容を記録する必要があります。また一つのリポジトリ内で複数のファイルを追加しても構いません。

2.4 管理しているファイルの変更を記録する

`add` したファイルを Git で記録したいときは

`git commit -m` **”コメント”**

とします。ここで **コメント**は「このファイルの変更に対するコメント」で置き換えてください。自分一人しか使用しないリポジトリの場合はこのコメントはなんでもよいです。

リポジトリ内で最終コミット以降に更新された全てのファイルの変更を記録したいときは

`git commit -a -m` **”コメント”**

としてください。

コミット直後に Git が記録している最新のファイルの状態をリビジョンと呼び、それぞれのコミットに「リビジョン名」が割り当てられます (後述)。

2.5 その他のコマンド

過去のコミットログを見たい場合は

```
git log
```

としてください。

最終コミットと現在のファイルとの差分を見るには

```
git diff
```

としてください。特定のリビジョンと現在のファイルとの差分を見たい場合は

`git diff` **リビジョン名**

としてください。ここで **リビジョン名**は比較したいリビジョンのリビジョン名で置き換えてください。ここでリビジョン名とは `git log` で該当するコミットに関するログ

```
commit 8ac50947b3f75158f457335c65f3ff93e687c045
Author: Ryuhei Mori <mori@is.titech.ac.jp>
Date:   Sat Sep 19 10:33:44 2015 +0900

    comment
```

の中で `8ac50947b3f75158f457335c65f3ff93e687c045` にあたる部分です。これはとても長いですが最初の数文字を指定すれば十分です。例えばこの例の場合は

```
git diff 8ac509
```

としてください。
　特定のリビジョン同士の差分を見るには

```
git diff リビジョン名1 リビジョン名2
```

としてください。
　コマンドのヘルプを見たいときは

```
git help gitコマンド名
```

としてください。ここで `git コマンド名`は git のコマンド名 (init, add, commit, log, diff など) で置き換えてください。また、ここで紹介したコマンド以外にもたくさんの git コマンドがあります。

3　やってみよう

Git のリポジトリを作って一通りコマンドを使ってください。一例としては、リポジトリを作る → なにかテキストファイルを作る → 追加する → コミットする → ファイルを編集する → 差分を見る → 変更をコミットする → ログを見る → ファイルを編集する → 最初のリビジョンとの差分を見る。

4　Git の clone, push, pull について

Git を使って複数人で共同開発するには `clone`, `push`, `pull` コマンドを使用する必要があります。この演習の授業でもこれらのコマンドを使って課題の配布、提出をおこないます。この章で使う用語の説明をします。

- ワーキングディレクトリ: 現在作業しているディレクトリで Git で管理されているもの。ディレクトリ `.git/` を直下に含むディレクトリ。
- ローカルリポジトリ: ワーキングディレクトリに対応するリポジトリ。実体としては `.git/` の中身。
- リモートリポジトリ: インターネット上あるいはその他ネットワーク上にあるリポジトリ。この授業では GitHub 上のリポジトリを指す。

4.1　リモートリポジトリを clone する

リモートリポジトリを clone してローカルリポジトリを作成するには

```
git clone リモートリポジトリのパス
```

とコマンドを実行します。ここで **リモートリポジトリのパス** の指定方法には HTTPS と SSH の二種類がありますが、この授業では SSH を用いてアクセスします。GitHub 上のリポジトリのパスはウェブブラウザからリポジトリのページにアクセスすることで調べられます (リポジトリのページにある “Clone or download” と書いてある緑色のボタンを押して “Clone with SSH” を選んでください)。例えば `alg2018/Alogirhtms-Datastructures` の場合はリポジトリのパスは

```
git@github.com:alg2018/Alogirhtms-Datastructures.git
```

です。なので

```
git clone git@github.com:alg2018/Alogirhtms-Datastructures.git
```

とすることで GitHub の `alg2018/Alogirhtms-Datastructures` というリポジトリを clone できます。現在のディレクトリの下に `Alogirhtms-Datastructures` というディレクトリが作成されました。これは Git のローカルレポジトリでもあります。

4.2 ローカルリポジトリの変更をリモートリポジトリに反映する

リモートリポジトリに書き込む権限がある場合にはローカルリポジトリの変更をリモートリポジトリに反映することができます。まずワーキングディレクトリの内容をローカルリポジトリに commit してください。

```
git status
```

とコマンドを実行して

```
nothing to commit, working directory clean
```

と最後の行に表示されていれば大丈夫です。このローカルリポジトリに対する commit をリモートリポジトリに反映するには

```
git push
```

とします。もしもリモートリポジトリが更新されている可能性があるのであれば先に次に説明する `git pull` を実行しておいてください。

4.3 リモートリポジトリの変更をローカルリポジトリに反映する

逆にリモートリポジトリの変更をローカルリポジトリに反映したい状況があります。複数人でファイルを更新している場合や一人が大学や自宅など複数の場所でファイルを更新している場合等が該当します。そのような場合には

```
git pull
```

とします。前回の pull の後にリモートとローカルで同じファイルの同じ箇所が変更されていると、変更の衝突を自力で解決する必要があります (今回は説明しません)。

5 この授業の課題の進め方について

この授業では GitHub の Fork という機能と Pull request という機能を使って課題の配布、提出を実現します。この授業で課題を受け取り提出するまでのワークフローは以下のようになります。

1. ウェブブラウザで GitHub 上の課題のリポジトリ (alg2018 の下のリポジトリ。学生は Read-Only) にアクセスし自分のアカウントに Fork する。そのためにウェブブラウザで GitHub 上の課題のリポジトリにアクセスし右上にある Fork と書かれたボタンを押す。
2. 自分のアカウントに Fork されたリポジトリをローカルに clone する。
3. ファイルを編集したり追加したりして課題を終らせる。その間には定期的に commit したりその他の Git のコマンドを使用したりする。
4. GitHub 上の Fork によって作成されたリポジトリに push する。単に `git push` とすればよい。課題を進める途中で push しても構わないが最後には必ず push する。
5. GitHub 上の Fork によって作成されたリポジトリから Fork 元のリポジトリに Pull request を送る。そのためにウェブブラウザで Fork によって作成されたリポジトリにアクセスしファイルリストの上の左側にある緑色のボタンを押す。そうすると Fork 元のリポジトリとの差分が表示される。Pull request のタイトルとメッセージを書いて緑色の “Create pull request” ボタンを押す。タイトルは最後の commit のメッセージが入力されているはずだが好きなものに変更してよい (「課題を提出します」など)。メッセージは簡単な感想や質問、課題の中で工夫した点などを書く。
6. もしも Pull request を送った後にファイルを更新して再提出したい時には自分の一回目の Pull request にコメントでその旨を書く。各課題につき Pull request は一人一回とする。

学生同士でお互いの GitHub 上のリポジトリや Pull request を見ることができます。自分が課題を終えた後に他の人の解法を見てみると、勉強になるかもしれません。以下が課題提出の注意点です。

1. Fork してから clone する。元の `alg2018/`**レポジトリ名**から clone しない。
2. 忘れずにワーキングディレクトリの内容をローカルリポジトリに commit する。
3. その後に忘れずにリモートリポジトリ (Fork で作成されたもの) に push する。
4. その後に忘れずに GitHub 上で Fork で作成されたリポジトリから Fork 元のリポジトリに Pull request を送る。

このうち一つでも忘れると正しく課題が提出できないので注意してください。課題が提出できたか不安なときは教員、TA に聞いてください。

日常での Git の使用

Git はとても便利な道具です。Git に慣れると継続的に編集するようなテキストファイルを全て Git で管理したくなります。今後プログラムやレポート、論文 (Tex ファイル) を書く機会が増えてくるかと思いますが、それらを Git で管理しインターネット上のリポジトリに保存しておくととても便利です。大学でも自宅でも同じファイルにアクセスすることができ編集履歴が全て残ります。そのため日常的に Git を使うことをすすめます。ただし GitHub は無料アカウントでプライベートリポジトリを作成できないので Bitbucket 等他のリポジトリホスティングサービスを使用した方がよいでしょう。