# 学術研究のためのオープンソース・ソフトウェア(6)Git

MIYAZAKI, Kenji / 宮﨑, 憲治

(出版者 / Publisher)
法政大学経済学部学会
(雑誌名 / Journal or Publication Title)
経済志林 / The Hosei University Economic Review
(巻 / Volume)
84
(号 / Number)
4
(開始ページ / Start Page)
133
(終了ページ / End Page)
159
(発行年 / Year)
2017-03-20
(URL)
https://doi.org/10.15002/00013820

【研究ノート】

# 学術研究のためのオープンソース・ソフトウェア（6）：Git

宮　崎　憲　治

## 1　はじめに

論文を書く際に文章を付け加えることだけでなく，減らす場合も多々ある。減らした後に，やっぱりもとに戻したくなるときもある。LaTeX や R ではソースコードをコメントアウトしておけばよいが，うっかり削除してしまった経験もあるだろう。そうならないよう，`file20130201.tex` など日付をつけてファイル管理する人もいるだろう。しかしながら，そのやり方だと同じ内容のファイルが無数に作成され混乱してしまうかもしれない。

共同研究の際に，ファイルをメイルでやりとりすることもが多いため日付や名前などを付ける人もいるだろう。メイルのやりとりだと論文を送った後，自分は手出ししないと決めているかもしれない。そうすると相手に書いてもらう番に，相手が忙しくなり論文にとりかかれず，完成が遅れるかもしれない。それぞれの原稿をダウンロードしてファイルを送りあうため，同じような内容のファイルが無数に増えていくことになる。

共同論文の管理として，Dropbox の共有機能が使えるだろう。たしかに便利であるが，競合ファイルが発生したときに統合が面倒となったり，過去の履歴が無料ユーザーの利用範囲だと30日しかないという制約がある。そのため，過去のいくつかのバージョンがそのまま置かれることになる。

こうしたことに対応するために，バージョン管理ソフトが昔から存在している。20年ほど前に私は RCS（Revision Control System）をいじっていたが，当時はひとつのファイルしか管理出来なかった。そのあと，プロジェクトごとを外部にリポートリポジトリとして集中管理する CVS（Concurrent Versions System）が流行り，ついこの間まで，その拡張版の Subversion が主流となっていた。

現在はプロジェクト管理はリポートリポジトリが一つの集中管理から複数の分散管理がトレンドとなっている。その代表的ソフトとして Git やHg がある。Git は，Linux を開発したリーナス・トーバルズが OS のバージョン管理のために作成したソフトである。Hg はスクリプト言語の Python のバージョン管理に使われている。現在は圧倒的に Git がバージョン管理に用いられている。

ちなみに Subversion に備わっていて，Git に備わっていないのは，ファイルのロック機能である。エクセルやワードなどのバイナリで作られたファイルの場合，これがないと競合問題が発生して困ることもある。ただ,これまで紹介した tex ファイル，R ファイル，LyX ファイルはすべてテキストファイルであり，ファイルの競合が発生した場合，差分が表示されて，マージすることができる。

この研究ノートはバージョン管理ソフトウェアを紹介し，それが学術研究にどのように使われるかを解説する。構成は以下のとおりである。まず Git の導入と使い方を説明し，リモートリポジトリとして有名なサイトである GitHub とBitbucket を中心に紹介し，便利に Git をつかうための GUI クライアントを幾つか紹介する。最後に学術研究のために私がどのように用いているのか解説する。

## 2　Git の導入と使い方

### 2.1　インストール

Windows では,

```
http://code.google.com/p/msysgit/
```

より，ダウンロードして実行すればよい。執筆時点でベータ版となっている。後述する GUI クライアントの SourceTree を導入する際には，内蔵の Git が導入することができるのでそれを利用してもよい。

Mac ではあらかじめ Xcode のコマンドラインツールを導入していれば，自動的に付随している。これを導入していない場合，もしくは最新版を導入したいのなら,

```
http://code.google.com/p/git-osx-installer/
```

で，ダウンロードして実行すればよい。もし，Mac で homebrew[*1] を導入しているのなら，ターミナル上で

```
sudo brew install git
```

と実行すれば, 最新版が利用できる。

Ubuntu は

```
sudo apt-get install git
```

とする。もし最新版を導入するなら

```
sudo add-apt-repository ppa:git-core/ppa
sudo apt-get update
sudo apt-get install git
```

と実行すればよい。

[*1] http://brew.sh/index_ja.html

## 2.2 使い方の基本

まずは，インターネット上に

```
http://git-scm.com/book/ja/
```

にある書籍を読んでみることを勧める。翻訳がおかしければ，原書を辿ればよい。その他，使い方は書籍やインターネット上に日本語も英語も豊富にある。以下，簡単に使い方を述べる。

最初に使用する場合，

```
git config --global user.name "ユーザー名"
git config --global user.email "メイルアドレス"
```

を設定しなければならない。SourceTree など後述の GUI アプリをインストールすれば，インストールの際にこれらを設定してくれる。

ローカルリポジトリの作成には 2 種類のやり方がある。ひとつめは，Github.com など後述のリモートホストで公開しているローカルリポジトリから作成する。そのために

```
git clone <リモートホストアドレス>
```

とすればよい。つぎに既存のフォルダからローカルリポジトリを作るなら

```
cd ~/project/test
git init
```

とすればよい。

ローカルリポジトリを作成後，変更履歴を記録していくが，そのまえに Git には 3 つの領域があることを覚えてもらいたい。まず「作業ディレクトリ」はユーザーが実際に作業する場所である。そして「ステージングエリア」は変更履歴として次のコミットに含めたいところを記録する場所で

ある。最後に「リポジトリ」は全ての変更履歴を記録している場所である。

この作業ディレクトリ上で text.txt を編集したあと

```
git add test.txt
```

とすればステージングエリアに登録される。なお

```
git add.
```

とファイル名を指定しなければ，当該プロジェクトのすべてのファイルが選択される。ステージングエリアの登録を取り消すためには

```
git reset HEAD test.txt
```

とする。もしいくつかのファイルを除外したければ .gitignore というファイルを作成して，除外ファイルを書き加える。くわしくはマニュアルを参照されたい。

ステージングエリアに登録するファイルを選んだあと

```
git commit -m "message"
```

とすればリポジトリに登録される。なおオプション -m "message" をつけないと，指定している標準エディタが起動する。Mac だと最初のエディタは Vi である。エディタからだと複数行のコミットメッセージが記述可能である。

バージョン管理は編集して，コミットを繰り返していってリポジトリ記録するのが基本である。現在の状態を見るには

```
git status
```

とする。オプション -sbx をつけるとコンパクトに一行表示になる。またこれまでの記録を見るには

```
git log
```

とすればよい。オプション `--oneline` をつけるとコンパクトに一行表示になる。それぞれの変更について例えば `f6fd1e790f336115f71519d3cf3dd7a151e6b156` といった長い ID がつけられている。これは SHA と呼ばれる。複数のコミット ID で識別可能な最初の数桁だけ入力すればよい。もしくは `git tag` でタグをつけると便利である。

過去を比較するには

```
git diff
```

とすると, 現在の作業ディレクトリとステージングエリアの比較ができる。なおステージングエリアに何もなければひとつ前の直前のリポジトリと比較となる。ステージングエリアに移動したあと直前のリポジトリとの比較は

```
git diff --staged
```

もしくは `git diff --cached` とする。ステージングエリアあるなしにかかわらず

```
git diff HEAD
```

とすれば直前のリポジトリと比較となる。

現在の作業ディレクトリと特定のSHA（コミットID）の比較は

```
git diff SHA
```

であり，過去を比較するには 2 つの SHA について

```
git diff SHA1 SHA2
```

とすればよい。

他にもある時点の変更履歴を確認する `git show` や，ファイルの行単位の変更履歴をみる `git blame` などがある。詳しくはマニュアルを参考にされたい。

### 2.3 ブランチ

Git の最大の特徴は簡単にブランチを作成できることである。ターミナル上で

```
git branch -n develop
```

とすると develop というブランチが作られる。そのブランチに移動するには

```
git checkout develop
```

とすればよい。作成したブランチのリストおよび現在のブランチをみるには

```
git branch
```

でよい。また

```
git branch -d develop
```

とすればブランチを削除することができる。

あるブランチを別のブランチに取り込むことをマージという。例えば master ブランチにいて，develop ブランチのコードをマージするには

```
git merge develop
```

とすればよい。マージしたとき，4 種類の結果が出力される。

1 . Already up-to-date

2 . Fast forward

3 . Auto Merge

4 . Conflict

Already up-to-date とでたとき, 両方とも同じだったので変更がないことを意味する。Fast forward とは一方ほうが古くて，もう一方の環境をそのまま反映したことを意味する。Auto Merge は両方とも変更があるけれど「賢く」取り込めたことを意味し, それに失敗すると Conflict が発生する。

Conflict が発生したときの対応の具体例を示す。例えば `text.txt` で以下のように表記されたとしよう。

```
<<<<<<< HEAD
a
=======
b
>>>>>>> develop
```

このとき

```
a or b
```

のように修正して,

```
git add text.txt
git commit
```

とすれば統合されたファイルをコミットできる。わからなくなれば，後述のようにマージ自体を取消せばよい。

バイナリファイルなど，時にはテキストファイルでもどちらかの箇所を採用したいことがある。

```
git checkout --ours text.txt
```

とすれば現在のブランチ（master）のファイル `text.txt` が採用される。

```
git checkout --theirs text.txt
```

とすれば統合しようとするブランチ（develop）のファイル text.txt が採用される。そうして

```
git add text.txt
git commit
```

とすればよい。

マージには，他のブランチを取り込む git rebase や他のブランチから取捨選択する git cherry-pick などがある。これらについてもマニュアルを参照されたい。これらの作業は過去を書き換えるのでリモートリポジトリを利用している場合には注意が必要である。

### 2.4　いくつかの変更

ファイルをコミットしたあとにタイポなどコミットメセージを変更するまでもない修正が必要になるかもしれない。このように前回のコミットをやり直したい時

```
git commit --amend
```

とするよい。コミットメッセージの変更も可能になる。

ファイル名を変更したい場合は

```
git mv text.txt text2.txt
```

とする。またそもそもファイルを削除するには

```
git rm text.txt
```

とすればよい。

作業中の変更をなかったことにするには

```
git checkout HEAD text.txt
```

とする。HEAD は直前のコミットを意味する。これを別のSHA（コミットID）を指定すればそのコミットの変更が適応される。全てのファイルを元に戻すには

```
git reset --hard HEAD
```

とすればよい。フラグ `--hard` 以外にも `--soft` およびその中間の `--mixed`（オプションなし）の場合がある。これらの違いについてはマニュアルを参照されたい[*2]。

過去の特定のファイルを取り出したいことがある。このとき現在 master ブランチにあるなら，取り出したいファイルを

```
cp text.txt text-tmp.txt
```

と別名に保存し，取り出したい過去の SHA を ID とすると

```
git checkout ID text.txt
```

する。そうすれば過去のファイル `text.txt` と一時的に名前を変えた `text-tmp.txt` を比較しながら編集することができる。

他にも過去の変更を「安全に」打ち消す `git revert` や，複数のコミットをまとめたり取捨選択をインタラクティブにおこなう `git rebase -i` などがある。これらについてもマニュアルを参照されたい。これらの作業は過去を書き換えるのでリモートリポジトリを利用している場合には注意が必要である。

[*2] http://qiita.com/annyamonnya/items/d845597606fbabaabcad

## 3　リモートホスト

複数のパソコンでバージョン管理を実施したいこともあるだろう。また学術研究において複数の研究者と共同研究をおこないたいこともあるだろう。この Git をサポートするリモートホストがいくつも作られていて，そのどれかのホストを利用すれば先に述べたことは実現可能である。以下幾つかのリモートホストと使い方と，それを便利にアクセスできるための SSH を解説する。

### 3.1　GitHub

この Git を対応したリモートホストとして, GitHub とBitbucket がある。特に Git が広まった理由の一つに GitHub が挙げられる（図 1 ）。GiHub はソフトウェア開発プロジェクトのための共有ウェブサービスであり，GitHub 社によって運営されている。無料でアカウントを作ることが出来きて，多くのオープンソースソフトウェア開発者が自身のプログラムを GitHub で公開している。不特定多数のひとが公開したコードをローカルリポジトリにクローンし，改良して，変更をオリジナルに取り込んでほしいと Pull Request をおこなう。こうしたことをソーシャルコーディングという。このソーシャルコーディングの中心にあるサイトが GitHub である。

利用するには本家のサイト（`https://github.com`）に行きアカウントを獲得すればよい。オフィシャルな使い方は以下を参考されたい。

```
https://help.github.com/
```

また以下のサイト

```
https://try.github.io
```

で Git および GitHub の使い方がインタラクティブに体験できる。GitHub について多数の本が出版されている。たとえば大塚（2014）がある。

## 3.2 Bitbucket

ただ GitHub は無料のアカウントだとすべてのリースコードを公開しなければならないので，プログラム技能に自信がない場合や共同研究のリポジトリには向いていない。非公開にしたいためウェブ上にリモートリポジトリを無料で設定するには，Dropbox を利用するか，Bitbucket のアカウントを手に入れればよい。イントロで紹介した Dropbox については後述する。

Bitbucket はソフトウェア開発プロジェクトのための共有ウェブサービスであり，Atlassian 社によって運用されている。商用プランと無料アカウントの両方を提供しているが，GitHub と違い無料アカウントでも非公開にでき，執筆時点で 5 人までのユーザーが使用できる。筆者が専門とする経済学においては共同研究者が 5 人を超えることがほとんどなく，少人数の共同研究に Bitbucket を用いることができるだろう。最初非公開で共同研究し，その後公開するという使い方も可能である。

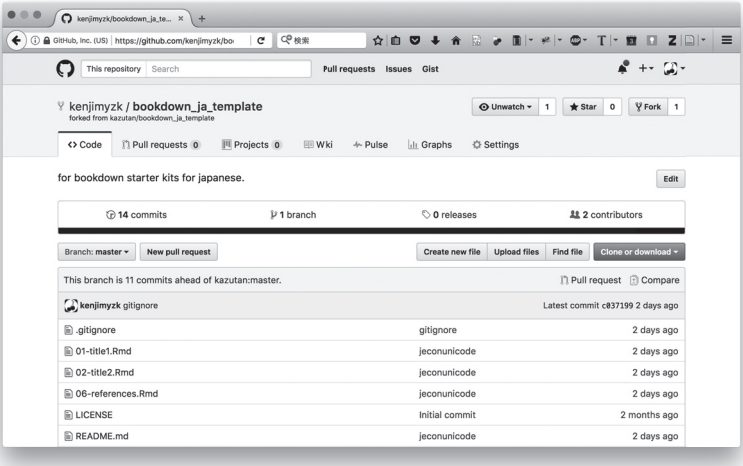


図 1　GitHub

利用するには本家のサイト（https://bitbucket.org/）に行きアカウントを獲得すればよい。本家のサイトのドキュメンテーションが充実している。

```
https://confluence.atlassian.com/bitbucket/
```

上のサイトには Bitbucket だけでなく, Git そのものや同じ会社が作成する SourceTree の使い方も解説してある。ただ日本語で書かれた出版本はあまり見かけないが，数少ない例外は大串他（2015）である。

### 3.3　GitLab

私がおこなう学術研究において, 扱うデータは公開されているものだが, 研究者によって使用を制限されている機密データを扱うときがある。もしくは 5 人を超える共同研究のためのプライベートリポジトリを作成したいこともあるだろう。そのような場合，先の GitHub や Bitbucket の利用が憚れることがあるだろう。そのため自分のサーバー上にリモートリポジトリを作成したいこともある。こうした目的のために GitLab（https://about.gitlab.com）を使えばよい。GitLab は Git サーバーを作成するためのオープンソースソフトウェアである。

利用するには https://about.gitlab.com/downloads/ にアクセスして，利用する OS を選択すればインストール方法が書かれてある。Ubuntu では利用可能であるが Mac や Windows では使えない。一方で Amazon Web Service（AWS）で運営することが可能である。たとえば以下を参考されたい。

```
http://qiita.com/morozumi_h/items/128d3254fd2eb4671966
```

他にも Git サーバーを作成するためのオープンソースソフトウェアはいくつもある。例えば Gitbuket（https://github.com/gitbucket/gitbucket）も知られているが私は試したことがない。

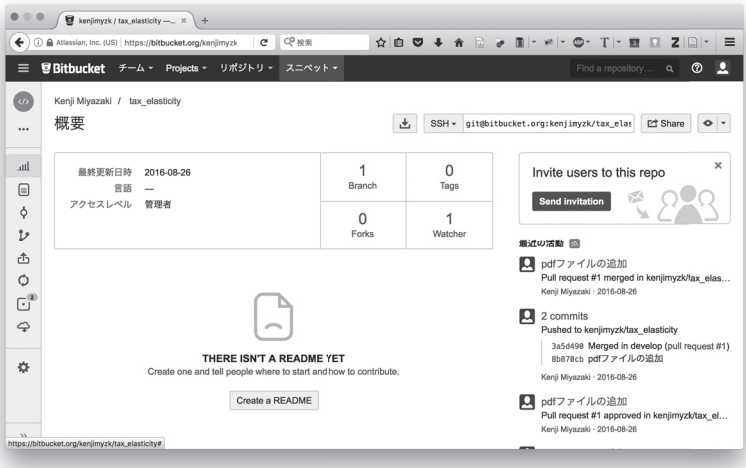


図２　Bitbucket

### 3.4　使い方

ここではリモートリポジトリでの使い方を述べる。はじめに Dropbox を例にとり，それを他のサーバーでどのようにコマンドラインで実行すればよいかを解説する。Dropbox 多くの人が利用しているストレージサービスである。Windows や Mac だけでなくUbuntu でも利用できる。複数のコンピューターやユーザでのファイル共有だけでなく Git のリモートリポジトリにも使える。

Dropbox を共有リポジトリに設定するにはターミナル上で

```
mkdir ~/Dropbox/share/repository.git
cd ~/Dropbox/share/repository.git
git --bare init
cd ~/project/test
git push ~/Dropbox/share/repository.git master
```

とすればよい。

他の人が，この共有ディレクトリにアクセスして，作業ディレクトリをつくるには

```
mkdir ~/temp
cd ~/temp
git clone ~/Dropbox/share/repository.git
```

とする。

それぞれの作業ディレクトリで

```
git pull ~/Dropbox/share/repository.git
```

として，ファイルを取り込むことができる。競合が発生すれば，ファイルの違いを確認しながら変更する。取り込んだファイルを編集した後

```
git push ~/Dropbox/share/repository.git master
```

とすれば，リモートリポジトリに変更が更新される。

ローカルブランチで develop を作成し，それをリモートに反映させるには

```
git push ~/Dropbox/share/repository.git develop
```

とすればよい。ローカルブランチで develop を削除し，それをリモートに反映させるには

```
git push ~/Dropbox/share/repository.git --delete develop
```

とする。

さらに自分だけのリモート管理だけでなく，Dropbox の共有機能を使えば，複数ユーザーで管理が可能である。ただコンフリクトが起こす可能性があるので利用には注意が必要で，商用の GitHub や Bitbukect を用いた

ほうがよい。

なお，GitHub や Bitbucket でリモートホストを作成した場合，`git@github.com:miyazakikenji/sample.git` といったURL が `origin` という短縮形で表せる。なのでこれらのリモートホストを利用した場合，先ほどの `git push` や `git pull` などのコマンドで登場した `~/Dropbox/share/repository.git` は `origin` と置き換えればよい。

先ほどの `git pull` コマンドは取り込んでマージを同時におこなっているが取り込むだけなら `git fetch` を利用する。また，一時的に作業を中断するには `git stash` を実行すればよい。これらについてはマニュアルを参照されたい。

複数ユーザーでリモートブランチを管理する場合，自身が管理者でなければファイル変更の権限がなく，`git push` を実行することは Pull Request を実施するということを意味する。管理者がこうした Pull Request を精査して取り込むことになる。管理者が複数いる場合，マージでコンフリクトを起こすと非常に面倒になる。作業を始める前に常に `git pull` からはじめて，リモートの新しい環境を取り込めた fast-forward で作業をするように心がけたほうがよい。

## 3.5 SSH

リモートで管理する場合，更新等にいちいちユーザー名やパスワードを求められる。この煩わしさを安全に避けるためには SSH の公開鍵を利用すればよい。SSH は暗号技術をつかった，リモートコンピュータと接続する技術である。昔の telnet はもはや使わなくなっている。SSH 接続は通常，公開鍵と秘密鍵の 2 つの鍵を使用する。公開鍵を相手のパソコンに登録することで安全に接続することができる。

公開鍵はMac やUbuntu の場合，ターミナル上で

```
ssh-keygen
```

で作成できる。作られた公開鍵は `~/.ssh/id_rsa.pub` に作られる。

Windows の場合，Git for Windows を導入していればコマンドラインから作成できるが，SourceTree や RStudio の GUI アプリケーションからでも作成できる。これらを導入してから，GUI での作成をすすめる.

この公開鍵をリモートホストに登録するには以下を実行する。まずファイル `~/.ssh/id_rsa.pub` の中身をクリップボードにコピーする。GitHub の場合，ログイン後右上のアカウントのところをクリックして

```
Settings ＞ SSH and GPG Keys ＞ New SSH Key
```

で先ほどコピーしたものを貼り付けて，鍵を追加する。

また Bitbucket の場合，同様にログイン後右上のアカウントのところをクリックして

```
アカウントの管理 ＞ SSH キー ＞ 鍵を追加
```

で先ほどコピーしたものを貼り付けて，鍵を追加する。

## 4　GUI クライアント

コマンドラインで，競合ファイルの差分を解決したり，バージョン管理を行うファイルを選別したり，また別のブランチを管理するのはいささか面倒である。それをグラフィカルに実行するアプリケーションは以下のようにいくつかある。

```
https://git-scm.com/downloads/guis
```

以下私が使ったことがある 4 つを紹介する。なお Windows の場合, これらのソフトについてデフォルトでインストールされる Git を用いないのなら，別途 Git のパスを設定する必要があることに注意されたい。

### 4.1 SourceTree

SourceTree は，Bitbucket を提供している Atlassian 社によって開発された GUI クライアントである（図３）。オープンソースソフトウェアでないが無料で利用可能である。Mac では，現在 SourceTree を使うのがデファクト・スタンダードとなっている。その後 Windows 版が登場して，Windows でも人気が出ている。

コマンドラインを一切触らずに，Git を扱うことが可能である。メニューも日本語化されている。日本人の利用者も多く，書籍も多く出版されている。例えば松下他（2014）では，Git コマンドとそれに対応する SourceTree の作業方法が紹介されている。

公式サイト

```
https://www.sourcetreeapp.com/
```

に行きダウンロードして実行すればインストールされる。最初にGit を導入していない場合でも自動的に導入してくれる。使い方は Bitbuket の際に紹介したサイトに行けば，SourceTree のドキュメンテーションが提供され


**図３　SourceTree**

ている。

### 4.2 SmartGit

SourceTree はMac とWindows しか提供していない。Ubuntu でも使えるGUI クライアントとして, syntevo 社が開発している SmartGit がある（図4）。オープンソースソフトウェアでないが, 営利目的以外になら執筆時点で無料で利用可能である。学術研究は営利目的に該当しないだろう。またGUI 操作をしたときにどのコマンドラインを実行したかを表示してくれるので，SourceTree より深く Git の仕組みが理解できるようになる。ただ，メニューは英語のままであり，リモート接続は SSH 接続などが前提になっているため，SourceTree 比べて少し敷居が高いかもしれない。

インストールについては，Mac や Windows の場合，本家サイト

```
http://www.syntevo.com/smartgit/
```

からバイナリをダウンロードして実行すれば自動的にインストールされる。なお Java[*3] が導入されていないとインストールされないので先に導入しておく必要がある。

Ubuntu は上述のサイトからdeb ファイルをダウンロードして

```
sudo gdebi smartgit-#_#_#.deb
```

とする。#_#_# はバージョン名である。なお gdebi をまだインストールしていなければ，まず

```
sudo apt-get install gdebi-core
```

を実施する必要がある。ドキュメンテーションは以下にある。

```
http://www.syntevo.com/doc/display/SG/Manual
```

[*3] https://java.com/ja/

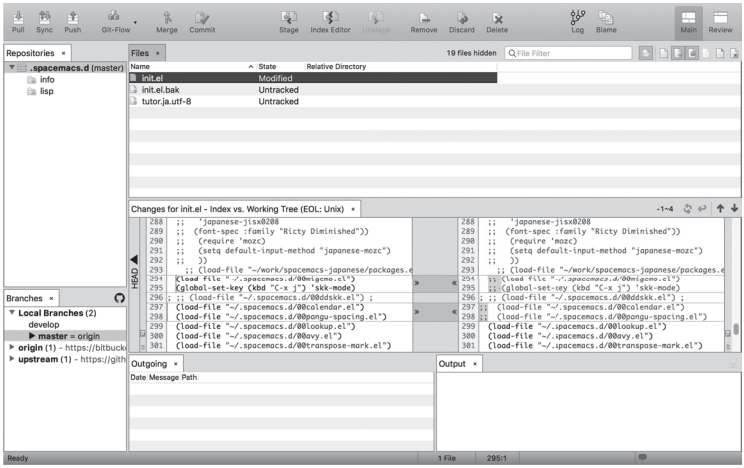


**図4　SmartGit**

### 4.3　RStudio

RStudio（`https://www.rstudio.com/`）は宮崎（2016a）で紹介したが，プログラミング言語 R の開発環境統合である。RStudio は Subversion と Git に対応している。R をつかったフォルダをプロジェクトに指定して，バージョン管理ができる（図5）。コミットをして，コメントを書いて差分を見たり，過去のファイルを取り出したりすることができる。

過去のファイルを取り出してそれを一時的に別名ファイルとして保存できる機能が便利である。それによって過去に削除してしまった機能を取り込んだりすることができる。ただブランチはすでにあるものを切り替えすることができるが，新たに作ることができない。

Ubuntu とMac は Git のパスが通っているのでそのまま使えるが，Windows の場合，［Tools］→［Global Options］→［Git/SVN］に移動して「Git executable:」でパスを指定する必要がある。また Mac でもデフォルトの Git が `/bin/git` となっている。homebrew の Git を使いたければ「Git executable:」を `/usr/bin/git` に変更する必要がある。

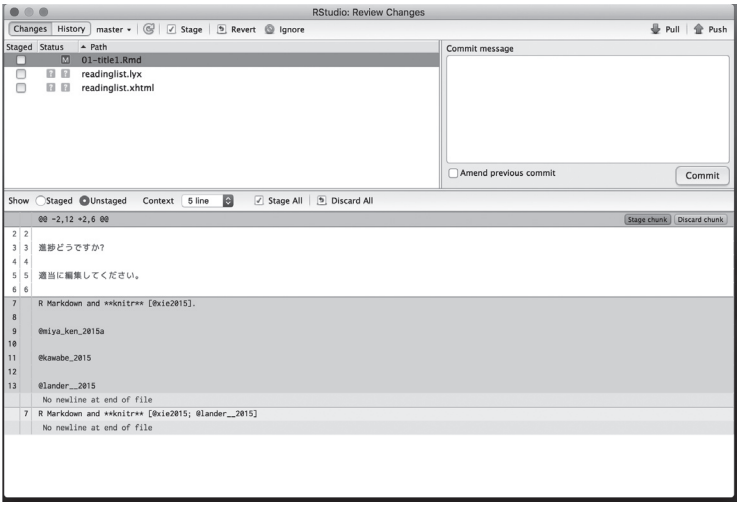


**図5　RStudio**

その他くわしい使い方は以下を参考にされたい。

`https://support.rstudio.com/hc/en-us/articles/200532077-Version-Control-with-Git-and-SVN`

### 4.4　LyX

LyX（`https://www.lyx.org/`）はグラフィカルに LaTeX 文書を作成するツールである（宮﨑, 2016b）。これから RCS もしくは Git を扱うことができる。ただ Git の機能は昔のバージョン管理システム RCS の機能と同等で,コミットと変更点の差分などを示すことだけである。ブランチを作成したり切り替えたりすることはできない。

正しく Git のパスが通っていれば，LyX から `git init` を実行できないため，ターミナルか他のクライアントで実施する。そうすると［ファイル］→［バージョン管理］に［変更をチェックイン］,［改名］,［レポジトリ版に戻す］,［旧改訂と比較］,［履歴を表示］が実施可能である。名称が RCS

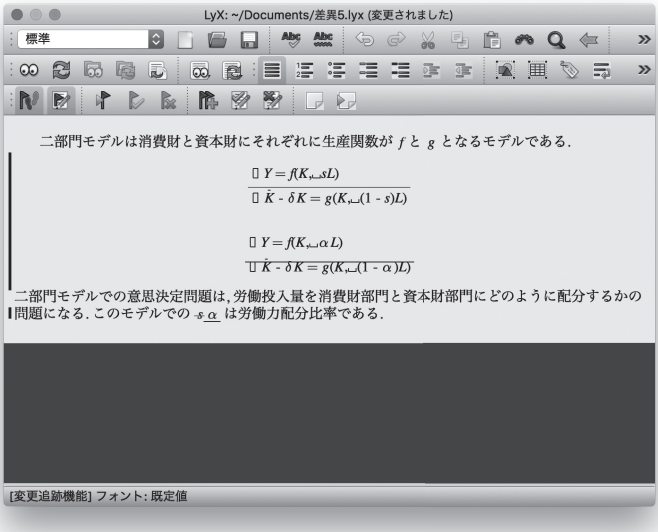


**図6　LyX**

の名残のままであるが，Git での `git -a commit`, `git mv`, `git reset`, `git diff`, `git status`, `git log` に対応している。旧改訂と比較について，Microsoft Word の変更履歴のようにどこが違っているのかをグラフィカルに示してくれる（図 6）。

ただ他のブランチとマージした際にコンフリクトを起こすとソースファイルを直接編集する必要がでてくるかもしれないので注意が必要である。編集するには Vim だと，LyX 用のプラグインがある。以下のサイトから入手できる。

```
https://wiki.lyx.org/Examples/VimForEditingLyx-files
```

その他，使い方は情報がいささか古いが以下を参考にされたい。

```
https://wiki.lyx.org/LyX/VersionControlInstallationAndUsage
```

### 4.5　使いわけ

以上, 私が利用したことがある GUI クライアントを紹介した。どのように使いわけたらいいのか個人的な意見を述べる。R でデータ分析をして, 論文を書く場合，ローカルのバージョン管理は RStudio を使って作成すればよいだろう。LyX 上ではマージのコンフリクトが発生しないよう注意して, コミットする, もしくは過去の差分を見るときのみに利用する。RStudio でこまめにコミットをおこない，一区切りがついたら，履歴を整理したり，他のブランチへマージするときに SourceTree もしくは SmartGit を活用すればよいだろう。Ubuntu を使わないのなら，SourceTree だけでよいだろう。

## 5　ワークフロー

研究プロジェクトのワークフローをまとめてみよう。まず研究プロジェクトごとにフォルダをつくる。たとえばフォルダ名を `project` としよう。プロジェクトのなかには文献リストと研究ノートと論文のためのファイルが入っているだろう。

文献リストは `project.bib` としてまとめられる。宮﨑（2015）で紹介したように Zotero で文献管理し，bib ファイルとしてここのフォルダに出力する。日本語文献を用いる場合には `jecon.bst` も必要であろう。

研究ノートとして, 詳細な数式展開を LyX でファイル `project.lyx` を作成し, 全ての計算結果を RStudio でファイル `project.Rmd` を作成する。`project.Rmd` に対して `knitr` を実行すると `pdf` もしくは `html` で結果が出力されるようにする。同時に論文作成のために図表（`tab.tex`, `fig.pdf`）が出力するようにも設定しておく。もしくは `project.Rmd` に対して `purl` を実行すると, 外部で実行するためのプログラム `project.R` を出力するようにする。必要に応じてオリジナルのデータファイル `data.csv` が必要に

なることもあるだろう。

文献リストと研究ノートをもとに学術論文を作成する。論文は研究ノートの一部を利用することになる。多くの日本人研究者は研究ノートは日本語で作成し，論文は主に英語で執筆することになるだろう。論文のファイル形式は `article.lyx` となる。文献リスト（`project.bib`），図表（`tab.tex, fig.pdf`）を読み込む形で，ヒューマンエラーを最小限にするように記述する。再現可能性を求められる場合は，`project.R` を用いるようにする。学術論文は基本が英語であるが日本語で書く必要もあるだろう。日本語の論文も書く場合 `jarticle.lyx` を作成する。

論文ファイルやプレゼンファイルについて，オリジナルは研究ノート（`project.lyx, project.Rmd`）である。少なくとも理論に関わる変更点は常に研究ノートを最新にして更新する。そしてその結果をコピペでなく出来る限り自動的に結果を組み込めるようにしておく。これらのファイルはテキストファイルなので Git によりバージョン管理しておく。

ちなみに研究成果の発表については英語なら `beamer.lyx`，日本語なら `jbeamer.lyx` を作成するようにする。計算結果の図表は自動的に結果を組み込めるようにしておく。プレゼンファイルについて発表場所によってページ数や表題が変わるけれど，複数のファイルを生成するのでなく Git のブランチ管理で対応したほうがよい。

ところで Github.com が GitHub Flow というワークフローを提案している[*4]。大塚（2014）よりまとめると以下である。

1．新しい作業をするときは master ブランチから記述的な名前のブランチを作成する
2．作成したローカルリポジトリにコミットする
3．同名のブランチを GitHub のリポジトリに作成し，定期的に push する
4．フィードバックが欲しいとき Pull Request を作成し，Pull Request

[*4] `http://scottchacon.com/2011/08/31/github-flow.html`

でやりとりする

5．他の開発者がレビューし，作業終了を確認したら master ブランチにマージするとなっている[*5]。

この GitHub Flow は学術研究に応用できるだろう。単著で論文を書く場合 develop ブランチを作り，そこで作業をする。こまめにコミットして，ひと段落するたびに master ブランチにマージする。複数のパソコンで共有するなら，リモートリポジトリを作成してもよい。Github.com はプライベートリポジトリを作成するなら有料のため，無料でプライベートリポジトリをつくれる Bitbucket を利用する。そこで定期的に Push する。

共著の場合，Bitbucket を利用する。Bitbukect は無料で共同利用できる人数は限られているが，経済学での共同研究なら十分である。リモートの master ブランチを編集できる人は限定して，develop ブランチを更新していく。そしてそれぞれの共著者ごとのブランチをつくり，それぞれ作業をして，develop にマージしていく。ある程度まとまってから master ブランチにマージする。

なお共同研究者でまた Git に慣れていないのなら，マージをしてコンフリクトがおこったとき対処できない可能性がある。そうならないよう，マージをする際に Fast-forward の原則が守られるように，事前連絡をするように心がければよい。もしくはそれぞれの develop ブランチに直接マージするのでなく，Pull Request するようにして，責任者がレビューのうえマージすればよい。

もし共同研究者が Git が使えないのなら PDF での出力も Git で管理する。そして，Bitbucket などにある課題（issue）設定をして，そのもとの掲示板上でやり取りすればよいだろう。

単著でも共著でもある程度完成して意見を聞きたいのなら，Bitbucket での閲覧許可をすればよい。そこで広く意見を聞き，論文をより仕上げて，学術雑誌に投稿するようにすればよいだろう。もしくは GitHub において

[*5] 実際はデプロイについても言及しているがここでは省略する。

公開してもよいだろう。このように Git をつかったバージョン管理は今後の研究の主流になると考えられる。

## 6 おわりに

これまで何回かにわたって学術研究のためのオープンソースを紹介した。研究のコアとなる目的は，再現可能な学術論文を PDF で作成することである。そのためには，図表作成はコピペを使わず，原データから自動的に出力できるようプログラムを書き，これまでの変更履歴を記録する必要がある。

このために中心となるソフトウェアは LaTeX, R, Git である。それをより便利にあつかうために Texmaker, RStudio, LyX を紹介した。あと文献管理について Zotero も紹介した。これらのソフトは全てオープンソースである。これにより再現可能な論文をヒューマンエラーを最小限にして作成すること可能となる。

経済学者にとって LaTeX は，約20年前の私が大学院生のときから知られており，R はここ最近，人気になっている。Git は，現在，経済学者の間であまり知られていないけれど，ほとんどのプログラマがこれを利用して自分のソースコードを管理している。近い将来，Git を利用できない研究者が取り残される可能性は高いだろう。

現在，コンピュータを取り巻く環境は日進月歩で変化している，数値計算の世界で，オープンソースである Python がこれまでデファクトスタンダードであった商用ソフトウェア MATLAB を凌駕しつつある。さらに Julia という比較的若いプログラム言語が経済学者の間で注目されている。この Julia, Python, および既に紹介済の R は Jupyter[*6] として，ブラウザベースでプログラム作成が可能になる。おそらく Jupyter を用いた研究ノ

[*6] http://jupyter.org/

ートが今後のトレンドになりうる。また20年前から愛用しているが必ずしも人にすすめられなかった Emacs も現在 Spacemacs[*7] として劇的に変化しつつある。これらのソフトウェアも機会があれば紹介したい。

## 参考文献

大串肇・久保靖資・豊沢泰尚（2015）『Git が，おもしろいほどわかる基本の使い方33〈バージョン管理，SourceTree，Bitbucket〉』，エムディエヌコーポレーション，東京。

大塚弘記（2014）『GitHub 実践入門 ~Pull Request による開発の変革』，技術評論社，東京。

松下雅和・船ヶ山慶・平木聡・土橋林太郎・三上丈晴（2014）『開発効率をUPする Git 逆引き入門』，シーアンドアール研究所，新潟。

宮﨑憲治（2015）「学術研究のためのオープンソース・ソフトウェア（2）: BiBTEX と Zotero」，『経済志林』，第83巻，第2号，119–149頁，11月。

――― （2016a）「学術研究のためのオープンソース・ソフトウェア（3）: R とRStudio」，『経済志林』，第83巻，第3号，143–178頁，2月。

――― （2016b）「学術研究のためのオープンソース・ソフトウェア（5）LyX」，『経済志林』，第84巻，第1号，137–163頁，9月。

[*7] http://spacemacs.org/