計算機科学第一

10 月 13 日

今週の目標

- バージョン管理システム Git の `clone`, `push`, `pull` 及びその他のコマンドが使えるようになる。
- Scala のビルドツール sbt と ScalaTest を用いた単体テストができるようになる。

今日の課題

1. GitHub 上にある `titech-is-cs115/students` リポジトリの `members/` ディレクトリの下に自分の名前のファイルを作成し自己紹介等を書く。
2. GitHub で配布されている今回の課題 (sbt と ScalaTest を用いた単体テスト) をやる。提出締切は今週金曜日 23 時 59 分 59 秒。

今日のワークフロー

1. SSH 鍵を GitHub アカウントに登録する。
2. メールで送られた Invitation を受け入れて titech-is-cs115 の students グループのメンバーになる (既にメンバーになっている人はスキップ)。
3. GitHub 上にある `titech-is-cs115/students` リポジトリの `members/` ディレクトリの下に自分の名前のファイルを作成し自己紹介等を書く。そのために Git コマンド clone, add, commit, push 等を用いる。
4. GitHub 上にある `titech-is-cs115/assignment0` リポジトリを自分のアカウントに fork する。
5. Fork によって作成された `#自分のアカウント名#/assignment0` リポジトリをローカルに clone する。この時にできたローカルリポジトリでまず sbt を起動しておいてから単体テストの課題に取りかかる (sbt の起動に時間がかかるので)。
6. 単体テストの課題を終わらせて`#自分のアカウント名#/assignment0` リポジトリに push する。
7. Fork によって作成されたリポジトリ `#自分のアカウント名#/assignment0` から Fork 元のリポジトリ `titech-is-cs115/assignment0` に Pull request を送る。

1 SSH 鍵の GitHub への登録

この授業の課題の配布、提出は GitHub を通じて行います。Git から GitHub 上のリポジトリにアクセスする度にパスワードを入力するのは面倒なので SSH 鍵を登録しておきます。ここで SSH とは Secure Shell の略で暗号化された通信を実現するための仕組みです。まず SSH 鍵を生成するために

```
ssh-keygen
```

とコマンドを実行します。すると

```
Generating public/private rsa key pair.
Enter file in which to save the key (/home/mori/.ssh/id_rsa):
```

と鍵ファイルのパスを尋ねられますがデフォルトのままでよいので、そのまま Enter を押します。次に

```
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
```

とパスフレーズ (鍵を使用するときのパスワード) を尋ねられますが、これは空にしたいので何も入力せずに Enter を押します。そうすると

```
Your identification has been saved in /home/mori/.ssh/id_rsa.
Your public key has been saved in /home/mori/.ssh/id_rsa.pub.
The key fingerprint is:
3c:08:1f:10:a4:c1:4e:46:cf:5a:6d:ca:8c:58:ad:2b mori@gmac01.is.titech.ac.jp
The key's randomart image is:
+--[ RSA 2048]----+
| oo.+.           |
|  +* o           |
| +o * +          |
| o.B = +         |
|. + + o S        |
|   .     .       |
|E .              |
| .               |
|                 |
+-----------------+
```

というように表示されて鍵が生成されます。ここで id_rsa が秘密鍵、id_rsa.pub が公開鍵です (両方ともテキストファイルです)。公開鍵は暗号化に用いられ秘密鍵は復号に用いられます。秘密鍵は他人に知られてはいけません。公開鍵は暗号化された通信をするために通信の相手に教える必要があります。公開鍵は第三者に知られても安全です。この授業でこれ以上公開鍵暗号の説明はしませんが、「秘密鍵は信用できない者に知られてはいけない」、「公開鍵は誰に知られたとしても安全」ということは知っておいてください。

次に公開鍵を GitHub に登録します。GitHub にログインして右上にあるアイコンから Settings を選びます。左のメニューから SSH keys を選びます。そこで Add SSH key を選び、Title と Key (公開鍵) を入力します。Title はなんでもよいです (ひょっとしたら空でもよいかも)。公開鍵として id_rsa.pub の内容を入力します。ホームディレクトリで

```
cat .ssh/id_rsa.pub
```

とコマンドを実行すると

```
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQC/gxywsteOQMka+SQRuSboMkamcKTp16
s1Kaac6GsdSIhZeJNfn+j/Ei9HOR7kg94ENIon2FHhgAffMtMIno9HZiiE+32ynxBf5trL
AgvzBnzDTu8PMfz3uxH6Yai5MDVufBlT++A42fwhxQQGcF4rmO77/2LWXSwTijGk7W2ji8
0OdBvZgmhwQNNg5LPa8x9JsPt4E3LgZPEREaVyxmPJzJFohRXLvMyqBtft3F60Qb7hAnrP
mUssRGLqxt8ah39HL/nN2t1KXMx3UH2pLMgxvxv/K6hhItX93vQM88YLtL8E+dB14Tp7lk
DxshNfgaA+qhlWrFgd98OLsvCeEMtb mori@gmac01.is.titech.ac.jp
```

というようにファイルの中身が表示されます。これをコピーアンドペーストで Key のところに入力してください。これで GitHub に SSH を通じてアクセスできるようになりました。もしも自宅など別の環境から GitHub にアクセスしたい場合には別途 SSH 鍵を作成して登録してください (GitHub には複数の公開鍵を登録できます)。

[この段落は余談です] SSH はもともとリモートマシンに暗号化された通信用いて安全にログインするための仕組みです。SSH を使えば西 7 号館の演習室に外からログインすることも可能です。設定は演習室で `~/.ssh/authorized_keys` というファイルを作成して自宅で使用する SSH 鍵の公開鍵を書き込むだけです (`~/`はホームディレクトリという意味)。これで自宅など他の場所から

```
ssh #アカウント名#@porto.is.titech.ac.jp
```

で演習室のマシンにログインできるようになります (Windows の場合は何か SSH クライアントをインストールして使う必要があります)。この場合に使用する秘密鍵は絶対に他人に知られないようにしてください。秘密鍵を知られてしまうと演習室のマシンにログインされてしまいます。

2 Git の clone, push, pull について

Git の基本的な使い方については前回の資料を見てください。より詳しい説明は公式のドキュメント `https://git-scm.com/` にあります。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。

2.1 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。例えば `titech-is-cs115/students` の場合はリポジトリのパスは

```
git@github.com:titech-is-cs115/students.git
```

です。なので

```
git clone git@github.com:titech-is-cs115/students.git
```

とすることで GitHub の `titech-is-cs115/students` 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 `students` 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。

2.2 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。

2.3 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)。

3 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。当初はこの 9 月に公開された GitHub の機能 “Classroom for GitHub” を利用したかったのですが問題があったため利用を見送りました。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 (titech-is-cs115 の下の `assignment-#課題番号#` というリポジトリ。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 (できれば提出したいリビジョン名も書く)。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放を見てみると、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. 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。

2. 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。

3. 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全てのテキストファイルを Git で管理したくなります。今後プログラムや論文、レポート (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。

4 sbt と ScalaTest を用いた単体テスト

今回は 2 種類の簡単な課題をやってもらいます。1 つ目は閏年を判定するプログラムとそのテストプログラムの作成です。説明は脇田先生の資料にあります。400 で割り切れる年についてもテストできるようにテストプログラムを書いてください。2 つ目は最大公約数を求めるプログラムとそのテストプログラムの作成です。最大公約数を求める関数 `GCD.gcd(x,y)` は入力 x と y の絶対値の最大公約数を返すものとします。入力に負の値があっても正しく計算しているかどうかテストできるようにテストプログラムを書いてください。両方の課題で正しく答を計算するプログラムを書いてください (ヒント: x の絶対値は `x.abs` で計算できます)。ディレクトリ `src/` と `test/` にある計 4 つのプログラムファイルを編集してください。テストをするためには sbt を起動して test コマンドを実行すればよいです。その場合 `test/` にある全てのテストが実行されます。特定のテストだけを実行したいときは testOnly コマンドを使用してください。今回の例で最大公約数を求める関数のテストだけを実行したいときは `testOnly cs1.assignment0.GCDTest` と sbt コマンドを実行してください。今日は sbt コマンドは test, ~test, testOnly, ~testOnly しか使う必要がありませんが、sbt について他のコマンドなどの詳しい説明が知りたい場合は公式のドキュメント `http://www.scala-sbt.org/` を読んでください。また、今日の課題のテストプログラムについてはリポジトリ内のテストプログラムの雛形に沿って書けばよいですが、ScalaTest について詳しくしりたい場合は `http://scalatest.org/` を参照してください。