# 7.1 バージョン管理を行う

**こんなケース**

アプリをチームで開発することになったのですが、どのようにソースコードをメンバ間で共有するべきか悩んでいます。
また、共有するとメンバ間で内容の不整合が起こったり、バックアップを取ったりするのが大変そうです。
師範、アドバイスを頂戴（ちょうだい）できませんか！？

**心得るべし**

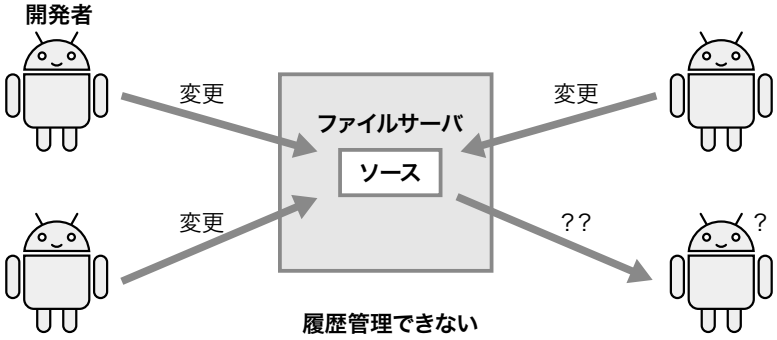
弟子よ、お前の悩みは全て「バージョン管理システム」で解決する。
じゃが、その学習コストは決して小さいものではないぞ？

1. 多人数での開発では、バージョン管理システムを導入すべし。
2. 多くの種類や製品の中から自分に必要なバージョン管理システムを選定すべし。
3. バージョン管理システムを使いこなせるよう学習すべし。

1人で開発している時とは違って、多人数でアプリを開発する際には、いろいろと考慮すべきことが多くなります。その1つとして、ソースコードの共有が挙げられます。皆さんはどのような方法でソースコードを仲間と共有しますか？ すごく原始的な方法ですと、メールでやり取りしたり、外部メディアを手渡すことによる共有だと思います。これよりも少し進んだ方法は、ファイルサーバでの共有が考えられます。

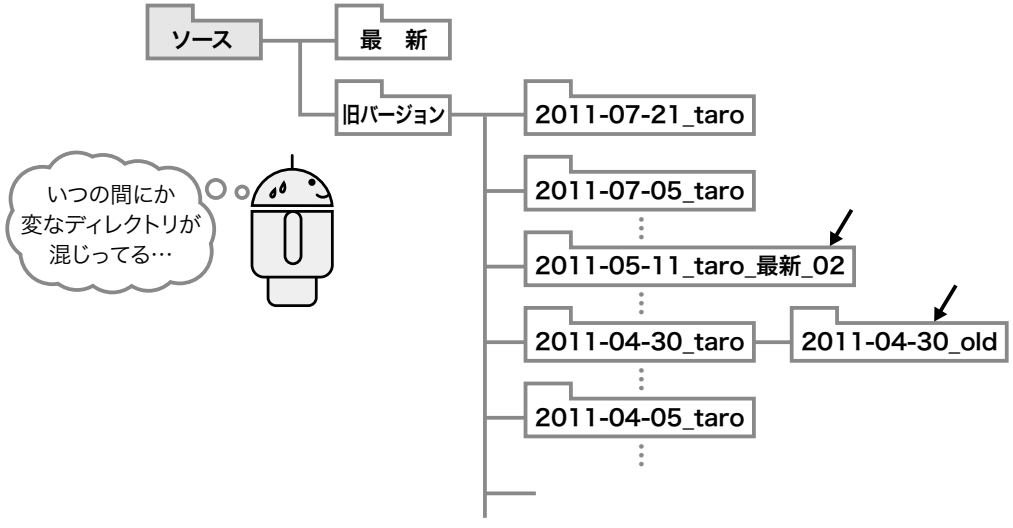
ソースコードの共有という視点のみで考えれば、ファイルサーバは悪い選択ではなさそうに思えます。しかし、アプリ開発においてソースコードは時とともに変化していくのが普通です。しかも多人数でコードを編集するのですから、その変化は複雑で自力で追跡するのは困難です（**図7.1.1**）。「誰が・いつ・どこを・どのように」編集したか知りたくてもファイルサーバだけでは、そういった機能は普通はありません。

図7.1.1　ファイルサーバでソースコードを共有する



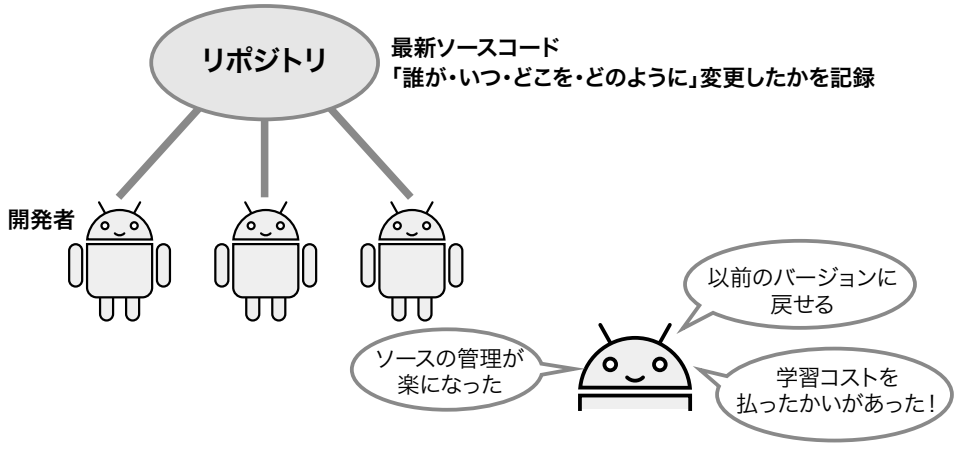

そこでよくある工夫の1つが、命名規則に従ったディレクトリによる分け方です。これは日付や連番などをディレクトリの名前にして、そこにソースファイルを格納する方式です。例えば「2011-07-21_taro」というディレクトリは、2011年7月21日時点で最新のソースファイルがtaroという人物に格納されたということを意味するわけです。他には接頭辞や接尾辞を付けて区別する方法もありますが、いずれにしても時が経つにつれ乱雑になりやすく、あまり良い方法とはいえません(**図7.1.2**)。

図7.1.2　命名規則に従ったディレクトリによる分け方




そこで登場するのがバージョン管理システムです(**図7.1.3**)。これを使用すると、リポジトリと呼ばれる領域を仲間と共有することができ、そしてさらにその変更履歴を全て管理してくれます。つまり、上述の「誰が・いつ・どこを・どのように」編集したかという情報が常に明確な形で記録されていくのです。これは多人数での開発では絶大な効果を発揮します。また、1人で開発する場合でもバージョン管理システムを使用して変更履歴を記録することによる利点を多く享受できます(→5.4節)。まだ導入していない場合は、ぜひバージョン管理システムの導入を検討してみてください。

図7.1.3　バージョン管理システム




一口にバージョン管理システムといっても、実際の製品はいくつも存在し、形態も数種類に分かれます。ここでは代表的なバージョン管理システムについて簡単に紹介します。自分のニーズや好み、ネットや周りの人の評判を聞いて選ぶとよいでしょう。

## ◆集中管理型

リポジトリが1つだけのタイプです。リポジトリはサーバに置かれ、開発者たちはそれに対してチェックアウトしたりコミットしたりします[1]。リポジトリが1つだけなので考え方も構造もシンプルですが、サーバと接続できない環境ではリポジトリに対する操作ができないこと、及び個人的な実験のためのブランチ／タグが作成しづらいことが短所です。

### ●CVS（Concurrent Version System）

かつて多くのオープンソースプロジェクトが利用し、デファクトスタンダードを築いた、集中型のバージョン管理システムです。

歴史は古く、1986年にDick Grune氏によって開発されたシェルスクリプトに端を発します。当時、単一ファイルに対するバージョン管理手段として浸透していたRCSの流れを汲むもので、ディレクトリを介して複数ファイルの管理を行えるようにRCSを拡張したものといってもよいでしょう。

とても軽量で、sshアカウントさえあればすぐにでも始められる反面、ファイルごとにリビジョンが付く、ディレクトリがバージョン管理されない、ファイルの移動やリネームが起きると歴史が分断され

1　バージョン管理システムでは、リポジトリから自身の環境にファイルをコピーすることを「チェックアウト」、編集したものをリポジトリに反映することを「コミット」と呼びます。

てしまう、バイナリファイルの取り扱いが非効率およびコミットがatomicでない（i.e. コミット中に中断されるとリポジトリの内容が中途半端に書き換わったままになってしまう）という欠点があり、特にMozillaやFreeBSDといった大規模プロジェクトにおいてはそれらが大きな問題となることがしばしばありました。

また、1980年代といういわば黎明期からあったシステムなので、メンテナンスしづらい構造になっているとされ、今では更新もあまり行われていません。CVSNTなどの亜種があります。現在では、SubversionやGitなどへ移行している利用者が多くなっています。

### ●Subversion

非常に広く使われている集中型のバージョン管理システムです。CVSと似た操作性を備えています。2000年にKarl Fogel氏をはじめとする一部のCVS開発者達の手によって、上記のようなCVSが抱えていた欠点を修復することを目的として始まりました。

Subversionでは、コミットはatomicであり、ファイルごとではなくコミットごとにリビジョンが振られ、ディレクトリもバージョン管理され、ファイルの移動およびリネームが起きても歴史が分断されず、またバイナリファイルも差分のみの保管となるためにディスク利用効率も良くなっています。サーバ側も、Apache 2.xのWebDAVサーバモジュールとして機能させることにより、これまでのsshアカウントだけでなくRADIUS、LDAPなどApacheが使用できる全てのユーザ認証方式をサポートした上に、「Apache 2.xさえあればSubversionサーバも使える」という特に企業環境にとっては理想的ともいえる状況を作り出しました。

しかし、CVSと比べ若干クライアント側コンピュータの負荷が高いことが欠点になります。

## ◆分散管理型

リポジトリが複数あるタイプです。サーバには中央リポジトリが、クライアントにはローカルリポジトリが置かれます。開発者はローカルリポジトリに対してコミットをするので、オフライン状態でもコミットができます。また、分散型は多人数の開発者のコミットをマージすることに長けています。加えて、中央サーバがないので、開発者個々人が好きなスタイルで開発を進めることができます。

特にどのシステムを選んでよいのかわからないような場合は、本書ではこちらのシステムを選択することをおすすめします。コミットをいちいちサーバに送らない分、実験的なコードでも特に気兼ねなく何度でも書き直すことができる上に、他の人が行った変更を取り込む際にも比較的スムーズに行うことができるためです。

### ●Git

有名なLinuxカーネルの開発者であるLinus Torvalds氏によって開発された分散型バージョン管理システムです。GitHub [1]という有名なホスティングサービスの登場もあり、近年非常に注目されているバージョン管理システムの1つです。