Git / GitHub を利用したオープンソース LMS 更新作業の効率化

外村 孝一郎 [1)], 寺崎 彰洋 [1)], 梶田 将司 [2,3)]

1) 京都大学 企画・情報部 情報基盤課
2) 京都大学 情報環境機構 IT 企画室
3) 京都大学 学術情報メディアセンター

tonomura.koichiro.8c@kyoto-u.ac.jp

An Efficient Update Process Using Git and GitHub for an Open Source LMS

Koichiro Tonomura[1)], Akihiro Terasaki[1)], Shoji Kajita[2,3)]

1) Planning and Information Management Department, University
2) Institute for Information Management and Communication, Kyoto University
3) Academic Center for Computing and Media Studies, Kyoto University

概要

京都大学では、オープンソースの Sakai CLE をベースにした学習支援システムを導入し、”PandA(People and Academe)” のブランド名で全学に提供している。オープンソースによるシステムは、コミュニティを通じた機能改善、バグフィックス、情報共有などの利点があるが、本学への導入に当たっては独自環境に応じたカスタマイズの必要があり、運用・サポートコストを圧迫するため、効率化が求められていた。本学では、Git および GitHub を利用することでシステムのアップデート作業の大幅な効率化を行った。本報告では、本学で行った省力化について技術面を中心に述べる。

1 はじめに

京都大学では、2013 年度から、Sakai CLE をベースにした学習支援環境 PandA を導入し、全学に向け提供している [1]。Sakai CLE は Apereo Foundation が中心となって開発が進められているオープンソースのコースマネージメントシステムであり、研究大学を中心に広く採用されている [2]。オープンソースによるシステムは、コミュニティを通じた機能改善、バグフィックス、情報共有などの利点があるが、サービス提供にあたっては、本学独自のセキュリティ格付けへの対応や教務情報システムとの連携などの機能追加を行った [1, 3]。また、システムの安定的運用にあたって、セキュリティ・バグフィックス・利便性の向上の観点から、開発コミュニティによる Sakai CLE 側のバージョンアップに追随した更新作業を実施する必要があり、学期毎ないし年度ごとの更新が欠かせない。

しかしながら、開発元のバージョンアップをそのまま適用すると、ソースコードのコンフリクトやセキュリティ対応などの問題がしばしば発生する。そのため、本学で独自にカスタマイズした部分の開発・更新を同時に行う必要があり、運用・サポートコストを圧迫しており、省力化・効率化が求められていた。

これらに対応するため、当初、バージョン管理システム Subversion を利用して管理・情報共有を行っていたが、2015 年より Git および GitHub を導入すると共に、独自で行ったカスタマイズ内容の整理を進め、大幅な省力化を図った。本報告では、本学で実施した Git および GitHub を利用したオープンソースベースの学習支援システム PandA のバージョンアップ作業について紹介する。

2 Git によるバージョンアップ対応

本学では、学習支援環境 PandA の導入に当たって、Sakai CLE 2.9.3 をベースに、機能追加やセキュリティ格付け対応などの改変を行ったが、十分な変更履歴の管理・分離が行えておらず、バージョンアップの度にそれらの機能との整合性維持に大幅な開発コストを必要としてきた。これらのバージョンアップ作業に対応するため、Sakai プロジェクトでも利用されていたプログラムのソースコードなどの変更履歴を記録したり追跡するためのバージョン管理システムの 1

つである Subversion を使用していた。しかしながら、Sakai プロジェクト側が Subversion から GitHub に移行することになったため、本学でも Git を導入しバージョンアップ対応を実施することとした。

2.1 コミットの機能毎の分離

Git では変更を記録するためにコミットという操作を行う。ひとつのコミットに含まれる前回からのコミットとの差分には一つの機能に関する変更のみが含まれることを基本とする。それにより、ある一つの機能が不要になったときなどに、その機能に対応するコミットで行った変更を取り消すことで機能が削除可能になる。また、その機能の実現のために行った変更を確認するためには、そのコミットだけを確認すればよいため、管理がわかりやすくなる。まず、Git の機能を利用して、現在開発中のソースコードを、ベースである Sakai CLE と本学独自に機能追加・変更した部分の二つに分離した。続いて、本学独自に機能追加・変更した部分を「機能」と「コミット」を一対一対応させるべく分離を進めた。

最初に、何もコミットされていない状態 (Initial commit) をコミットする。次に Sakai CLE からオリジナルのソースコードをコピーしてコミットし、一旦それを削除した後、現在開発中のソースコードをコピーしてコミットする。これにより、ソースコードを二つに分離できる。以下に、git の log コマンドを利用して確認した状態を示す。

```
% git log --pretty=format:'%h %s'
44f1db7 misc
669592b sakai-2.9.3-all
d56492f Initial commit
%
```

git のコミットには前回との差分が記録されている。そして、最初のコミット d56492f には何も含まれていない。従って、コミット 669592b には　Sakai CLE バージョン 2.9.3-all に含まれるファイルが全て記録されている。もっとも上位のコミット 44f1db7 misc には本学独自の変更分が全て含まれている。この misc を、可能な限り機能ごとのコミットに分離した。加えて、Sakai CLE のオリジナルのソースコードに対して、2.9.3-all だけでなく、以前のバージョンの情報を反映することで、Sakai CLE がどのような更新を積み重ねてきたのかを確認できるようにするためのコミットを追加した。

```
% git log --pretty=format:'%h %s'
d26b167 misc
65f5902 sakai-2.9.3-all
09f1997 sakai-2.9.2-all
7c5404b sakai-2.9.1-all
0de9f6e sakai-2.9.0-all
a2b078c sakai-2.8.x-all
d56492f Initial commit
%
```

次に、コミットの機能ごとの分離の例を示す。たとえば、2.9.3-all にはコミュニティでも既知の不具合 LSNBLDR-305 があり、そのままだと Sakai CLE の起動時に不具合が発生する。この機能コミットは misc のソースコードに含まれているため、これに対応した変更を一つのコミットとして分離すると手順を以下に示す。

- git rebase -i 09f1997 コマンドによりインタラクティブに修正を開始
- LSNBLDR-305 に対応する修正をソースコードに加え記録
- git rebase –continue コマンドでコミットを反映

この操作で、LSNBLDR-305 に対処するための変更は一つのコミットとして整理され、その他の変更部分との二つに分離された。

```
% git log --pretty=format:'%h %s'
7cce4a7 misc
22fb011 LSNBLDR-305: 起動しない
65f5902 sakai-2.9.3-all
```

以上の作業を繰り返すことで、独自に追加したソースコードを機能毎に分割する事を行った (図 1)。

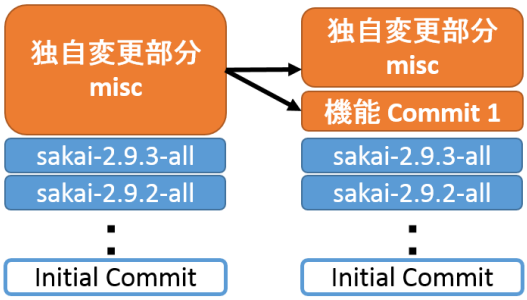

図 1　ソースコード分割

2.2　バージョンアップ対応

次に、機能ごとに分割されたコミットをバージョンアップされた Sakai CLE に適用する。今回は Sakai 2.9 に独自機能を追加して、サービスを提供したが、git の rebase 機能を利用して、Sakai 10 への機能追加を行う。最初に、Sakai 10 のオリジナルのソースコードのみで構成されたブランチを作成し、Sakai 2.9 から rebase 機能を利用して、base を変更する (図 2)。

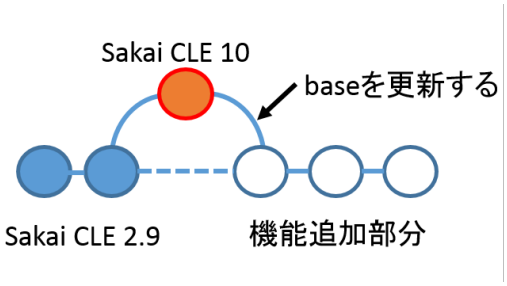

図 2　rebase による base の変更

もし、Sakai 10 と独自追加したコミットの間でソースコードのコンフリクトが発生していた場合、該当するコミットの適用をスキップした後、コンフリクトしたコミットを git の cherry-pick 機能を利用し、後から個別に適用する。その際、当該コミットの機能を実現するために、ソースコードの変更を必要とする場合は随時対応する (図 3)。

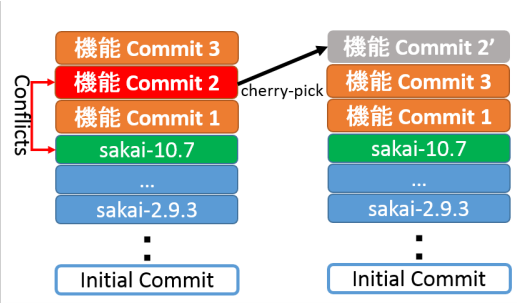

図 3　コンフリクトの解消

以上の作業を全ての機能コミットに対して、適用の可否を判断しつつ実施することで、最終的にサービスイン可能なバージョンアップ版が完成する。ここで注意すべき事項として、Sakai 10 側で新たに追加された機能に対しては、本学側でそのまま適用可能かどうかの判定は別途必要となり、必要に応じて新たに機能コミットを作成する必要がある。

3　GitHub によるチーム開発環境の構築と情報共有

開発チーム内での情報共有の効率化を図るため GitHub を利用している。Git では個人が利用するローカルリポジトリと、複数で利用できるリモートリポジトリの機能を利用でき、GitHub は Git で利用できるリモートリポジトリを提供している。利用開始に当たっては、セキュリティ設定を考慮し、本学の情報格付けに抵触する部分を分離し、GitHub 上に広く公開可能な情報と学内の開発者のみ共有する情報を分離して公開している ”https://github.com/kyoto-u/” 。前節で解説した機能コミットの管理のため、GitHub の Issues 機能を利用している。

- 機能追加や bugfix を行う前に issue として GiuHub 上に登録
- コミットを行う際必ず、対応する issue 番号の情報を付与

これらを行うことで、開発チーム内での問題点の共有、課題情報管理の効率化を行った。本チームでは学内のセキュリティ格付けに影響しない部分のソースコードを GitHub 上に openpanda として公開しており、オープンソース開発に興味のある学生などに公開し、機能の追加提案や改善を取り入れることも行っている。

4　まとめと今後の課題

本報告では、バージョン管理ツールによる京都大学での Sakai CLE のバージョンアップ省力化について述べた。大きな機能追加が行われるメジャーバージョンアップには QA(品質管理) 等が必要となるため、導入までの期間を必要とするが、マイナーバージョンアップにおいては、各学期開始までに最新版を反映したサービスの提供を行うことが可能となった。2016 年度後期授業においてはマイナーバージョンアップで最も直近にリリースされた Sakai 10.7 によるサービスを全学に提供している。今後の課題としては、

- コミットが完全に一対一対応していない
- サービスバージョンとコミット番号の管理が明確にできていない
- 開発コミュニティによる bugfix と本学による bugfix の整理が不十分

などが挙げられる。また、本学で開発されたツールや改良点を Sakai 開発元 Apereo Foundation へ提供し、開発コミュニティへの貢献を行っていきたいと考えている。いずれの課題も Git および Github の機能を利用して実施していくことを検討している。

参考文献

[1] 外村孝一郎、京都大学における Sakai CLE による学習支援システムの導入と運用、大学 ICT 推進協議会 2013 年度年次報告、2013

[2] Apereo Foundation. https://www.apereo.org/

[3] Shoji Kajita and Koichiro Tonomura, “Course Link Tool for Loosely Engaging Sakai CLE with Student Information System”, Open Apereo 2014 Conference, Miami, FL, U.S.A., June 1-5, 2014