千葉工業大学 社会システム科学部

プロジェクトマネジメント学科

平成 26 年度 卒業論文

活動ログのマイニングによる
人的資源マネジメント
The human resources management
by the mining of the activity log

プロジェクトマネジメントコース

矢吹研究室

1142106 丸山準人／Junto MARUYAMA

指導教員印	学科受付印

目次

第 1 章

緒論

第1章　緒論

1.1 研究背景

人材マネジメントに統計分析を活用する試みがスポーツ界で広まっている．

野球界で最も有名なものとしてマネー・ボール[1]が挙げられる．その中で登場するアスレチックスという球団は，他球団に比べて資金面で劣っていたため，有名な選手を補強できずにいた．そこで，選手の成績を統計的に分析して少ない資金でチームの戦術に合う選手を獲得した．その結果，全球団の中で最高の勝率を記録した．しかし，この構成方法はあくまで確率論によるため，長期戦においては効果を発揮するが，プレーオフなどの短期決戦においては選手の力量が勝因を決めることも多いため課題はある．

サッカー界では，チームを統計的な手法で強化するのは，野球チームを統計的な手法で強化するよりも難しかった．その理由として，サッカーは野球ほど分析すべきデータが単純ではないためより高度な分析が必要となる．さらに，野球の場合は統計上の知識／分析力があればある程度は容易に分析できるが，サッカーの場合は分析力とサッカーの知識の双方において理解することが不可欠なのが特徴的である．サッカーは時代と共に戦術が変化し，必要なスキルもそれに合わせて変化することから，選手の評価基準も柔軟に変更する必要がある．

その中でセイバーメトリクスを利用して成功を収めているチームがある．それは，プレミアリーグ（イングランドのリーグ）のニューカッスルである[2]．プレミアリーグは全 20 チームが所属していてその中にマンチェスター・ユナイテッドやチェルシー，アーセナルといった経営規模が大きいクラブが多数いる．今までのニューカッスルの用いた戦術は，前線の長身の選手にロングパスを出し競り合ったこぼれ球を拾い攻撃することが多かった．しかし，その主力を引き抜かれたことと監督が変わったことにより，その戦術が大いに変化した．その戦術は，4-3-3 というコンパクトなシステムである．コンパクトにすることにより，中盤の選手が活きよりポゼッションの高い戦術に変化した．新たに定められたチームの編成基準は，「ロングパスによるチャンスメーク」よりも「ショートパスによるチャンスメーク」である．選手の基準において「ロングパスによるチャンスメーク」よりも「ショートパスによるチャンスメーク」を重視したのは，ロングパスによるチャンスメークは正確性がなくゴールに繋がりにくいが，ショートパスによるチャンスメークは正確性がありゴールに繋がる確率が高くなるからである．そこで補強した選手は，怪我をしたために他のクラブの興味を引かなくなっていた元有名選手である．その結果，チームはリーグ戦 7 位という好成績を収め，EL（ヨーロッパリーグ）への出場権を獲得した．トップ 10 が目標だったチームには EL 出場という結果は，成功と言っても良いだろう．

本研究では，スポーツ界で行われているような統計解析手法を用いた人材マネジメントの，ソフトウェア開発の現場への導入を検討する．

現在，オープンソースソフトウェア開発は GitHub 上で行われていることが多い．

オープンソースソフトウェアとは，ソフトウェアの設計図にあたるソースコードを，インターネットなどを通じて無償で公開し，誰でもそのソフトウェアの改良，再配布が行えるようにすることであり，そのようなソフトウェアの名称である．その特徴として企業，個人など参加形態を問わずに誰でもプロジェクトに参加することができる．

過去に GitHub 上で行われているプロジェクトの各メンバの活動ログを収集し，役割分担の実態を明らかにする研究が行われていた[3]．この研究で，Push する行為とリポジトリにスターを付ける行為は別のメンバが行っていることが多いことが明らかになった．

そこで，スポーツ界で行われているような統計解析手法で分析することによりプロジェクトにどのような変化をもたらすかを調査する．

1.2 研究目的

GitHub 上で多く行われているオープンソースソフトウェア開発のプロジェクトを用いて，活動ログを統計解析手法で分析する．その結果からどの役割にどのような人材が適しているかを調査する．

1.3 研究方法

研究方法は以下の通りである．

① GitHub 上に存在するオープンソースソフトウェア開発のプロジェクトを調査する
② GitHub の API[4]を利用して活動ログを収集する
③ 得られた活動ログを，スポーツ界で行われているような統計解析手法で分析する
④ 過去に得られた結果と今回得られた結果を比べて考察する

1.4 プロジェクトマネジメントとの関連

チームスポーツにおいて選手を評価する客観的な方法を確立することは，プロジェクトにおいてメンバを評価する客観的な方法の確立につながる．

本研究で検証する手法は，PM の人的資源マネジメントに役立つだろう．

参考文献

[1] マイケル・ルイス／中山宥訳. マネー・ボール　奇跡のチームをつくった男. ランダムハウス講談社. 2004.

[2] 山中忍. プレミアリーグ版“マネー・ボール”？清貧クラブのニューカッスルが躍進. 2012. http://number.bunshun.jp/articles/-/216542

[3] 関口元基. オープンソースソフトウェア開発における役割分担の実態調査. 千葉工業大学, 2013, 卒業論文.

[4] GitHub Developer. http://developer.github.com/v3/activity/events/types/#gollumevent

第 2 章

スポーツ界の現状

第 2 章 スポーツ界の現状

2.1 マネー・ボールとは何か

ビリー・ビーンはかつて超高校級選手としてニューヨーク・メッツから 1 巡目指名を受けるほどのスター候補生だった．当時のスカウトの言葉を信じ，名門のスタンフォード大学の奨学生の権利を蹴ってまでプロの道を選んだ．しかし，自身の性格も災いして泣かず飛ばずの日々を過ごすことになった．その後様々な球団を転々として，挙句の果てに引退した．第二の野球人生としてスカウトに転進し歩み始める物語である．

2001 年のポストシーズンにオークランド・アスレチックスは，ニューヨーク・ヤンキースの前に敗れ去った．その結果，オフにはチームの核となるジアンビーなどの 3 選手が FA（他球団とも選手契約を締結できる権利を持つ選手のこと）による移籍が決定的となり，補強が迫られていた．アスレチックのゼブラルマネージャーとなっていたビリーは，2002 年シーズンに向けて戦力を整えるべく補強資金を求めるも，スモールマーケットのオークランドを本拠地とし，資金に余裕のないオーナーの返事はつれなく，補強資金は例年通りしか出なかった．ある日，イエール大学を卒業したピーター・ブランドに出会った．ピーターは各種統計から選手を客観的に評価するセイバーメトリクスを用いて，他のスカウトとは違う尺度で選手を評価していた．その理論に興味を抱いたビーンは，その理論をあまり公にできず肩身の狭い思いをしていた彼を自身の補佐として引き抜き，他球団から評価されていない埋もれた戦力を発掘し，低予算でチームを改革しようと試みた．

野球統計学から高い評価を得た選手たちを獲得するも，ビリーの思うように勝てず，また監督もその意図に反し，自身の考えに基づいた起用をした．その結果 14 連敗を喫し，周囲から批判を受けた．業を煮やしたビリーは監督が起用していた選手をトレードに出し，自身が推し進める野球に合った選手を起用させた．またそれと同時にビリーは選手とも積極的に交流し，自身が野球界に革命を起こすであろう野球理論を選手に啓発した．その結果，MLB 史上初の 20 連勝という大記録を達成した．しかし，地区予選で敗戦し，ワールドシリーズへの出場はできなかった．野球統計学はあくまで確率論によるため，長期戦においては効果を発揮するが，プレーオフなどの短期決戦においては選手の力量が勝因を決めることも多く，個々の選手の力量が低くチーム力で勝つタイプのアスレチックスはまけてしまう．だが，その野球理論に目を付けたボストン・レッドソックスは，彼らの理論が使えると判断し，ビリーに野球史上最高額ともいえるビッグオファーを提示するも，ビリーはそれを断り，アスレチックスでワールドチャンピオンを目指すべく，今も挑戦している．

そして皮肉なことに巨額な資金を使えるレッドソックスが，彼らの野球統計学の理論を用いたことでワールドチャンピオンになり，その「セイバーメトリクス」の有用性は一気に野球界へと広がった．

2.2 マネー・ボールが野球界に与えた影響とその後の結果

野球統計学をチーム運営に応用する場合，以下のようなことを行う．

- 「出塁率」などおおくのスカウトが着目しない指標により選手を評価し，当時の球界では低評価だった選手を獲得し，チームも好成績を上げた．
- トレードでも年棒と適正評価（適正価値）が吊り合わない選手は，容赦なくトレード要員として放出し，その代わりチームに必要で若くて年棒の安い選手を獲得，FA は年棒高騰に繋がるため基本的に引き止めない．
- 費用対効果の高い選手に関しては年棒が低い時期に長期複数年契約をすることで年棒の抑制を図る．
- ドラフトでも「将来性」という不確実性を排除し，即戦力になる大学生を優先的に獲得する方針にする．また，選手の身辺調査を行い，須高に問題があり，なおかつ更生不可なら選手としての能力が高くても獲得はしない．

こういったものが挙げられる．しかし，これらの手法はあくまでも資金の貧しい球団が資金が豊富にある球団と互角に戦えるために行っていた「弱者の戦略」でしかない．こういった手法がチーム運営やスカウティングに広まるに連れて，出塁率が高い選手の獲得競争が激化し安価で獲得できないなど，ビリーが行っていたマネー・ボールの優位性は薄れてしまう．しかしながら，マネー・ボール自体も発展途上のため，様々な変化や発展があり，その中でチームごとにカスタマイズされていくのではないだろうか．また，投手力がウリのチームはそれに合ったスカウティングをし，野手育成に定評のあるチームはそれに合ったスカウティングをする，その際にこれまでの理論を応用しているのではないだろうか．

2.3 日本野球に与えた影響と代表的な事例

日本において最も有名なのが「野村 ID 野球」がこの野球統計学（セイバーメトリクス）に当たる．野村監督が行ったセイバーメトリクスは，「プロのスカウトのみによる主観的な選手評価に頼るのではなく，客観的なデータに基づき，多額の投資を行うリスクを回避する」ことである．

この野球統計学を日本で初めて本格的に導入し，大きな成功を収めた球団が，「北海道日本ハムファイターズ」である．ベースボール・オペレーション・システム[1]（球団の保有している選手の能力・立ち位置を見えるか（数値化）して，分類している IT データベースのこと）に多額の投資をし，選手の能力を出来るだけ細分化して数値評価に置き変えている．ただこれはあくまでもシステムであり，全てのチームがこれを導入しても必ずしも上手くいくとは限らない．大事なのは「活用できる力」である．

2013 年度のペナントレースは低迷していたが，その問題として考えられるのが「監督」だろう．これは，原作の中でも実際にあった．ゼブラルマネージャーが獲得してきた選手だとしても最終的な起用の権限は監督にある．その問題を解決するために日本ハムは「監督に最小限の権限」しか与えない．日本ハムが目指すチームとして「編成部が導き出した

チームや選手の育成プランをきちんと実践し，なおかつコーチや監督がプラスアルファの効果を生み出し，そして監督やコーチに依存することのないチーム」が挙げられる．

2.4 マネー・ボールがサッカー界に与えた影響

2.4.1 選手の獲得方法について

まず，選手の獲得について説明する．

プロ野球の場合は，ドラフト制度により所属チームが決まる．また，外国人選手の登録枠も制限がある．それに対して，J リーグの場合は，試合に出場できる人数制限はあるものの，外国人選手も含めてすべて自由競争での獲得となっている．そのため，選手が所属するチームを決めることができる．選手がチームを選ぶ基準として以下の例が挙げられる．

- チームの基盤がしっかりしている
- 選手の質が高く，魅力的なサッカーをする
- 自分を高く評価してくれる（高年俸）

等の理由により，将来有望な選手は J1 の強豪チームに入るのが自然の理となっている．

そこで，マネー・ボール理論で考えてみる．選手獲得の際に重要とされている資質の評価方法を，従来とは違う視点から見るということである．サッカーというのは，「点を取られなければ負けないスポーツ」,「試合終了時に 1 点でも多く得点していれば勝つスポーツ」という原点がある．その原点に立ち返り，今まで評価してきた判断基準（つまり，背が高い，足が速い，テクニックがある）を強豪チームを作るにあたって解釈を変えてみる．

野球を例にしてみる．硬式野球の経験が全くないソフトボールの選手をドラフトで指名した日本ハムが，「従来の常識を一度疑ってみて，素質で選手を獲得する」というスタンスを取ってチーム編成を行っていた．現代のような情報化社会では，どのチームも同じようなスカウティングのノウハウや世界中にちらばる選手の情報を「データ」として持っていてもおかしくはないだろう．

しかし，その「データ」をどの角度から眺めるかはそれぞれのチームのセンスで，大きな違いが生まれるのは間違いないだろう．チームが飛躍する時，それは「ローコスト・ハイパフォーマンスが実現した」時である．簡単に説明すると，「安く獲得した選手が，思った以上にチームに貢献してくれる」状態が，一度に複数起こった時に，そのチームは台風の目と呼ばれる．逆に言えば，「今まで実績で選手を獲得したものの，すでにその力が落ちている」ような場合，「ハイコスト・ローパフォーマンス」という状態になってしまう．

2.4.2 野球にあってサッカーにはあまりないものについて

以心伝心という言葉をサッカーでいうとアイコンタクトと言い換えることができる．ゲームが動いている瞬間瞬間の（次のプレーを判断する時間）短い時間の中では，たとえ有名な選手であったとしてもパスミスを犯すのは当たり前のことである．ただ，サッカーの試合では，常にボールがピッチの中で動いている訳ではなく，リスタートの瞬間というのが多々あります．その例としていくつか挙げる．

- 試合開始時のボールが静止している瞬間
- 点が入ったり，取られたりした後のキックオフ
- ファウルを取られた後の直接・間接フリーキックの場面
- PK の場面
- ボールがタッチラインを割って，スローインをする場面
- ボールがゴールラインを割って，ゴールキックをする場面

等が挙げられる．このような場面に野球のようなサインプレーでリスタートすることによりチャンスが広がるのではないだろうか.

2.4.3 マネー・ボール理論を取り入れ成功した例

マネー・ボール理論を取り入れて成功を収めたチームがいる．それは，プレミアリーグ（イングランドのリーグ）のニューカッスルであるプレミアリーグは全 20 チームが所属していてその中にマンチェスター・ユナイテッドやチェルシー，アーセナルといった経営規模が大きいクラブには足元にも及ばない．会計・監査法人大手のデロイト社による「クラブ長者番付[2]」によれば，ニューカッスルの昨季売り上げは約 9800 万ユーロ（約 105 億円）である．今季のリーグ優勝を争うマンチェスターの両雄や，残る CL 出場 2 枠を争うロンドンの 3 強の数字には一桁足りないのである．加えてニューカッスルは，3 シーズン前に 2 部リーグに降格まで経験している．それにもかかわらず，今季のニューカッスルは 7 位という結果で終わり，資金面では上にいるリバプールよりも良い成績で終えることができた.

ニューカッスルは，リバプールに多くの主力選手を引き抜かれながら，リバプールが補強に費やした資金の 5 分の 1 以下しか費やさなかった．リバプールの米国人オーナーは，野球界で成功した「マネー・ボール理論」のコンセプトの信望者として知られている．そのモットーは，「過小評価されている選手に本来の価値を見出す」ことである．ニューカッスルは，この点においてもリバプールを凌いでいると言えるだろう.

参考文献

[1] BOS（ベースボールオペレーションシステム）- Let it be - JUGEM. 2013. https://www.google.co.jp/url?sa=t&rct=j&q=&esrc=s&source=web&cd=3&cad=rja&uact=8&ved=0CCsQFjAC&url=http%3A%2F%2Fletitbe.jugem.cc%2F%3Feid%3D1531&ei=dwo_VIPaIeHUmgW_voLgAg&usg=AFQjCNGvU7yL2RN2wRteshsBRXNUUgvq3Q&sig2=tX-6O9GnyozTEbNWzCn0iw

[2] デロイト・フットボール・マネー・リーグ http://ja.wikipedia.org/wiki/%E3%83%87%E3%83%AD%E3%82%A4%E3%83%88%E3%83%BB%E3%83%95%E3%83%83%E3%83%88%E3%83%9C%E3%83%BC%E3%83%AB%E3%83%BB%E3%83%9E%E3%83%8D%E3%83%BC%E3%83%BB%E3%83%AA%E3%83%BC%E3%82%B0

第 3 章

オープンソースソフトウェアの概要

第 3 章 オープンソースソフトウェアの概要

3.1 オープンソースとは何か

オープンソースとは，プログラムのソースコードを無償で一般に公開し，誰でもそのソフトウェアの改良，再配布の自由を認めることを指す．多数の個人が浸透性をもってソフトウェアの開発，流通に携わることにより，より高品質で柔軟かつ廉価なソフトウェアの開発を実現するための概念である[1]．オープンソースは，Open Source Initiativel（オープンソース・イニシアティブ）によって策定された定義によれば，プログラムのソースコードが入手可能であることやソフトウェアの変更や派生品を許可すること，特定グループを差別したり利用分野を制限しないことなどの条件を満たす必要がある．バブリックドメインソフトウェアとは異なり，著作者の著作権を保つことができる．オープンソースの思想では，プログラムのソースコードを多くの人々に吟味させて，改良や拡張を促すことで，ソフトウェアの機能・性能が向上されていくことが期待できる．

オープンソースで開発・提供されるソフトウェアは，オープンソースソフトウェアと呼ばれ，利用形態まで含めれば，「利用者が一定の条件のもとで，自由にソースコードを利用，複写，改変，再領布できるソフトウェア」と定義されるだろう．オープンソースソフトウェアでは，オリジナルのソースコードは無償で一般に公開されるが，OSS 開発者によって定められた条件下であれば，ソースコードの入手者はソースコードを自由に利用，複写，改変，再領布することが可能である．このオープンソースソフトウェア開発者により定められた条件は，オープンソースソフトウェアライセンスと呼ばれ，オープンソースソフトウェアの流通過程において非常に重要な役割を果たしている．

また，オープンガバメントやオープンデータなど，個人や機関が保有するデータやアプリケーションなどのリソースを「オープン」にすることにより，リソースの多様な活用を促し，イノベーションを創出しようとする取り組みが各所で行われてるが，こうした取り組みの根底にあるものもオープンソースであり，オープンソースソフトウェアはこの概念が速く適用された代表的なものと言えるだろう．

3.2 オープンソースの定義

オープンソースは，単にソースコードへのアクセスを意味するものではない．オープンソースソフトウェアの配布条件は，以下の基準を満たす必要がある．

表 オープンソースソフトウェアの配布条件

	内容
Free Redistribution （自由な再領布）	ライセンスは，複数の異なるソースからプログラムを含む集計ソフトウェア配布の構成要素としてのソフトウェアを販売したり配ったら任意のパーティーを制限してはならない．ライセンスは，そのような販売のための使用料やその他の手数料を必要としないものとする．
Source Code （ソースコードの開示）	プログラムは，ソースコードが含まれている必要があり，ソースコード内の配布だけでなく，コンパイルされた形式を許可する必要がある．製品のいくつかのフォームは，ソースコードと一緒に配布されない場合は，無料でインターネットを介してダウンロードし，好ましくは合理的な再生産原価を超えないためのソースコードを取得する広く報道手段があるに違いない．故意難読化されたソースコードは許可されない．そのようなプリプロセッサや変換の出力のような中間形式は許可されない．
Derived Works （派生著作物に関するライセンス）	ライセンスは，改変された派生作品を許可する必要があり，それらが元のソフトウェアのライセンスと同じ条件の下で配布されるようにする必要がある．

Integrity of The Author's Source Code （著作権のソースコードの完全性）	ライセンスは，改変された形態で配布されてからソースコードを制限することができる唯一のライセンスである．ビルド時にプログラムを修正することを目的としたソースコードに「パッチファイル」の配布を許可している．ライセンスは，明示的に変更されたソースコードから構築されたソフトウェアの配布を許可する必要がある．ライセンスは，オリジナルのソフトウェアとは異なる名前やバージョン番号を選ぶために派生作品を必要とする場合がある．
No Discrimination Against Persons or Groups （個人やグループに対する差別禁止）	国家によっては外国為替法によりソフトウェアの輸出制限を行っている場合があるが，OSI ライセンスではそのような制限をかけることは許可されていない．これは，オープンソースの進化のためには多くの人が触れる必要があり，それを締め出すことは禁止されている．
No Discrimination Against Fields of Endeavor （利用分野に対する差別禁止）	オープンソースコミュニティでは，学術・商業利用のためのユーザの参加も奨励している．このため，ライセンスで使用分野に制限をかけることを禁止している．
Distribution of License （ライセンスの継承）	プログラムに付随する権利は，プログラムがそれらの当事者による追加ライセンスの実行を必要とせずに再配布されすべてに適用されなければならない．
License Must Not Be Specific to a Product （特定製品に特化したライセンスの禁止）	プログラムに付随する権利は，特定のソフトウェア配布のプログラムの幸福の一部に依存してはならない．

License Must Not Restrict Other Software （他のソフトウェアを制限するライセンスの禁止）	ライセンスは，ライセンスされたソフトウェアと一緒に配布される他のソフトウェアに制限を置かなければならない．プログラムは，その分布から抽出され，プログラムの使用許諾契約の条項内で使用または配布されている場合は，プログラムが再配布されすべての関係者は，元のソフトウェアの配布と伴わせて付与されているものと同じ権利を持つべきである．
License Must Be Technology-Neutral （テクノロジーニュートラルなライセンス）	OSI ライセンスではいかなる項目もその技術に対して制限を設けることはできない．これは FTP ダウンロードや CD-ROM での分布，ミラーサイトからの分布といったクリック・ラッピングとの相性の悪い領布方法があり，またこのライセンス方法を共用することで再領布を足止めしてしまう結果を生む危惧が発生してしまう．そのため，OSI に準拠するライセンスは特定の技術に固執することなく，技術的に中立なライセンスを保持する必要がある．

代表例に，LinuxOS などがある．Linux の場合，インターネットなどで取得し，対価を支払わずに利用できるなどの利点がある反面，商業利用を目的に改変した場合，改変部分の開示を求められることがある．

3.3 オープンソースソフトウェアの種類

オープンソースソフトウェアは，今や非常に幅広い領域をカバーしており，その種類は多様である．代表的なオープンソースソフトウェアには，Ruby，Perl，Python などのプログラム言語から，Linux や Solaris といった Operating System，Apache HTTP Server や nginx などの Web サーバーソフトウェア，Apache Tomcat や JBoss などのアプリケーションサーバソフトウェア，そして Mozilla Firefox や Apache Open Office などのエンドユーザアプリケーションがあり，これを見るだけでもオープンソースソフトウェアは多数存在することがわかるだろう．また，オープンソースソフトウェアである Linux をとってみても，Red Hat ブランドの Linux ディストリビューションや Android など Linux から派生したオープンソースソフトウェアも存在し，その種類は非常に多い．

以下では，代表的なオープンソースソフトウェアを分野別にまとめた．

表 オープンソースソフトウェアの種類

分野	オープンソースソフトウェアの種類
プログラム言語	Java，Ruby，Perl，Python，PHP， R（R 言語）など
OS（Operating System）	Linux（Fedora, CentOS, Ubuntu 含む）Solaris， Android など
ツールキット，フレームワーク	GNOME，KDE，LXDE など
仮想化環境	Qt，GTK+など
統合開発環境（IDE）	KVM，Xen など （デスクトップ仮想化にも対応）
Web サーバー	Eclipse，NetBeans，WideStudio など
アプリケーションサーバー	Apache HTTP Server，ngix など
データベースサーバー	MySQL，PostgreSQL，Firebird， OpenLDAP，Apache Derby など
データ処理	Hadoop（データ分散処理技術），HBase， Cassandra，　Redis（非構造型データベース）
拡張機能	Iptables，OpenSSL（SSL アクセス認証） ip6tables（パケットフィルタリング／NAT） TCP Wrapper（ネットワークフィルタリング）

特定アプリケーション向けサーバー	Exim，Sendmail，Postfix（電子メールサーバ／メール転送エージェント） Samba（ファイルサーバー） Squid（プロキシサーバー） ZFS（ファイルシステム
アプリケーション連携機能	Amanda（バックアップ），Liferay， JBoss Portal，eXo（ポータル） LISM（アカウント ID 管理） Open AM（SSO：シングルサインオン） Pentaho，Plone，Drupal， Jaspersoft（データ解析／BI：ビジネスインテリジェンス） DotNetNuke（検索エンジン） Selenium（ウェブアプリケーションテスト）
アプリケーション	Apache OpenOffice，Compiere，eGroupWare， LibreOffice（オフィススイート） ADmpiere（EPR アプリケーション） Scalix Zimbra（グループウェア） dotProject（プロジェクト管理） Mozilla Firefox（ウェブブラウザ） Mozilla Thunderbird（電子メールアプリケーション） osCommerce（E コマースサイト構築） SugarCRM（CRM アプリケーション） SalesLabor（CRM アプリケーション） WordPress（ブログプラットフォーム） VLC（メディア再生プレーヤー）
その他	Git（ソースコードバージョン管理） jQuery（JavaScript ライブラリ）

上記にある通り，オープンソースソフトウェアはその種類の多さもさることながら，今では特定企業が開発したプロプライエタリとしてのソフトウェアに代わり，業務に利用されるようなケースも多くみられるようになっており，システム開発の様々な分野においてそのプレゼンスは強まっている．

なお，ソフトウェアではないが，ウェブサイト上のコンテンツをオープンソース化するという取り組みがある．代表的な例として，インターネット上の百科事典サイト Wikipedia

であり，同サイト上のテキストコンテンツは，Creative Commons というライセンスのもと，誰でも自由に編集したり，複製したり，領布したりすることができる形となっている．他に，地図データベースをオープンソースで作成するプロジェクト OpenStreetMap[2]も知られている．

3.4 オープンソースの今後

オープンソースは，システム開発，運営負担が大きい大規模システムを中心に，引き続き主要なソフトウェアリソースとして活用されていくと考えられる．特に現在の IT 業界のトレンドであるクラウド，ビックデータ，オープンガバメント，そしてソーシャルメディアなどの分野では，大量のデータの利用・管理などが重要な課題となっており，それを支えるシステム基盤は大規模化する一方であることから，オープンソースソフトウェアの活用が拡大していく可能性が高いだろう．ただし，その一方で企業側に「オープンソースは使いにくい」といった意識がある点は否めないだろう．以前問題視されていた「無償ソフトウェアのためサポートが得られない」という問題は，商用サービスの展開により解決しつつあるが，もう一つの問題点として，オープンソースソフトウェアの利用の際には「オープンソースソフトウェアライセンス」に従う必要があり，このライセンスの遵守や取り扱いが煩雑であるため，利用しづらいと考える企業が増える可能性がある．

ここで，オープンソースソフトウェアとオープンソースソフトウェアライセンスについて再確認してみる．オープンソースソフトウェアとは「ソフトウェアの利用者が一定の条件のもとで，自由にソースコードを利用，複写，改変，再領布できるソフトウェア」でのことであり，ここで言う一定の条件にあたるのがオープンソースソフトウェア開発者の定めるオープンソースソフトウェアライセンスとなる．

オープンソースソフトウェアライセンスは，基本的に各ソフトウェア開発者が自由に作成･適用できるものであり，GNU General Public License, BSD License, Apache License, Commn Public License など，現在 100 以上存在していると言われている．しかし，ライセンスの内容は非常に複雑であり，個人レベルの開発者が独自のライセンスを作成することは難しいのだ．そのため，多くの開発者は，オープンソース文化の啓蒙を目的に設立された国際非営利組織 Open Source Initiative（以下 OSI）やオープンソースソフトウェアを推進する非営利組織 Free Software Foundation（以下 FSF）が認定したライセンスを流用するケースが多くなっている．

現在のオープンソースソフトウェアライセンスのほとんどは，「コピーレフト」と呼ばれる概念を何らかの形で含んでいることが多いのだ．「コピーレフト」とは，著作者が開発した（オープンソースソフトウェアやソースコード）に対する権利を保有したまま，利用者に著作物を複写，改変，再領布する自由を与えるものであるが，その一方で，著作物が複写，改変，再領布される形で派生物（二次的著作物）の領布者に対しても，もともとと全く同じ条件で派生物を領布することを義務付けるものである．著作物が領布され続ける限

り，制限なく適用され続けるほか，ソフトウェア利用者に対して，利用者がソースコードを改変した際に，改変部分のソース公開までを義務付けたり，ソフトウェア利用者がソースコードを他のソフトウェアのコードと組み合わせた際に，他のソースコードの公開までが必要となったりと，著作物の流通過程における扱いを厳格に定める概要となっている．

代表的なコピーレフトライセンスとして，Free Software Foundation が作成した GNU General Public License（GPL）がある．以下に GPL v3（以下 GPL）の概要をまとめる[3].

- ライセンシーは，領布するオープンソースソフトウェアに GPL を適用しなければならない．
- ライセンシーは，オープンソースソフトウェアをオブジェクトコード形式で領布する際，対応するソースコードを領布先に対して後悔しなければならない．
- ライセンシーは，オープンソースソフトウェアに改変を加えて領布する際，改変を加えた事実及び日付を明確にしなければならない．
- ライセンシーは，オープンソースソフトウェアをソースコード形式で領布する際，GPL の本文を明示しなければならない．
- ライセンシーは，追加的条項を加えることにより，GPL に例外を設けることが出来る．
- ライセンサは，領布するオープンソースソフトウェアに自身の特許が含まれる場合，ライセンシーに対して当該特許を無償でライセンス付与しなければならない．
- ライセンシーは，領布先に対して，領布するオープンソースソフトウェアに含まれる自身の特許に関する特許侵害訴訟を起こしてはならない．
- ライセンサが差別的な特許契約を締結した際，ライセンシーにも当該特許契約が付与される．
- ライセンサは領布するオープンソースソフトウェアに関して，いかなる保証も提供しない．
- ライセンサは領布したオープンソースソフトウェアが引き起こす損害に対して，一切責任を持たない．
- ライセンシーは，オープンソースソフトウェアを領布する際，著作権および無保証の旨が記載された定型文を明示しなければならない．

逆に，このコピーレフトの概要を含まないライセンスもある．例えば Apache License などは，オープンソースソフトウェアの利用者に改変部分のソースコードの公開までは義務付けておらず，ライセンシーには，利用するソースコードを他のソフトウェアのソースコードと組み合わせた際に他のソースコードを公開する義務も発生しない．

同ライセンスの概要は以下の通りである．

- ライセンシーは，オープンソースソフトウェアをソースコード形式で領布する際，ライセンス本文・著作権・特許・商標・帰属についての周知を明示しなければならない．
- ライセンシーは，オープンソースソフトウェアに改変を加えて領布する際，改変加えた事実をわかりやすく周知しなければならない．

- ライセンシーは，オリジナルオープンソースソフトウェアに帰属周知が含まれている際，同周知をオープンソースソフトウェアに含まなければならない.
- ライセンサは，領布するオープンソースソフトウェアに自身の特許が含まれる場合，ライセンシーに対して当該特許のライセンスを付与しなければならない.
- ライセンシーが誰かを特許違反で訴えた場合，ライセンシーのライセンサより付与された特許ライセンス権利が失効する.
- ライセンサは領布するオープンソースソフトウェアに関して，いかなる保証も提供しない
- ライセンサは領布したオープンソースソフトウェアが引き起こす損害に対して一切責任を持たない.
- ライセンサは，オープンソースソフトウェアを領布する際，著作権および無保証の旨が含まれた定型文を表示しなければならない.

なお，今後，オープンソースソフトウェアの利用拡大が見込まれるクラウド，ビッグデータ，オープンガバメントの分野についてみると，関連オープンソースソフトウェアのライセンスは，次のような状況となっている.

- クラウドコンピューティング分野：CloudStack（Apache License），OpenStack（Apache License），Eucalyptus（GPL），CloudFoundry（Apache License），OpenShift（Apache License）
- ビッグデータ分野：Hadoop（Apache License），Cassandora（Apache License），HBase（Apache License）
- その他：Git（GPL），Drupal（GPL）

このように，企業戦略に関連するシステムの構築に利用されるオープンソースソフトウェアの多くは比較的制限が緩やかな Apache License が適用されており，企業にとって利用上の障害が引き下げられていると言える．一方で，オープンガバメントで利用されているような Git や Drupal といったコンテンツ管理向けオープンソースソフトウェアについては，適用される対象がオープンな取り組みであることから，コピーレフトが明確に規定される GPL のような厳格なライセンスでも問題にならないものと考えられる.

また，開発しやすさとは相反するが，オープンソースソフトウェアライセンスの厳しさは訴訟のリスクを下げる可能性がある．プロプライエタリなソフトウェア開発企業が基本的な特許を押さえているデータベースを複製していないことを保証する手立てではなく，オープンソースソフトウェアの（商業）利用を行う企業は，常に特許・著作権侵害の訴訟リスクを負うことに留意すべきである.

これに対して，Apache License の場合，オープンソースソフトウェアをもとにある事業者が自由に改変し，商業化したソフトウェアについて，特許権又は著作権違反の訴えを受けた場合，その事業者がもっぱら訴訟リスクを引き受けることとなるが，これと比較するとソフトウェアをそのままの形で再配布することを義務付ける GPL の場合，訴訟リスクは

そのソフトウェアを利用した企業だけにとどまらず，開発コミュニティにも及ぶことから，自然と GPL ライセンスのコミュニティにおいては，他者の知的財産権侵害に対して慎重な開発体制をとらざるを得ないものと想定される．最も，GPL の場合は，対象オープンソースソフトウェアについて商業利用の禁止が規定されることも多い．

オープンソースソフトウェアがもつ訴訟リスクへの対策として，開発者個々がオープンソースソフトウェアライセンスを使い分けていくというも考えられるが，個人でライセンスの内容を吟味する労力は多大であり，かつ開発コミュニティごとにライセンス形態が決まっていることが多く，変更は困難であると想定されることから，それには時間がかかると見込まれるだろう．

参考文献

[1] The Open Source Definition. 2013.
http://opensource.org/osd
[2] OpenStretMap.
http://www.openstreetmap.org/
[3] GNU オペレーティング・システム
http://www.gnu.org/licenses/gpl.html

第 4 章

GitHub について

4 章 GitHub について

4.1 GitHub とは何か

Andreessen Horowitz の発表によると, 1 億ドルという途方もない額が GitHub に投資された．GitHub はその金で一体何をするつもりか，Andreessen Horowitz にとって良い投資だったのか，そんな巨額の投資は GitHub にとって良いことなのか，さまざまな意見や憶測が，Web の至るところに出現した．しかしそもそも，GitHub とはどのようなものなのか，デベロッパにとってなぜそんなに人気があるのか，GitHub は「コードを共有したり公開するためのサービスだ」とか,「プログラマのためのソーシャルネットワーキングサイト」だ，として通説されているがいずれも，GitHub が特別な存在である理由を説明していない．

GitHub はその名のとおり，Git を使うためのハブである．Git は Linux の作者 Linus Torvalds が始めたプロジェクトで，GitHub の訓練士 Matthew McCullough の説明によると，Git は昔からあるバージョン管理システム(version control system, VCS)の一種であり，プロジェクトの改訂履歴を管理し保存する．プログラムのコードに利用されることが圧倒的に多いが，実は Word のドキュメントでも Final Cut のプロジェクトでも，どんなタイプのファイルでも管理できる．とにかく，コンピュータのプログラムにかぎらず，どんなドキュメントでも，すべての段階の草案やアップデート履歴を保存し管理できるファイルシステムなのだ. Git 以前によく使われた CVS や Subversion といったバージョン管理システムでは, 一つのプロジェクトの複数の参加者に対し, 中央的な“リポジトリ”(repository, 保存庫）が一つだけあった．一人のデベロッパがどこかを書き換えると，その変更が直接，中央的リポジトリにも及ぶ．しかし Git は分散バージョン管理システムなので，一人一人がリポジトリ全体のコピーを保有し，そのコピーに対して変更を行う．そしてその変更が OK となったら，中央的サーバにその変更を“チェックイン”する．コードを書き換えるたびにサーバに接続しなくてもよいから，これまでよりもきめ細かい変更とその共有が，やりやすくなる．

GitHub は，いろんなプロジェクトのために Git のリポジトリをホスティングするサービスだ．独自の便利な機能がたくさんある．Git はコマンドラインツールだが，GitHub は Web 上でグラフィカルなユーザインタフェイスを提供する．セキュリティのためのアクセス制御もあり，また各プロジェクトに wiki やベーシックなタスク管理ツールなど，コラボレーションのための機能も提供する．

GitHub の最大の目玉機能が“フォーク(forking)”だ．フォークとは，食器のフォークの先端のように，一つのプロジェクトが複数に分派していくこと．それを GitHub では，誰かのリポジトリをほかの人がコピーすることによって行う．オリジナルに対するライト(write, 書き込み)アクセスがなくても，それを自分のところで改変できる．自分が行った変更をオリジナルに反映したかったら，オリジナルのオーナーに“プルリクエスト(pull request)”と呼ばれる通知を送る．そしてオーナーは，ボタンをクリックするだけで，その人のリポ

ジトリに対して行われた変更を自分のリポジトリにも取り入れることができる．人のコードを自分の（メインの）コードに導入することを，デベロッパ界隈の用語でマージする(merge, 併合)と言う．マージは複数のデベロッパが関わるプロジェクトにおいて最重要な工程であり，GitHub ではそれが安全確実迅速にできる．

これら三つの機能フォークとプルリクエストとマージがあるために，GitHub はきわめて強力なサービスなのだ． GitHub 以前には，オープンソースのプロジェクトに寄与貢献しようと思ったら，まずそのプロジェクトのソースコードを手作業でダウンロードし，変更をローカルに行い，そして“パッチ(patch, つぎあて)”と呼ばれる変更内容の一覧をプロジェクトのメンテナにメールする．メンテナはそのパッチを評価し，OK となったらそれらの変更をマージする．パッチは，まったく未知の人から突然送られてくることも多い．

しかし，GitHub ではネットワーク効果が絶大な威力を発揮する，と Pollack は説明を続ける．プルリクエストを送ると，プロジェクトのメンテナは自身のプロフィールを見る．そこには，自身が GitHub 上で行った寄与貢献(contributions, コントリビューション)がすべて書かれている．自身のパッチが受け入れられたら，自身はオリジナルのサイト上で信任を取得し，そのことがプロフィールにも載る．メンテナがプロフィールを見るときそれは，自身の評判を知るための履歴書のようなものだ．GitHub 上にたくさんの人間とたくさんのプロジェクトがあればあるほど，プロジェクトのメンテナが優れた寄与貢献者(contributors, コントリビュータ)をより見つけやすくなる．これぞまさに，ネットワーク効果の典型だ．パッチは，特定のメンテナの手に渡るだけでなく，公開の場で議論することもできる．

GitHub のインターフェイスを使わないメンテナにとっても，GitHub の存在はコントリビューションの管理を容易にしてくれる．オープンソースのサーバサイド JavaScript プラットホーム，Node.js のメンテナ Isaac Schlueter によると，最終的にはとりあえずパッチをダウンロードしてコマンドラインでマージしている．GitHub のマージボタンは使わないため，敷居を低くして誰もが気軽に参加できるため，オープンソースの開発がより民主化され，若いプロジェクトの成長を助けることが期待される．

GitHub には，一般に公開されるオープンソースのリポジトリ集という顔のほかに，企業にプライベートのリポジトリを売ったり，そのソフトウェアの企業向けのオンプレミスのインスタンスを売るビジネスとしての顔がある．これらのソリューションにはもちろん，オープンな GitHub のネットワーク効果というアドバンテージはないが, コラボレーションの機能はそっくりそのままある．ただしビジネスとしてのそれには，敵もいる．

Atlassian が 2010 年に買収した BitBucket も，競合製品の一つだ（VCS ないし SCMS(source code management system)として Git だけでなく同じく好評な Mercurial も使える）．今年の初めに Atlassian が立ち上げた Stash は，プライベートでオンプレミスな Git リポジトリを BitBucket 的/GitHub 的なコラボレーション環境で設営利用できる製品だ. Atlassian はこのほか，バグトラッカーの Jira や wiki ソフトの Confluence など，デベロ

ッパのためのコラボレーションツールも売っている．Atlassian は 2010 年に Accel Partners から 6000 万ドルを調達している．この 6000 万ドルという数字を見れば，GitHub の 1 億ドルの意味もおよそ分かるだろう．同社が未来に向けて何を志向しているのかも分かる．

しかし，おそらく特に重要なのは，GitHub がコードに関して図書館のようになっていることだ．Git を使うときめ細かい粒度の小さい変更の記録ができるので，まったくの初心者でもあるいはエキスパートでも，世界のもっとも偉大なデベロッパたちの足跡を踏みしめることができ，彼らがプログラミングの難題をどうやって解いたかを知ることができる．今回の投資の結果がどう出ようとも，それこそが，GitHub が今後の世界の歴史に残す，驚異的な遺産である．

4.2 GitHub の機能について

4.2.1 Git について

「Git」とは，プログラムソースなどの変更履歴を管理する分散型のバージョン管理システムのことである．これは，もともと Linux の開発チームが使用して，徐々に世界中の技術者に広まっていった．

「Git」の最大の特徴として挙げられるのは，分散型の名の通り，ローカル環境（自分のパソコンなど）に，全ての変更履歴を含む完全なリポジトリの複製が作成されることである．これは，各ローカル環境がリポジトリのサーバーとなれるということである．分散型ではない，これまでのバージョン管理システムでは，サーバー上にある 1 つのリポジトリを，利用者が共同で使っていた．このため，利用者が増えると変更内容が衝突したり，整合性を維持するのが大変である．

「Git」では，ローカル環境にもコードの変更履歴を保持（コミット）することができるので，リモートのサーバーに常に接続する必要がなく，ネットワークに接続しなくても作業を行うことが出来るのだ．

こういったメリットが受けて，近年のバージョン管理システムの主流になっている．

4.2.2 リポジトリについて

リポジトリとは，ファイルやディレクトリの状態を記録する場所である．保存された状態は，内容の変更履歴として格納されている．変更履歴を管理したいディレクトリをリポジトリの管理下に置くことのより，そのディレクトリ内のファイルやディレクトリの変更履歴を記録することが出来る．

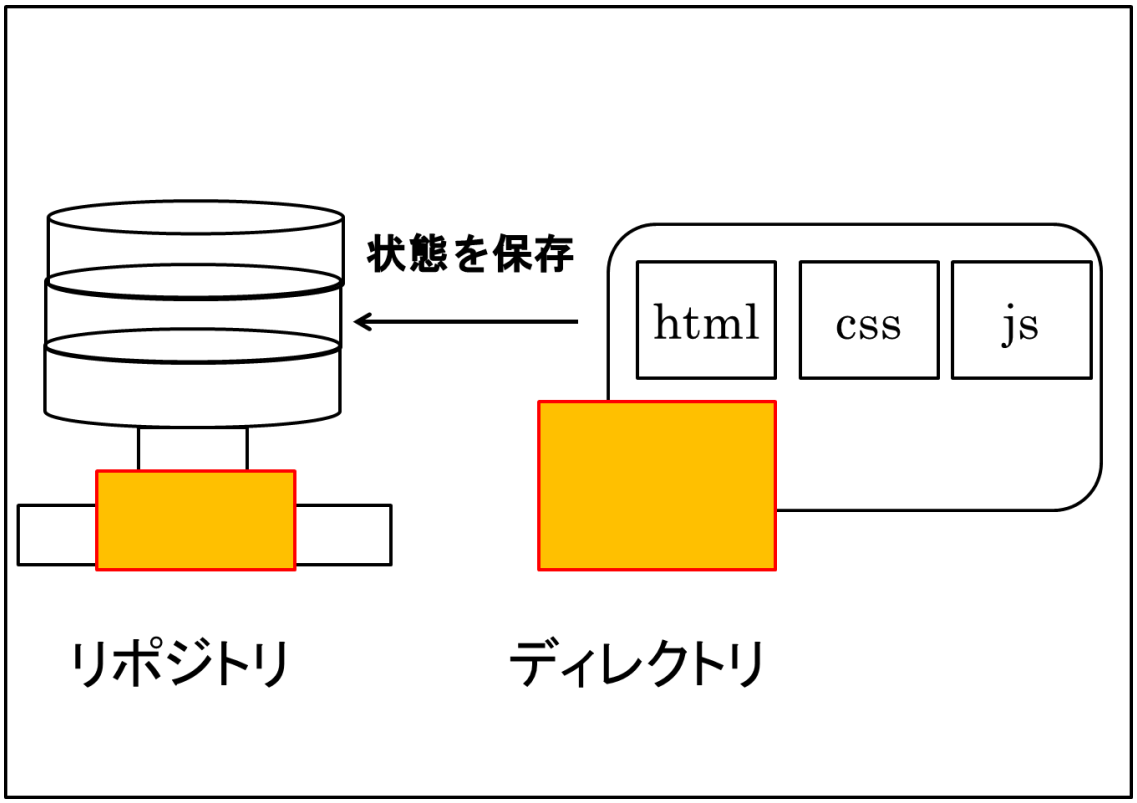

図 ディレクトリ内のファイルやディレクトリの変更履歴を記録する例

まず，Git のリポジトリは，リモートリポジトリとローカルリポジトリの 2 種類に分けられる．

- リモートリポジトリ：専用のサーバに配置して複数人で共有するためのリポジトリである
- ローカルリポジトリ：ユーザ一人ひとりが利用するために，自分の手元のマシン上で配置するリポジトリである．

リポジトリをリモートとローカルの 2 種類に分けるとこにより，普段の作業はローカルリポジトリを使い，全て手元のマシン上で行うことが出来る．

自分のローカルリポジトリで作業した内容を公開したい時は，リモートリポジトリにアップロードして公開する．また，リモートリポジトリを通して他の人の作業内容を取得することが出来る．

4.2.3 コミットについて

ファイルやディレクトリの追加・変更をリポジトリに記録するにはコミットという操作を行う．

コミットを実行するとリポジトリ内では，前回コミットした時の状態からどこを修正したかがわかるようにリビジョンと呼ばれるものが作成される．

このコミットは，次の図のように時系列順に繋がった状態でリポジトリに格納されている．このコミットを最新の物から辿ることで，過去の変更履歴やその内容を知ることが出来るようになっている．

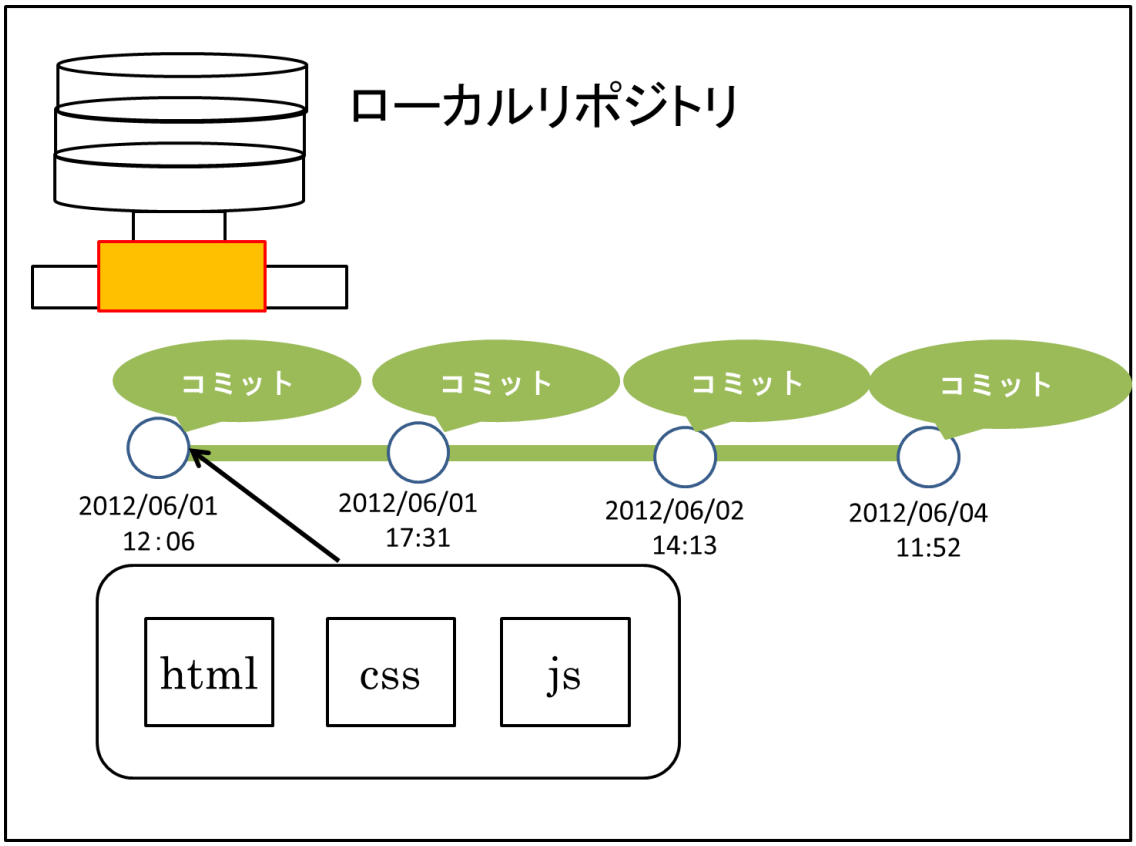

図 コミットがリポジトリに格納せれている状態

これらのコミットには，コミットの情報から計算された重複のない英数字 40 桁の名前が付けられている．この名前を指定することで，リポジトリの中からコミットを指定することが出来る．

バグ修正や機能追加などの異なる意味を持つ変更は，できるだけ分けてコミットするようにする．後から履歴を見て特定の変更内容を探しやすくなるためである．

コミットの実行時には，コミットメッセージの入力を求められる．コミットメッセージは必須となっているため，入力せずに実行するとコミットできない．

コメントは，他の人がコミットの変更内容を調べる場合や自分で後から履歴を見直す際に大切な情報になる．そのため，変更内容のわかりやすいコメントを書くようにする方が良いだろう．

4.2.4 リポジトリの共有について

- Push：リモートリポジトリで自分の手元のローカルリポジトリの変更履歴を共有するには，ローカルリポジトリ内で変更履歴をアップロードする必要がある．そのために，Git では Push という操作を行う．Push を実行すると，リモートリポジトリに自分の変更履歴がアップロードされて，リモートリポジトリ内の変更履歴がローカルリポジトリの変更履歴と同じ状態になる．

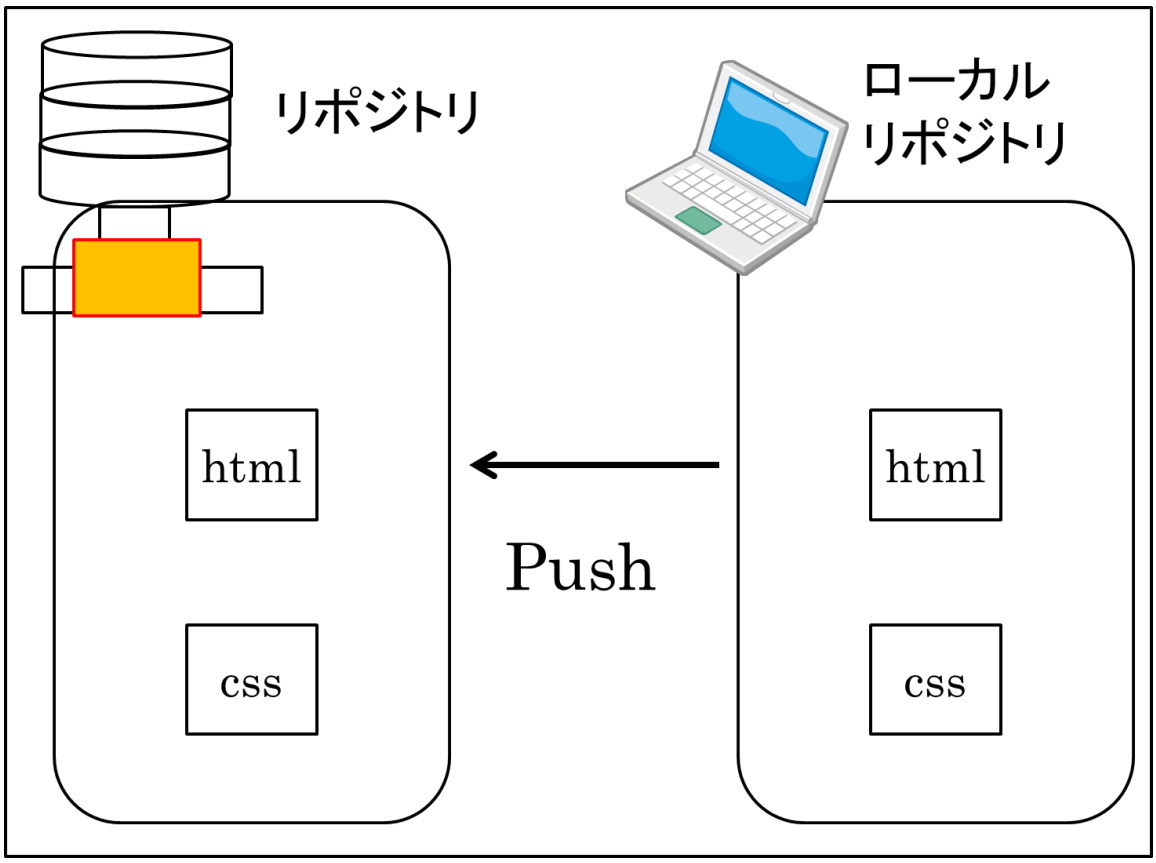

図 ローカルリポジトリから Push を実行した例

- Clone：リポジトリを複製するには，Clone という操作を行う．Clone を実行すると，リモートリポジトリの内容をそのままダウンロードしてきて，別のマシンにローカルリポジトリとして作成することができる．また，クローンしたローカルリポジトリは変更履歴も複製されているので，元々のリポジトリと全く同じように履歴の参照やコミットをすることが出来る．
- Pull：リモートリポジトリからローカルリポジトリを更新するには Pull という操作を行う．Pull を実行すると，リモートリポジトリから最新の変更履歴をダウンロードしてきて，自分のローカルリポジトリにその内容を取り込むことが出来る．

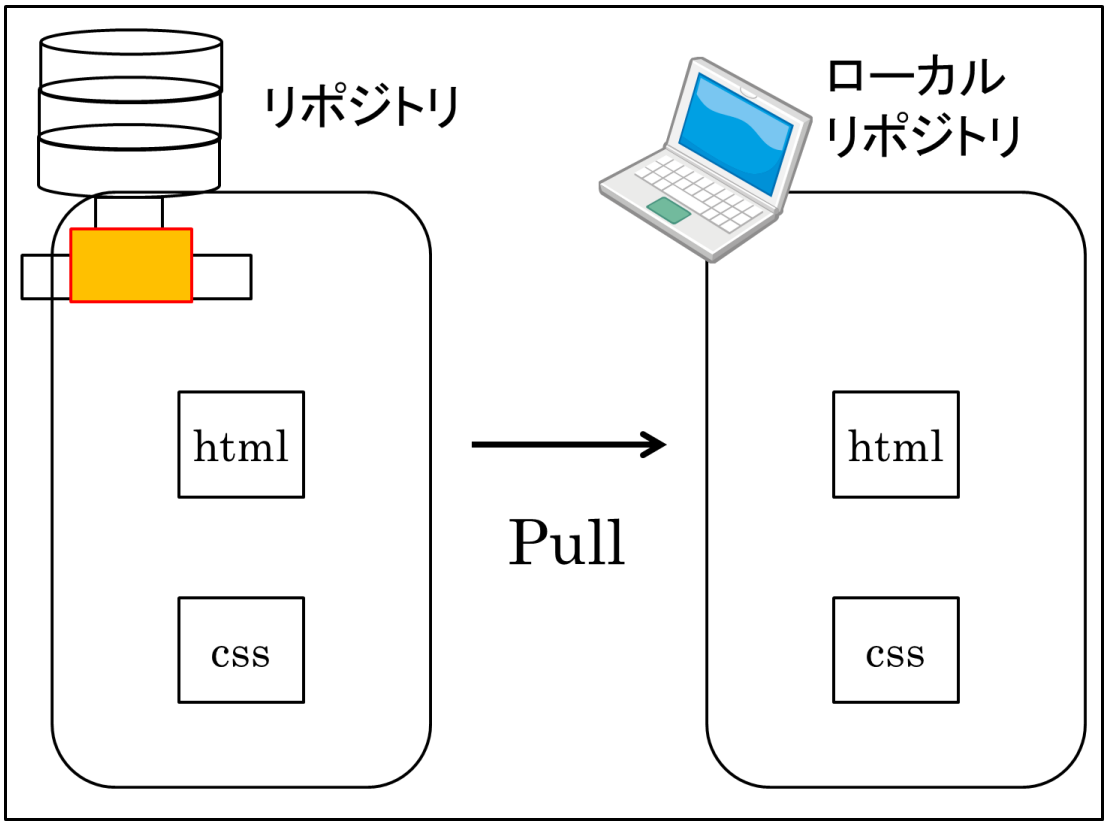

図 ローカルリポジトリに Pull を実行した例

4.2.5 変更履歴のマージについて

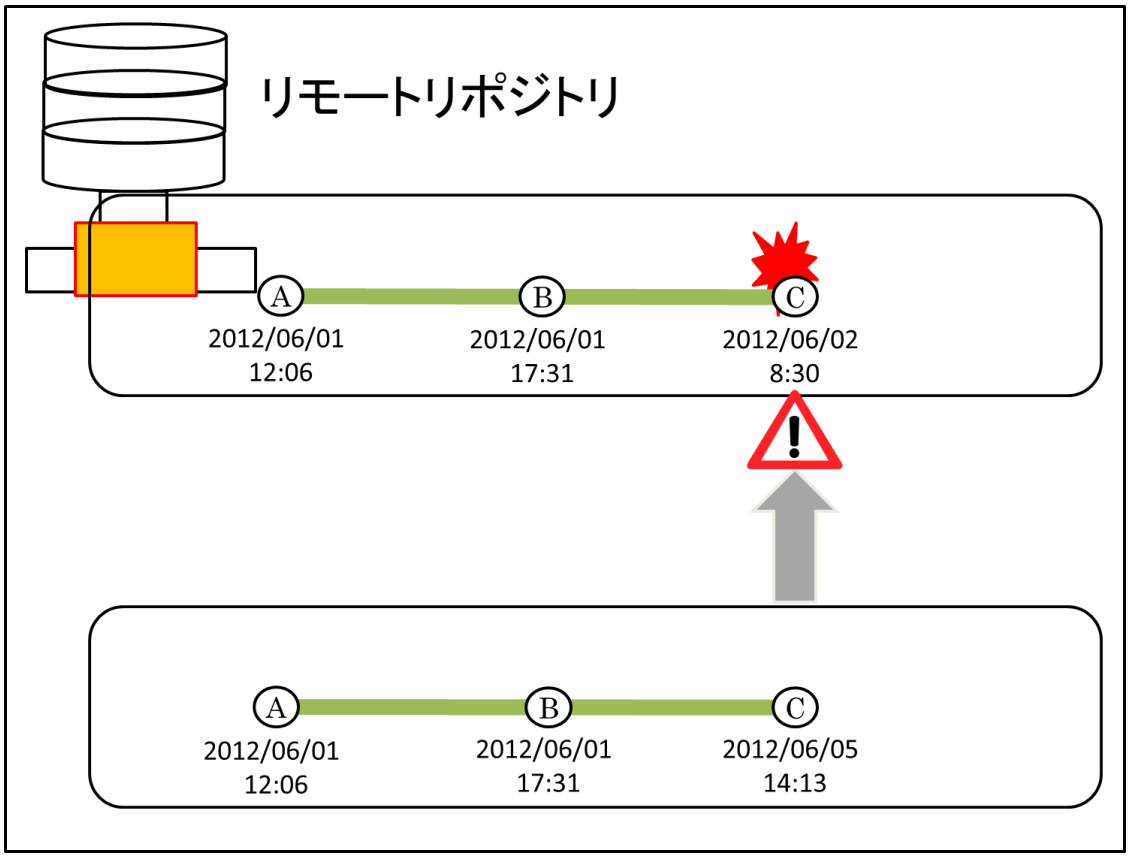

図 Push が拒否された例

最後に Pull を実行してから次の Push をするまでの間，他の人が Push をしてリモートリポジトリを更新してしまうと自分の Push が拒否されてしまう．

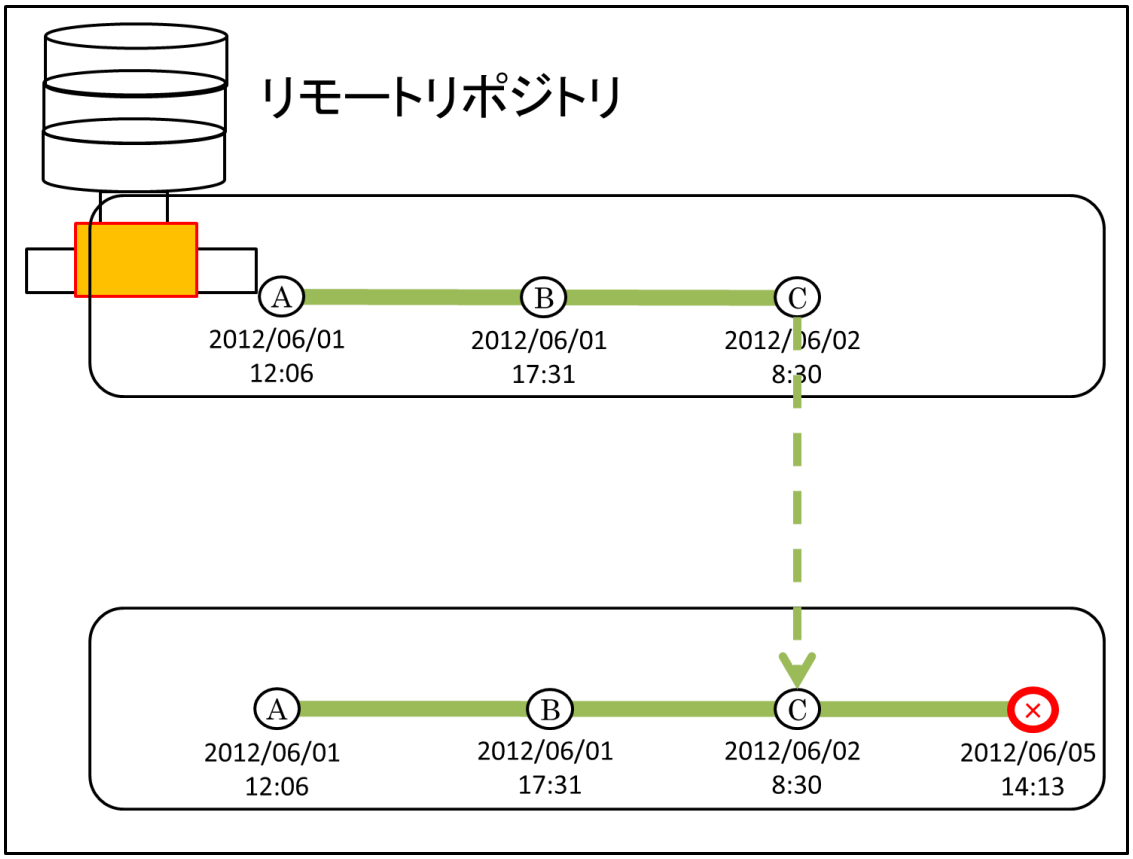

図 マージを行わないと Push が拒否される例

この場合は，マージという作業を行って他の履歴での変更を取り込むまで自分の Push は拒否されてしまう．これは，マージを行わないまま履歴を上書きしてしまうと，他の人が Push した（図中のコミット C）が失われてしまう．

4.2.6 ワークツリーとインデックスについて

Git で，Git の管理下に置かれた実際に作業しているディレクトリのことをワークツリーと呼ぶ．そして，Git ではリポジトリとワークツリーの間にはインデックスというものが存在している．インデックスとは，リポジトリにコミットする準備をするための場所である．

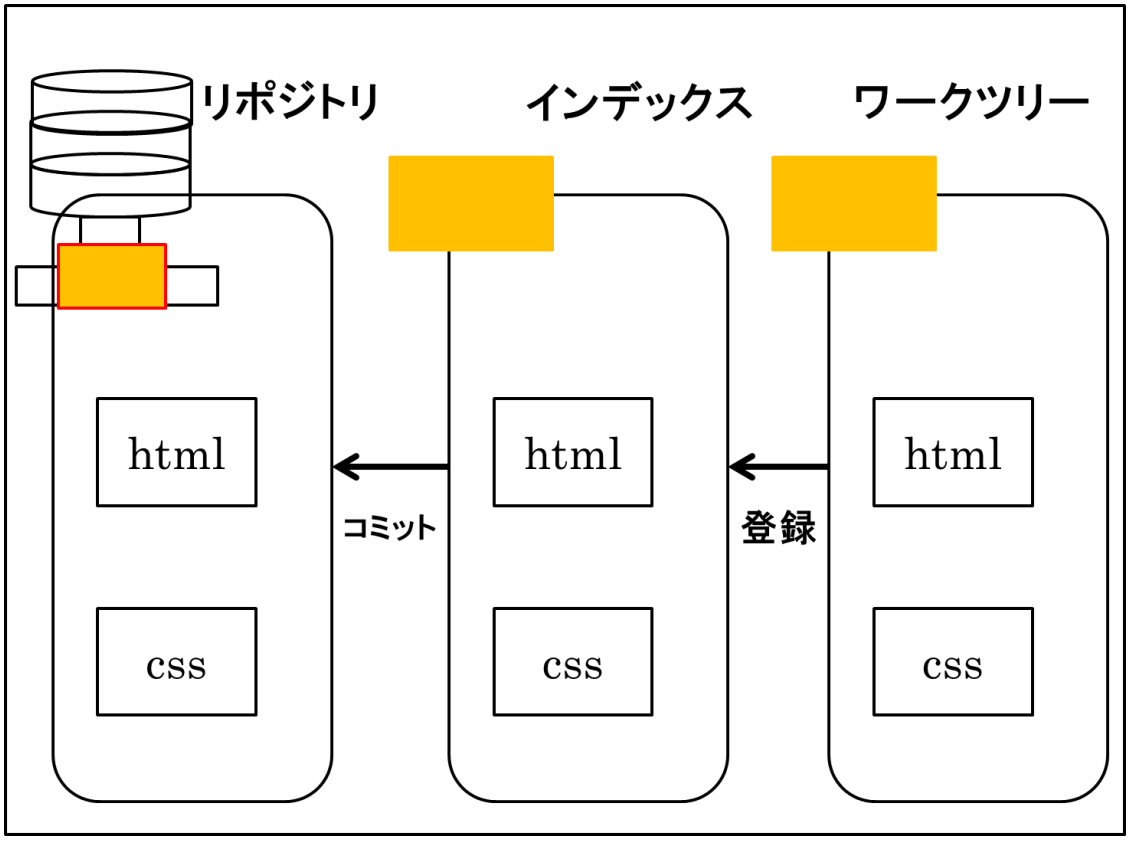

図 インデックスに登録してコミットする流れ

Git では，コミットを実行した時にワークツリーから直接リポジトリ内に状態を記録するのではなく，その間に設けられているインデックスの設定された状態を記録するようになっている．

そのため，コミットでファイルの状態を記録するためには，まずインデックスにファイルを登録する必要がある．

このようにインデックスを間に挟むことで，ワークツリー内の必要ないファイルを含めずにコミットを行ったり，ファイルの一部の変更だけをインデックスに登録してコミットすることが出来る．

4.3 GitHub の API について

GitHub の API には以下のイベントがある.

イベント	内容
CommitCommentEvent	コメントを行ったイベント.
PushEvent	プッシュを行ったイベント.
DeleteEvent	デリートを行ったイベント.
IssuesEvent	IssuesEvent を行ったイベント.
PullRequestEvent	プルリクエストにコメントを行ったイベント.
WatchEvent	スターを付けるイベント.