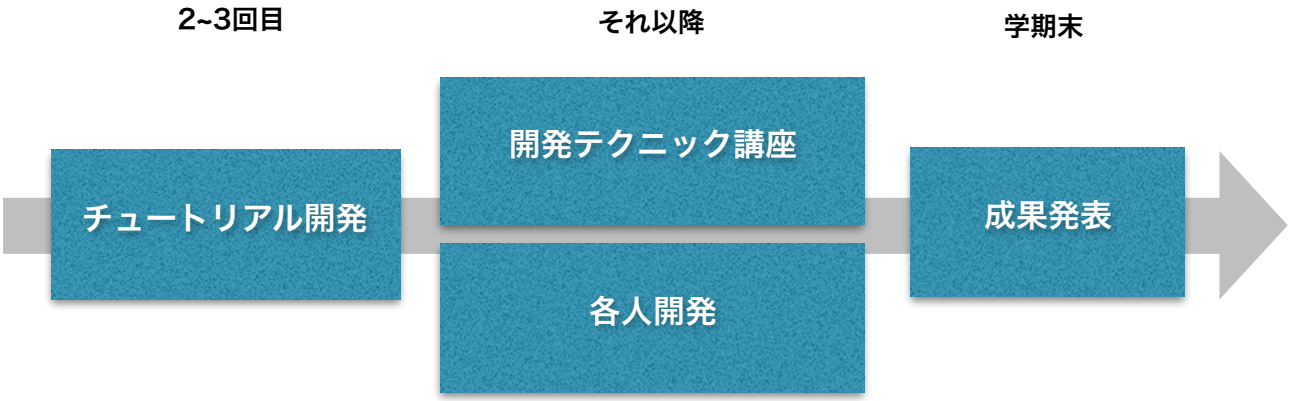
課外プロジェクト 2014
~実践的プログラミング 1~

2014 4/14 d8161105 渡部未来

1 この授業について

　会津大学の授業ではリテラシーでワークステーションの基本操作を学び、プログラミングの授業でプログラムの基礎を学ぶ。しかし、実践的なアプリケーション開発について学べる授業は少ない(興味がある方は、ソフトウェア工学I & II や ソフトウェアスタジオの授業を履修すると良い)。そこで本授業では、実践に近い形でアプリケーション開発の基礎を学ぶ。

1.1 授業スケジュール



　この授業では最初の数回を利用して、**チュートリアル開発**ということで、基礎的な環境作りと簡単なアプリ開発を通してアプリケーション開発の「いろは」を学んでいく。それらを学んだのちに、各人が本格的に自分の作りたいアプリケーションを設計・実装をする。

1.2 成果物

　特にこだわりが無ければプログラミング言語***Ruby***を使ったCUI、GUIアプリケーション、***Ruby on Rails***を使ったWebアプリケーションなどを作成してもらう。しかし自分の使いたい言語やフレームワークがある場合は、自分の判断でそれらを使っても構わない。

1.3 成果発表

　最終的に学期末には、各人が作ったアプリケーションの成果を発表する。発表は5~10分程度で行なってもらう。必要に応じてデモなどの用意もするように。アプリケーションが完成していない場合は、中間発表ということで現状を報告する。

2 シェル

開発を行うときには、ターミナルによる操作回数が多くなる。開発だけでなく会津大学で私生活をおくる上でも必須である。シェルを整備するだけで作業の効率が何倍にも上がるので、暇があれば整備をしておこう。以下に例として普段の効率を上げるためのシェル設定を２つ紹介する。既に自分のシェルを確定している人は、自分のシェル文法を押さえておこう。

2.1 ログインシェルの設定

```
$ chsh
```

chshは、ログイン時のシェルを変更するコマンドである。

```
Old Shell: /usr/local/gnu/bin/bash
New Shell [ /bin/sh /bin/csh /usr/local/bin/tcsh /usr/local/gnu/bin/bash ]:
```

以上のように聞かれるので、***bash***のパス(**/usr/local/gnu/bin/bash**)を打ち込もう。

2.2 よく使うコマンドはaliasに

bashの設定を変更するには、ホームディレクトリの***.bash_profile***を変更しよう。最後の行に以下の二行を書き込もう。

```
# ls -F 色付き
alias ls='ls -F --color'
```

#から始まるのはコメント行である。**ls**コマンドを打つときは、ただの**ls**コマンドよりもディレクトリ、実行ファイル、、シンボリックリンクなどが区別出来たほうがいいため、**-F**オプションを付けよう。また視覚的な判別がしやすいように **--color** オプションも付けよう。

2.3 プロファイルの再読み込み

シェルの設定は再ログインすれば反映されるが、即時反映したい場合は以下のコマンドを使う。

```
$ source .bash_profile
```

これで無事に**ls**が色付いて入れば成功だ。

2.4 現在位置をプロンプトに表示したい

cdをする度に自分の位置を確認するために**pwd**をタイプしていないだろうか？頻繁に行なっているなら相当なタイムロスになってしまう。そういう場合はプロンプトにカレントディレクトリのパスを表示してしまえばいい。以下を***.bash_profile***の最終行に追加すると良い。※なお、***bash***では**=**の後に空白を入れるとエラーとなるため気をつける。

```
# プロンプトにカレントディレクトリを表示
export PS1='[ \w ] $ '
```

この時の大括弧[]や$、スペースなどは見栄えのために用意しているものだ。重要なのは**\w**である。これを使うことでカレントディレクトリのパスを表示することができる。編集が終わったら**プロファイルの再読み込み**を再びおこなって、変更を確認しよう。

2.5 まとめ

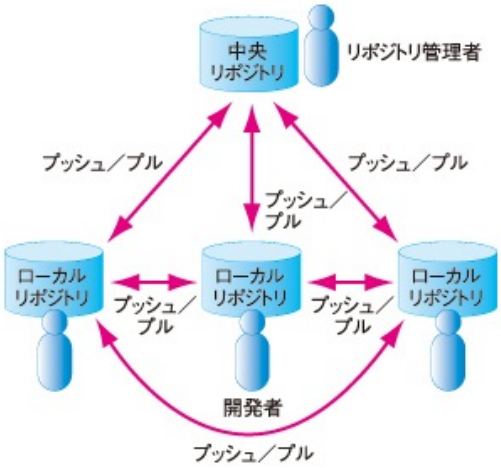
この節では、普段一番触る機会が多いターミナルのシェルの拡張の基礎について触れた。何か環境を拡張するときは、頻繁に触るものほど効果は絶大だ。更なる効率化を目指す場合は、シェルの拡張だけでなく、エディタ(emacs, vi)の拡張などにも手を出してみるといいだろう。

3 ソースコード管理

ソースコードを適切に管理することは、開発の第一歩である。定期的なバックアップ、バージョン管理、他人とのソースコードの共有が上手く行えれば、障害時のリスク回避、作業効率、技術力の向上に繋がる。

3.1 バージョン管理

ソースコードを適切に管理するためには、バージョン管理システムを導入することでおこなえる。



具体的にソースコードを管理するということは以下の点を管理するということである
・差分を管理する
・バックアップをとる
・他人とソースコードを共有する
バージョン管理システムでは、ソースコードを管理する一つの単位を**リポジトリ**として扱う。リポジトリでは差分を管理して、失敗した作業の状態を復元したり、変更を確認したりすることができる。開発者が扱うリポジトリ単体のことを**ローカルリポジトリ**と言う。チームで開発を行う場合は、各人の作業内容をマージする必要があるため、サーバーにもリポジトリを設ける。これを**中央リポジトリ**と言う。中央リポジトリは、個人ごととは別にリポジトリを管理するので、バッ

クアップと他人とのソースコードの共有するという意味も持つ。他のリポジトリの変更を取得することを**プル**と言い、他のリポジトリへ変更を知らせることを**プッシュ**と言う。

3.1 git

バージョン管理システムの１つとして、**git**を紹介する。この資料では最低限の解説しかしないため、詳しい解説を求む場合は、サルでもわかるGit入門 (http://www.backlog.jp/git-guide/) 無料電子書籍Pro Git(http://progit-ja.github.io)を参考にすると良いだろう。

3.1.2 gitの導入

```
$ wget http://git-core.googlecode.com/files/git-1.7.7.4.tar.gz
$ gunzip git-1.7*
$ tar xf git -1.7*
$ cd git-1.7*
$ ./configure
$ make
```

大学の環境には***git***が用意されていないため、自分で***git***を取得し、インストールする必要がある。以下のコマンドを叩くとインストールが行える。

ホームディレクトリに***git-1.7.7.4***というディレクトリが出来るので、***.bash_profile***でパスを通すように。

3.1.3 gitの基礎

```
$ cd # 一応ホームディレクトリへ戻る。
$ mkdir git_sample_project
$ cd git_sample_project
$ echo ‘hello git!’ > memo
$ more memo
```

バージョン管理システムでは、プログラムのプロジェクトディレクトリをリポジトリとする。そのため、まず、簡単なサンプルプロジェクトを作ろう。

```
$ git init                              # カレントディレクトリにgitリポジトリを作成
$ git add .                             # 変更ファイルを追加
$ git commit -m ‘first commit!’  # 変更を確定
```

このプロジェクトのルート・ディレクトリは、***git_sample_project***なので、そのディレクトリで以下のコマンドを打つ。

```
$ echo ‘aiueo’ >> memo                  # メモに文章を追記
$ more memo                             # メモの中身を確認
$ git add .                             # 再び変更を追加
$ git commit -m ‘change contents’       # 変更を確定
```

```
$ git status
$ git log
```

これで最初のバージョンを作り上げることができた。続いて*memo*ファイルの内容を変更してみよう。

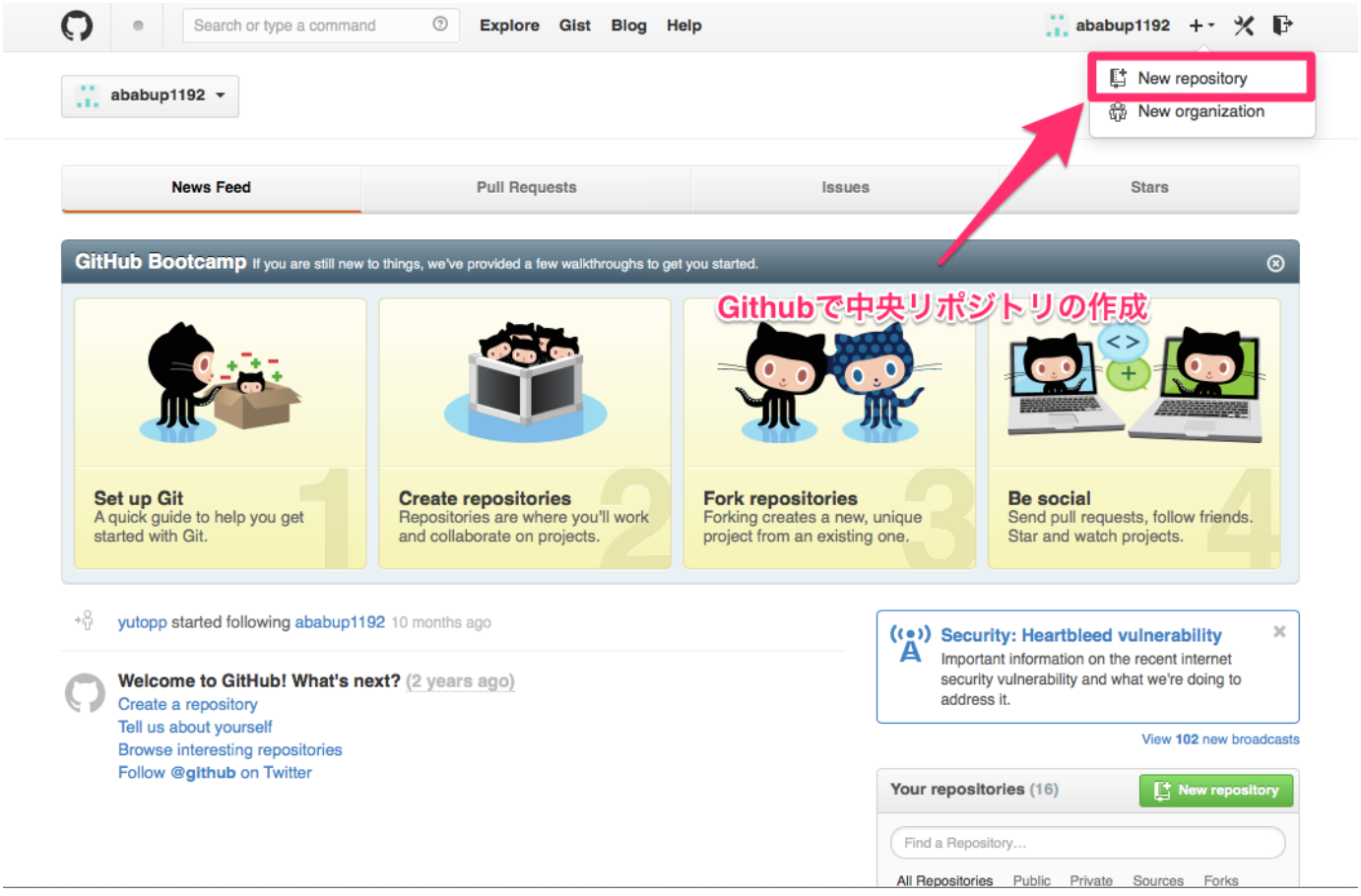
　これで無事二回目のバージョンを追加することに成功した。
以上のコマンドを使うことで変更を確認することができる。

3.2 github

　*git*を使って差分を管理することができたが、その結果を確認することはコンソール上では扱いが大変である。また、サーバー上にリポジトリをプッシュしていないため、バックアップとして機能していない。そのため中央リポジトリ (3.1の図を参照)を用意しなければならない。サーバーを用意するのも一苦労なので、*github*（https://github.com)というリポジトリ管理をしてくれるクラウドサービスを利用する。*github*は様々な人々が利用するソーシャルリポジトリサービスである。プログラミング技術を学びたい場合は、他人のコードを参考にするのも大変勉強になる。

3.2.2 githubの利用

　*github*の登録が済んだら、ログインをする。ログイン後に自分のユーザ名の隣の+ボタンを押し、**New repository**を選択しよう。フリープランではpublic (公開リポジトリ)しか選択できない。学生の場合はmicroプランに無料で加入する方法もあるので、検討してみるといいだろう。もし自由にprivateリポジトリを扱いたい場合は、*bitbucket*（https://bitbucket.org)というリポジトリサービスに登録するといいだろう。今回は、リポジトリ名を*git_sample_project*にして、**Create repository**ボタンをクリックしよう。



3.2.3 githubでのssh設定

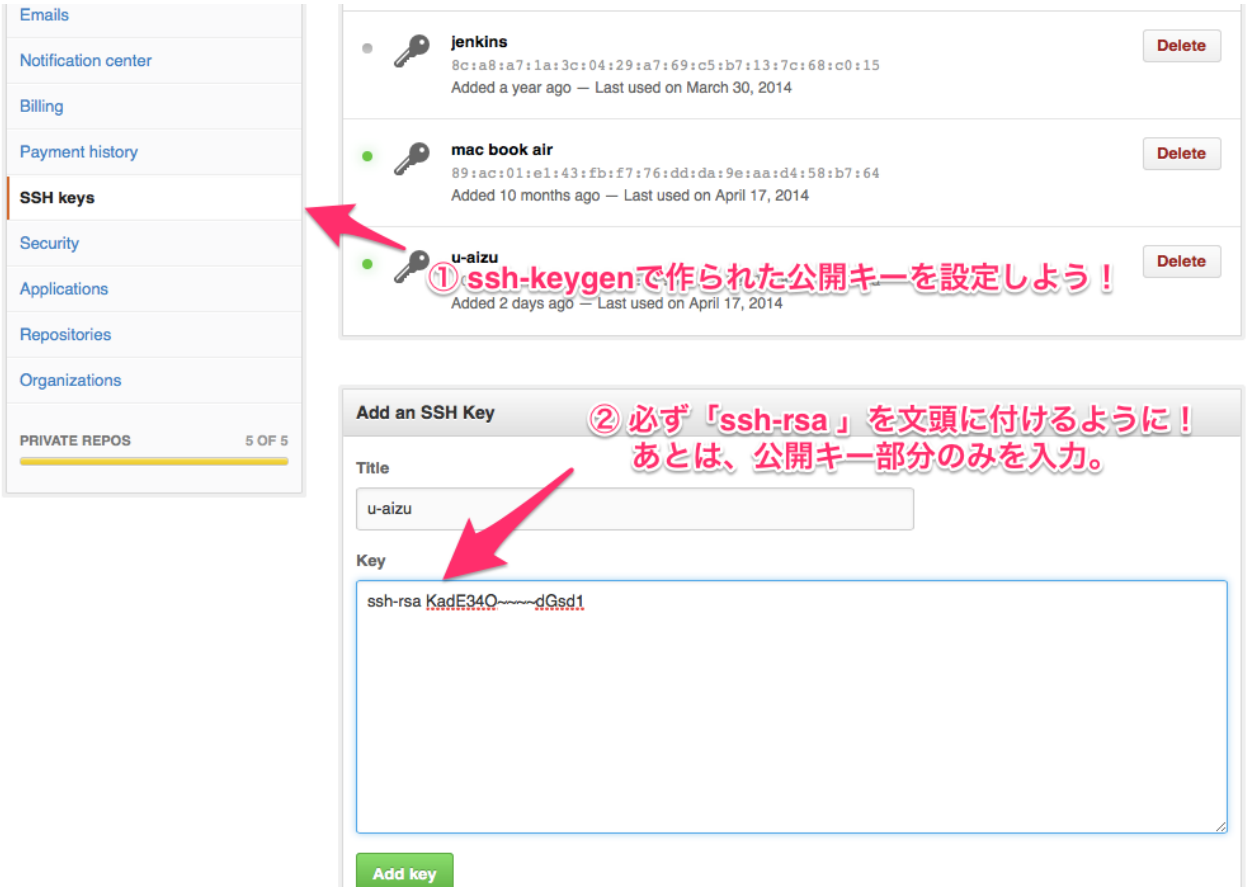
```
$ ssh-keygen -t rsa
```

gitのリポジトリのプッシュは、**ssh (Secure Shell)**によるセキュアな方法を使う。以下のコマンドを使って公開キーを作成する。パスフレーズやパスワードを聞かれた場合はブランクでOK。

```
$ cd ~/.ssh2
$ more id_rsa_2048_*.pub # 多分こんな名前 * には a,b などが書いてある。
$ chmod 600 rsa_2048_*  # .pub ではない、秘密キーは危険なので必ずパーミッションを閉じる
```

作成したら**~/.ssh2**ディレクトリへ移動して公開キーを取得する。
その中のうち必要なキーの部分(AAkV9z~~~みたいな部分)だけをコピーする。

githubの設定画面(右上のスパナとドライバーのアイコン)の**SSH keys**のリンクをクリックし、**Add SSH key**のボタンをクリックする。タイトルを入力し、文頭に「**ssh-rsa** 」と入力したあとキーをペーストする。



3.2.4 githubへプッシュ

sshの設定が終わったら、***github***へ先ほど作ったローカルリポジトリをプッシュしてみよう。

```
$ git remote add origin git@github.com:ユーザ名/git_sample_project.git
$ git push -u origin master
```

ターミナルへ戻り以下のコマンドを打つ。あとは***github***でリポジトリを見てみれば、差分や変更時間などか詳細に確認できる。

3.2.5 コミットやプッシュのタイミング

ひと通り***git***の使い方が分かったところで、**コミット**のタイミングや**プッシュ**のタイミングは、いつすればいいか気になるだろう。**コミット**つまり変更を確定するタイミングは、小規模なプロジェクトなら、1ファイルの追加、修正につき**コミット**をする。大規模なプロジェクトなら、1機能の追加、修正をする度に**コミット**をすることを心がけよう。**プッシュ**は障害が怖いなら**コミット**のタイミングと同じタイミングで、気にしないならば、作業したその日の最後に**プッシュ**をするといいだろう。**コミット**や**プッシュ**の頻度が少なければ少ないほど修正や障害に巻き込まれるリスクが高くなるので、なるべく頻度を上げよう。

3.2.6 まとめ

gitはソースコードを管理するためのツールだが、何も開発だけに限ったことではない。普段授業で扱っているソースコードも貴重な資源である。***github***や***bitbucket***でそれらのコードを一元管理することで学外からもソースコードを閲覧、コピーすることができるので、授業の予習、復習などに役立てることもできる。

3.2.7 課題

本課外プロジェクトの課題は、未提出・未実行でも評定には影響しない。しかし、確実にスキルアップに繋がるので是非チャレンジしてみよう。

・授業で提出したファイルを***bitbucket***で管理してみよう。
・***bash***もしくは、自分のシェルを改造して便利にしてみよう。
・***.bash_profile***などのファイルを***github***や***bitbucket***で管理してみよう。
・**ssh**の仕組みをもっと学んでみよう。
・***git***の使い方をもっと学んでみよう。