

WebDAVバージョンング実装 Subversion

HTTP拡張プロトコルで
バージョン管理をする

早川 仁 HAYAKAWA Hitoshi

はじめに

約半年前、筆者がWebDAVについて執筆や講演を行ったときには日本語での記事はほとんど皆無に近く、海外のサイトやメーリングリストに参加する以外にはWebDAVに関する情報を入手することはできなかったのですが、最近はおちこちで聞くようになり、また雑誌やWebサイトにも記事が増えてきました。WebDAVをサポートするサーバやクライアントも段々と数が増えてきており、2002年は日本でもWebDAVがブレイクする年になるのではないのでしょうか。

WebDAVについて詳しく書いても他のものと内容が繰り返しになってしまうため、この記事ではWebDAVに関しては読み進めるために必要なことだけに絞り、WebDAVのバージョンング実装であるSubversionの解説を主とします。

WebDAV

WebDAVは“Web-based Distributed Authoring and Versioning”を略したもので、時にはDAVと呼ばれます。RFC2518で定義されており、特定のアプリケーションの名前ではなくWebDAVというプロトコルの名前です。このことは、HTTPやFTPというプロトコルは存在してもHTTPというアプリケーションは存在しない、ということの思い浮かべると理解できるのではないのでしょうか。下記はWebDAVをサポートするア

プリケーションの一例です。WebDAVをサポートするアプリケーションの最新のリストはIETF WEBDAV Working Groupのサイト(参考URL参照)の下部にある一覧を参照してください。

WebDAV サーバ

- Apache (mod_dav)
- Microsoft IIS (Internet Information Services/Server)
- Zope

WebDAV クライアント

- Windows 2000/XP やOffice 2000のWeb フォルダ機能
- cadaver
- davfs

DAVのVは「バージョンング」を意味しますが、実はRFC2518ではバージョンングに関する仕様は盛り込まれていません。バージョンングに関してはDelta-Vという別のプロトコルが定義されており、現在INTERNET-DRAFT (draft-ietf-deltav-versioning-20)の状態です。

WebDAV プロトコルとは

WebDAVは簡単に言うと「HTTPにメソッドを追加して、コンテンツの読み書きを可能にしたプロトコル」です。表1はHTTP 1.1の、表2はWebDAVのメソッド一覧です。WebDAVにはHTTP 1.1の全メソッドも含まれます。

WebDAV バージョニング実装 Subversion

通常のHTTPアクセスはWebサーバ（HTTPサーバ）にWebブラウザ（HTTPクライアント）でアクセスし、コンテンツを読み込みます。WebDAVアクセスはWebDAVサーバ（多くの場合WebDAVサーバ機能を持つWebサーバ）をWebDAVクライアントでアクセスし、コンテンツ（WebDAVの用語ではリソースと呼びます）を読み書きします。図1は「HTTPで

コンテンツを取得」したときの通信を、図2は「WebDAVでリソースの状態を取得」したときの通信を抜粋したものです。なおコンテンツ（リソース）の読み込み時は、WebDAVもHTTPと同様にGETメソッドでアクセスを行います。より正確には「WebDAVはリソースの読み込み時にはHTTPでアクセス」しています。

表1 HTTP 1.1 (RFC2616) のメソッド

メソッド	動作
OPTIONS	使用可能なメソッド一覧の取得
GET	リソースの取得
HEAD	ヘッダ情報の取得
POST	エンティティの送信
TRACE	リクエストがそのまま戻ってくる（ループバックする）
CONNECT	予約語（プロキシなどでの利用を想定）
DELETE	リソースの削除
PUT	リソースの作成・更新

図1 HTTP による通信

```

リクエスト
GET / HTTP/1.1
Accept: */*
Accept-Language: ja
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0; NetCaptor 6.5.0B4)
Host: 192.168.10.5
Connection:Keep-Alive

レスポンス
HTTP/1.1 200 OK
Date: Tue, 1 Jan 2002 10:00:00 GMT
Server: Apache/1.3.14 (Unix) (Red-Hat/Linux) mod_ssl/2.7.1
OpenSSL/0.9.5a DAV/1.0.2 PHP/4.0.1pl2 mod_perl/1.24
Last-Modified: Wed, 18 Oct 2000 22:31:33 GMT
Accept-Ranges: bytes
Content-Length: 2890
Content-Type: text/html

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<HTML>
<HEAD>
<TITLE>Test Page for the Apache Web Server on Red Hat Linux</TITLE>
</HEAD>
<!-- Background white, links blue (unvisited), navy (visited), red (active) -->
(以下略)

```

表2 WebDAV (RFC2518) のメソッド

メソッド	動作
HTTP/1.1の全メソッド	表1参照
COPY	リソースのコピー
MOVE	リソースの移動
MKCOL	コレクションの作成
PROPFIND	プロパティの取得
PROPPATCH	プロパティの追加・変更・削除
LOCK	リソースのロック
UNLOCK	リソースのロックの解除

図2 WebDAV による通信

```

リクエスト
PROPFIND /DAV/Readme.txt HTTP/1.1
Accept-Language: ja,en-us;q=0.2
Content-Type: text/xml
Content-Length: 380
Depth: 1
User-Agent: Microsoft Data Access Internet Publishing Provider DAV 1.1
Host: 192.168.10.5

レスポンス
HTTP/1.1 207 Multi-Status
Date: Tue, 1 Jan 2002 10:05:05 GMT
Server: Apache/1.3.14 (Unix) (Red-Hat/Linux) mod_ssl/2.7.1
OpenSSL/0.9.5a DAV/1.0.2 PHP/4.0.1pl2 mod_perl/1.24
MS-Author-Via: DAV
Keep-Alive: timeout=15, max=99
Content-Type: text/xml; charset="utf-8"

<?xml version="1.0" encoding="utf-8"?>
<D:multistatus xmlns:D="DAV:">
<D:response xmlns:lp0="DAV:" xmlns:lp1="http://apache.org/dav/props/">
  <D:href>/DAV/Readme.txt</D:href>
  <D:propstat><D:prop>
    <lp0:creationdate>2001-05-02T06:35:45Z</lp0:creationdate>
    <lp0:getcontentlength>3199</lp0:getcontentlength>
    <lp0:getlastmodified>Wed, 02 May 2001 06:35:45 GMT</lp0:getlastmodified>
(以下略)

```

HTTP 拡張プロトコルでバージョン管理をする

WebDAV の特徴

すでに述べた通り、WebDAVはHTTPを拡張（メソッドを追加）したものです。そのためWebDAVはHTTPの利点をそのまま引き継いでいます。アクセスに必要なポート番号、セキュアな通信を行うための方法はHTTPと同じです。

アクセスに必要なポート番号

読み書きどちらのアクセスであっても、80番ポート（SSLを使用するときは443番ポート）しか使用しません。そのためファイアウォールの後ろにある企業内LANの中など80番ポートしか空いていないような場合でも、ファイルの読み書きが可能となります。

HTTPと同じ方法のセキュア通信

HTTPでの通信をセキュアにするのと同様の方法で、WebDAVでの通信もセキュアにすることができます。一般的な方法としては以下があります。ただしSSLに関しては、WebDAVクライアントが対応している必要があります。WebフォルダなどはSSLに対応しています。

- ポートフォワーディング
- SSL/TLS
- IPsec
- PPTP/L2F/L2TP

Subversion

SubversionはWebDAVのV、つまりバージョンニングのDelta-Vプロトコルを実装したアプリケーションですが、Delta-Vの仕様をすべて実装しているわけではなく、Delta-V非対応クライアントによって追加されたリソースを自動的にバージョン管理するオートバージョンニング機能など、実装されていない機能もあります。

バージョンニングとは

その名前が表すとおり、何かをバージョン

管理することをバージョンニングと言います。たとえば何か適当な文章の入ったテキストファイルfoo.txtを作り、さらにこれをbackup1.txtという名前でコピーしておきます。foo.txtに何行か加え、backup2.txtという名前でコピーします。そしてさらにfoo.txtに何行かを加えます。すると最終的に、backup1.txt、backup2.txt、foo.txtの3つのファイルができたわけですが、それぞれの関係は基本的には図3のように、外側のファイルは内側のファイル（の変更点）を含むこととなります。

このことは「backup1.txtおよびbackup2.txtはfoo.txtの古いバージョン」と言いかえることができます。今は手動コピーにより各バージョンを作成および管理したわけですが、このようにファイルの各バージョンを管理することをバージョンニング（あるいはバージョン管理）と呼びます。

バージョン管理システムの役割

各バージョンを管理するシステム（アプリケーション）はバージョン管理システムと呼ばれます。図4を見てください。ファイルを作成してバージョン管理システムに登録します。このときのファイルをバージョン1とします。次にこのファイルに手を加え、また登録を行います。ファイルはバージョン2となります。その後また手を加えて登録を行うと、バージョン3となります。この時点で、ファイルのバージョン1～3が登録されていることとなります。

バージョン管理システムを使うことにより、好きなときに過去のバージョン（この場合、バージョン1～3）を取り出すことが可能となります。たとえば編集・修正したファイルを前のバージョンに戻したくなった場合、いつでも過去のバージョンを取り出すことが

図3 ファイル内容の包含関係

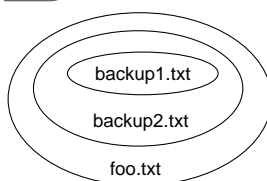
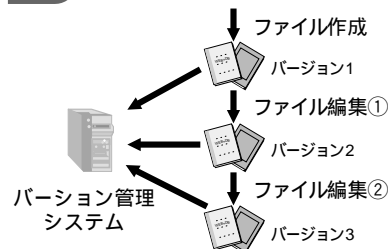


図4 ファイルのバージョンニング



WebDAV バージョニング実装 Subversion

きます。また、手元にあるファイルや登録してある過去の各バージョンとの差分を表示し、何のどこを修正したのかを表示することも可能です。特にプログラムのソースを管理しているときには、この「過去のバージョンを取り出す」という機能が非常に便利です。

バージョン管理システムは難しい物ではありません。手作業（コピーなど）でやれることとの差異はほとんどありません。言うまでもなく、コピーや差分の表示は手作業でも可能です。要は面倒なことを自動的にやってくれるかどうかというだけで、難しく考える必要はありません。

Subversion のアーキテクチャ

Subversion は Apache と mod_dav (WebDAV モジュール) に依存しているため、実行には必ず両者が必要となります (図5)。サーバに Apache が理解できないメソッドが送られてきた場合、そのメソッドは mod_dav に送られ、mod_dav が理解できないメソッドが送られてきた場合には mod_dav_SVN (Subversion モジュール) に送られます。

用語の説明

Subversion では各ファイルなどを管理しているサーバ内の場所をリポジトリと呼んでいます。またバージョン番号のことをリビジョンあるいはリビジョン番号と言います。ファイルをリポジトリに登録することは CVS と同様に「インポート」あるいは「追加」と言います。またリポジトリに登録されているファイルなどを取り出して更新を行い、リポジトリにその更新を反映することを「コミット」あるいは「チェックイン」と言います。

Subversion におけるバージョンの管理

Subversion はファイル単位ではなくツリー単位でバージョン管理しています。CVS のようにファイル単位で管理している場合、ファイル間に依存関係がないときにはうまく働きますが、プログラムのソースファイルを管理しているときなどのように依存関係が存在していると、不都合が発生する可能性があります。その

ため CVS などでは依存関係がうまく解決できているときや、正常にコンパイルおよび動作するようになったタイミング時などに、タグを打ったりブランチを作成することで、明示的に「その時点」を指定しておく必要があります。一方、ツリー単位でバージョンングを行っている場合、基本的にはタグやブランチといったことに関する考慮は必要なくなります。

ここで Subversion でのリビジョンの内部管理について見ていきましょう。

① リビジョン1

まずファイル1を作成してサーバの hoge ディレクトリにインポートしてみます。このときのリポジトリは図6のようになります。リポジトリのリビジョン1にはファイル1だけが存在しています。ディレクトリエントリの中にはそのディレクトリに含まれるファイルの名前だけが格納されます。UNIX のファイルシステムと同様の構造ですが、サーバ上にこのようなファイル/ディレクトリが実際に作られるわけではなく、Berkeley DB 内に論理的に構成されるだけです。

② リビジョン2

次に (リポジトリに) ファイル2を追加します。すると図7のようにリビジョン2が作成されます。ファイル1は更新されていないため、中身はリビジョン1の物を共有することになります。

③ リビジョン3

今度はファイル1を更新、ファイル2をファイル2-2にリネーム、foo ディレクトリを追加してみます。リポジトリは図8のようになりました。

図5 アーキテクチャ

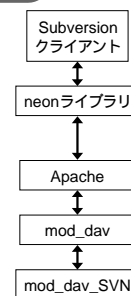


図6 リビジョン1の内部構造

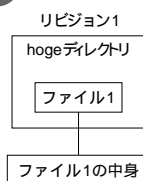
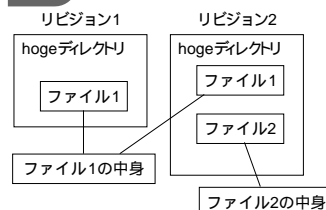


図7 リビジョン2の内部構造



HTTP 拡張プロトコルでバージョン管理をする

図8 リビジョン3の内部構造

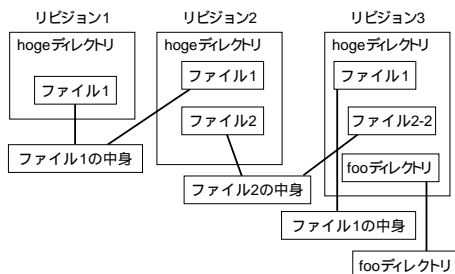


図9 リビジョン4の内部構造

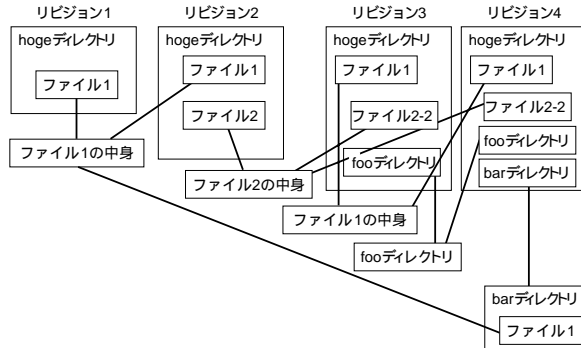


表3 Subversion が利用するメソッド

メソッド	動作
OPTIONS	表1参照
GET	"
DELETE	"
COPY	表2参照
PROPFIND	"
PROPPATCH	"
MKACTIVITY	アクティビティを作成
CHECKOUT	リソースの取り出し(チェックアウト)
MERGE	リソースのマージ
REPORT	リソースおよびプロパティの情報の取得

ファイル単位ではなくツリー構造でリビジョンを管理しているため、ファイル(もちろんディレクトリでも)をリネームしたとしてもバージョン管理が容易です。これがSubversionにタグが存在しない(必要としない)理由です。ファイル単位で管理されている場合、ある時点でタグを付けないと過去のバージョンを取り出すときに、各ファイルのどのリビジョンを取り出せばいいのかわからなくなってしまう危険性がありますが、Subversionの場合はタグの代わりにリビジョン番号を覚えておくだけです(コミット時に記録するログに更新内容をちゃんと書いておくと、管理しやすいです)。

ブランチに関しては、「コピー」という少々違う概念が導入されました。図7や図8を見てわかるように、中身が変更されていない場合はリビジョンが上がってもリンク先が変更されるだけで、同じ中身のものを複数持つわけではありません。

既存のツリーを別のサブディレクトリに「コピー」し、従来のブランチで行うような作業はこの下で行うことで、リソースを無駄にせず、また理解しやすい形

式でブランチのような分散作業が可能となります(CVSのブランチは混乱したりしませんか?)。

④リビジョン4

図9はリビジョン1のツリーを(barディレクトリに)コピーすることでブランチングを行い、リビジョン4が作成されたときの図です。言うまでもありませんが、OSのコピーコマンドでファイルをコピーしてリポジトリに追加した場合は、同一内容のファイルが単に複製されるだけです。

Subversion が利用するメソッド

表3はSubversionが機能するために必要となるメソッドです。OPTIONSからPROPPATCHまでは、ApacheあるいはWebDAVが提供する機能(メソッド)をそのまま利用しています。そのためWebDAVクライアントはWebブラウザの上位互換品で、SubversionクライアントはWebDAVクライアントの上位互換品と考えると理解しやすいかもしれません。

CVSとSubversionの差異

バージョン管理システムとして最も利用されているアプリケーションはCVSです。数多くの(特にオープンソース系の)プロジェクトでソースコードやドキュメントなどの管理に使用されています。Subversionは後発のプロジェクトであり、CVSを非常に意識しています。CVSのほとんどの機能を持ち、利用者にとって不都合のある機能は拡張されています(たとえばバイナリファイルの扱いなど)。ここではCVSとSubversionの差異を説明します。

WebDAV バージョニング実装 Subversion

管理の単位

CVSではファイル単位でバージョンングしますが、Subversionではツリー単位で管理しています。そのためCVSでは「foo.txtのリビジョン5」とは言えませんが、Subversionの場合「リビジョン5(のツリー)のfoo.txt」としか言えません。

サーバアクセス時の帯域使用量

リポジトリから取り出したファイルを、ローカルで修正した後に修正前の中身に戻したり(revert)ファイルの変更点(diff)を表示したりするときに、サーバにアクセスしません。これはリポジトリから取り出したファイルを別のディレクトリにも持っているから可能で、多少ディスク容量は無駄になってしまうかもしれませんが、昨今のディスク容量の増加とネットワーク帯域の増加分を考えると帯域の方が明らかに貴重かつ高価なため、このような設計になっています。またファイルの変更点をサーバに反映する際に、CVSのようにファイル全体ではなく、変更した差分だけを送るようになっています。

ディレクトリもバージョンングの対象

CVSでは、ディレクトリはバージョンングの対象とはなりません。そのためディレクトリを作成して追加("cvs add")した場合、CVSでは即座にサーバに反映されますが、Subversionではコミットを行うまで追加されません。また当然ながら、過去のバージョンを取り出すとディレクトリ(およびその配下のファイル/ディレクトリ)が削除/追加されることになります。

ファイルやディレクトリにプロパティを追加できる

WebDAVやDelta-Vでは、ファイルやディレクトリにプロパティという任意の名前と値の組を追加することができます。これにはシステムにより自動的に追加・更新されるLiveプロパティと、ユーザが任意に追加・更新できるDeadプロパティの2種類があります。Liveプロパティの例としては、ファイルの種類を表す"svn:mime-type"などがあります。

コンフリクト時の動作

あるファイルが存在していたとします(バージョン1とします)。他の人がそのファイルを更新すると最新版はバージョン2になりますが、また別の人がバージョン1のファイルを元に更新しサーバへ登録を行おうとすると、コンフリクトが発生します。コンフリクトが発生すること自体には問題がありません。もしコンフリクトが発生しないと、図10のように他人の作業を上書きしてしまう危険性があるからです。

すでに述べた通り、CVSはファイル単位でバージョンングを行っています。複数のファイルを同時にサーバに反映しようとした場合、ファイルによってはコンフリクトが発生して登録されないかもしれません。もちろんそのことに気付いてコンフリクトを修正すればよいのですが、発生したことに気付かなくて内容に矛盾が起こってしまう可能性もあります(あるファイルは新しい内容で、あるファイルは古い内容ということが起こりえる)。特にソースコードを管理している場合には問題となるでしょう。

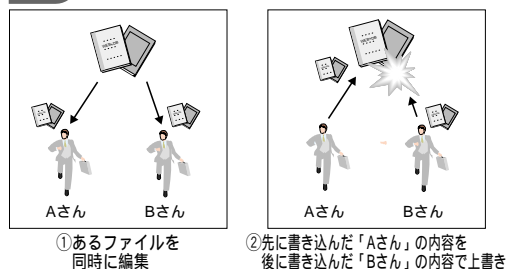
Subversionの場合はバージョンングはツリー単位のため、コンフリクトが発生したファイル/ディレクトリが1つでもあった場合、どのファイル/ディレクトリも登録されずに元の状態のままです。つまり反映の動作はアトミックです。

バイナリファイルの扱い

CVSではバイナリファイルを登録する場合、-kbオプションを付けてキーワードの展開と改行コードの変換を行わないようにする必要があります。もしこのオプションを忘れると、登録したファイルが破壊されてしまいます。

Subversionは独自のアルゴリズムでバイナリファイ

図10 ファイルを上書きしてしまう例



HTTP 拡張プロトコルでバージョン管理をする

ルかどうかを自動判別します。ファイルの種類は “svn:mime-type” というプロパティで管理しており、これが “text/*” 以外の場合はバイナリファイルと判断しています。もし自動判別に失敗したときには手動でこのプロパティの削除や設定を行うことで、バイナリファイルと指定することができます。なおバイナリファイルがコンフリクトした場合、ローカルのファイル名の末尾に.orig が付加されてリネームされ、サーバ上のバイナリファイルがローカルにコピーされます。

ブランチとタグの扱い

CVS と異なり、Subversion にはブランチおよびタグは存在しません（必要ありません）。

Subversion の導入

Subversion は Apache および WebDAV を利用しているため、インストールの前に Apache および WebDAV をインストールしておく必要があります。また Subversion は Apache 2.0 の API を想定しているため、Apache 1.3.x ではコンパイルを行うことすらできません。また、原稿執筆時点での最新版であるリビジョン 718 (subversion-r718) はそのままでは undefined symbol になり、Subversion モジュールをロードすることができず Apache の起動が失敗してしまいます。そのためネットワーク経由で Subversion を利用できません。現時点でネットワークを経由して Subversion サーバとクライアントがアクセスを行うには、少し前のリビジョン 88 (subversion-M3) を使用する必要があります。クライアントに関してはリビジョン 718 でもサーバに問題なくアクセスできますが、log など一部

図 11 Apache のインストール

```
$ tar zxvf httpd-2.0_20011225051214.tar.gz
$ cd httpd-2.0/src/lib
$ tar zxvf ../../apr_20011225052334.tar.gz
$ tar zxvf ../../apr-util_20011225052352.tar.gz
$ cd ..
$ ./buildconf
$ ./configure --enable-dav --enable-so --
prefix=/usr/local/apache2 --with-mpm=prefork
$ make
$ su
# make install
# /usr/local/apache2/bin/apachectl start
```

のコマンドはリビジョン 88 では実装されていないためエラーになることがあります。そのため、クライアントのバージョンもサーバとあわせておいたほうが無難かと思います。以後、明記していないときは、リビジョン 718 に関する内容です。

Apache のインストール

まず次の場所から Apache 2.0 (httpd-2.0), apr-util, apr を入手します。ここでは原稿執筆時点の最新版である httpd-2.0_20011225051214.tar.gz, apr_20011225052334.tar.gz, apr-util_20011225052352.tar.gz を利用します。なお Subversion のリビジョン 88 を使う場合、このリビジョンは現在の Apache 2.0 ではコンパイルできないため 2001 年 9 月版を利用する必要があります。

Apache 関係ファイルの入手先

http://cvs.apache.org/snapshots/httpd-2.0/
 http://cvs.apache.org/snapshots/apr/
 http://cvs.apache.org/snapshots/apr-util/

Apache 関係ファイル 2001 年 9 月版の入手先

http://www.gihyo.co.jp/wdpress/source/vol7/httpd-2.0_20010915041212.tar.gz
 http://www.gihyo.co.jp/wdpress/source/vol7/apr_20010915041901.tar.gz
 http://www.gihyo.co.jp/wdpress/source/vol7/apr-util_20010915041912.tar.gz

Apache のインストールは非常に簡単です。図 11 のようにしてインストールを行ってください。\$ というプロンプトは適当な一般ユーザ、# は root ユーザでの処理を表します。なお autoconf, m4, libtool がないと buildconf でエラーになるため、apt あるいは rpm であらかじめ導入しておく必要があります。

あとはブラウザでのアクセスや “telnet localhost 80” などを実行して、Apache が正常に起動したことを確認しましょう。うまく起動しない場合には /usr/local/apache2/logs/error_log を参照しながら、設定を調整してください。(httpd.conf にも記載されていますが) カーネルの設定によっては図 12 のようなエラーが表示されて起動しないことがあります (Debian Woody

WebDAV バージョニング実装 Subversion

図 12 setgid でのエラー

```
[alert] (22)Invalid argument: setgid: unable to set group id to Group xxxxxx
```

のkernel-image-2.2.19-udma100-ext3-i386の場合など)。そのときにはhttpd.confのGroupをnogroup（か何か適当なグループ）にして再度試してみてください。

Subversion のインストール

Subversion は現在絶賛開発中ですから、まだ動作が不安定な箇所が残っています。そのため最新版のリビジョン718 (subversion-r718) と、少し前のリビジョン88 (subversion-M3) との両方のインストール方法を説明します。

まず次の場所からSubversion とBerkeley DB 3.3.11以降を入手してください（クライアントとしてのみ使用する場合にはBerkeley DBは不要です）。ここではsubversion-r718.tar.gz とdb-3.3.11.tgzを利用します。ネットワーク経由でSubversion サーバとクライアントがアクセスする場合には、サーバおよびクライアントマシンの両方でインストールを行ってください。

Subversion の入手先

<http://subversion.tigris.org/servlets/ProjectDownloadList>

Berkeley DB の入手先

<http://www.sleepycat.com/update/index.html>

インストールは図13のように行いますが、ここで1つ注意点があります。みなさんがこの記事を読んでいるころには新しい（修正された）バージョンが出ているかもしれませんが、現バージョンではSubversionに含まれているaprは正常に動作しません。図13 - ①のように同梱版のaprを削除し、Apacheのインストールディレクトリ以下のaprを指定してconfigureを実行するようにしてください。

リビジョン88をインストールする場合は図14のようにします。

Subversion サーバを使うには、どのディレクトリ（URI）をSubversionクライアントにアクセスさせるのかを指定する必要があります。図15にhttp://IPアドレス/svnにアクセスさせるときの設定を示します。図15をhttpd.confに追加してApacheを再起動してく

図 13 Subversion と Berkeley DB のインストール

```
$ tar zxvf subversion-r718.tar.gz
$ cd subversion-r718
$ tar zxvf ../db-3.3.11.tar.gz
$ mv db-3.3.11 db
$ rm -rf apr
$ ./configure --with-apxs=/usr/local/apache2/bin/apxs --with-apr-includes=/usr/local/apache2/include --with-apr-lib=/usr/local/apache2/lib
$ make
$ su
# make install
```

図 14 リビジョン 88 のインストール

```
$ tar zxvf subversion-M3-r88.tar.gz
$ cd subversion-r88
$ tar zxvf ../db-3.3.11.tar.gz
$ mv db-3.3.11 db
$ ./configure --with-apxs=/usr/local/apache2/bin/apxs
$ make
$ su
# make install

# echo "/usr/local/BerkeleyDB.3.3/lib" >> /etc/ld.so.conf
# ldconfig
```

図 15 httpd.conf の設定

```
LoadModule dav_svn_module modules/libmod_dav_svn.so

<Location /svn>
DAV svn
SVNPath /home/svn
</Location>
```

ださい（おそらくLoadModuleの行はすでに設定されているはずですが）。

Windows 版 Subversion クライアントについて

SubversionのWebサイトからは、Windowsで動作するSubversionクライアントのバイナリ（.exeファイル）も入手できます。クライアントとしての利用以外はまだ不安定ですが、ネットワーク経由でSubversionサーバとやりとりするだけであれば、問題なく使えるかと思います。使い方はUNIX版とまったく同じです。

Apache および Subversion のアンインストール

参考までにApacheおよびSubversionのアンインストール方法について説明しておきます。どちらも非常に簡単で、まず“/usr/local/apache2/bin/apachectl stop”を実行してApacheを停止しておきます。停止忘れてファイルを削除してしまった場合、“killall

HTTP 拡張プロトコルでバージョン管理をする

httpd ” を実行すればOKです。

Apache をアンインストールするには /usr/local/apache2 をディレクトリごと削除し、次に以下のライブラリを削除すれば終了です。

Apache の使用するライブラリ

```
/usr/local/lib/libapr.*
/usr/local/lib/libexpat.*
/usr/local/lib/libneon.*
```

Subversion のアンインストールは以下のファイルを削除してください。

Subversion アンインストール時に削除するファイル

```
/usr/local/man/man1/svn.1
/usr/local/info/svn*
/usr/local/bin/neon-config
/usr/local/bin/svn*
/usr/local/lib/libsvn_*
```

/usr/local/BerkeleyDB.3.3/ (他のアプリケーションで使用していない場合)

リポジトリの歩き方

Subversion を使うためには、まずリポジトリを準備する必要があります。最初にローカルでの使用方法を示し、その後ネットワーク経由でのアクセス方法について説明します。

リポジトリの作り方

ここではリポジトリを /svn に作成するとします。“svnadmin create /svn” を実行してください。するとホームディレクトリ以下にsvn というディレクトリができ、/svn にリポジトリが作成されました。非常

図 16 リポジトリにファイルを追加する例

```
$ mkdir ~/tmp
$ cd ~/tmp
$ touch 1 2 3
$ svn import file://$HOME/svn . test
Adding      ./test
Adding      ./test/1
Adding      ./test/2
Adding      ./test/3
Commit succeeded.
```

に簡単ですね。リポジトリに関する操作を行うときには、このディレクトリ(リポジトリ)を指定する必要があります。

ファイルの追加

ファイルをリポジトリに追加しましょう。図16を実行してみてください。リポジトリのtest ディレクトリ以下に1, 2, 3 というファイルが追加されます。複数のプロジェクトなどを同一のリポジトリ内で管理する際には、図16のtest を適当なディレクトリに変えてあげればよいでしょう(たとえばXXXBankProject など)。

チェックアウト

リポジトリから追加したファイルをチェックアウトしてみましょう。図17を実行すると、/cotest/test 以下に先ほど追加したファイルが作成されます。このディレクトリにあるsvn ディレクトリ(リビジョン88の場合はSVN ディレクトリ)はSubversion が内部的に使用するディレクトリなので削除しないでください。“svn log” を実行すると、チェックアウトしたツリーはリビジョン1であることがわかります(リビジョン88ではREPORT メソッドが実装されていないため、“svn log” は実行できません)。

コミット

ファイルを更新してリポジトリにコミットします。図18を実行後に“svn log” を実行してみてください。リビジョンは2になっているはずです。

なお図18 - ①ではファイルごとの変更点の表示を、図18 - ②ではローカルにチェックアウトしてきたツリー(ワーキングディレクトリと呼びます)のうち、まだリポジトリにコミットされていない(あるいはリポジトリでは変更されているがローカルに反映されてい

図 17 チェックアウト例

```
$ mkdir ~/cotest
$ cd ~/cotest
$ svn co file://$HOME/svn/test
A test/1
A test/2
A test/3
```

WebDAV バージョニング実装 Subversion

ない) 変更点を表示します。“svn status”で表示されるマークの意味は表4を参照してください。プロパティが変更・更新された場合には、それぞれのマークの後ろにMやUなどのマークが付きます。たとえばプロパティだけが変更された場合は“_M”と表示され、ファイルの中身も変更されたときには“MM”と表示されます。

ファイルのリネームと過去のリビジョンの取り出し方

CVSとSubversionの差異の項で、Subversionはツリー単位でバージョン(リビジョン)を管理しているということを説明しました。ファイルをリネームして、このことを検証してみましょう。図19を参照してください。なおドキュメントのどこにも記述はありませんが、図19-①の“svn status”(省略形は“svn st”)が表示される+はファイルがリネーム(あるいは移動)されたことを表しているものと思われます。

“svn log”を実行すると、リビジョン3が作成されたことを確認できます。次にリネーム前のリビジョン、つまりリビジョン2を取り出してみましょう。リビジョンを指定するには図20のように-rオプションを使用します(オプションやコマンドの省略形は基本的にCVSと同じです)。ちなみにsvnに関する説明は“man svn”で表示されますが、拙訳のものを<http://www.gihyo.co.jp/wdpress/source/vol7/svn.1>に置いてあります。Subversionの開発が落ちついた段階で、JM Project (<http://www.linux.or.jp/JM/>)に登録する予定です。

1-2というファイルは削除され、1というファイルが復活したかと思えます。さらに“svn -r 1 up”を実行してみましょう。subというディレクトリが削除され、中身が空の1, 2, 3というファイルが作成されたはず(リビジョン1ではファイルをtouchしただけですから)。最後に“svn update”を実行すると、先ほどの図19を実行した後の状態に戻ります。

1-2というファイルは削除され、1というファイルが復活したかと思えます。さらに“svn -r 1 up”を実行してみましょう。subというディレクトリが削除され、中身が空の1, 2, 3というファイルが作成されたはず(リビジョン1ではファイルをtouchしただけですから)。最後に“svn update”を実行すると、先ほどの図19を実行した後の状態に戻ります。

ブランチの作成(コピー)

Subversionにはブランチというものは存在せず、「コ

表4 “svn status”実行時のマークの意味

マーク	意味
A	追加 (Add)
D	削除 (Delete)
R	置き換え (Replace)(ファイルなどを削除した後に、追加した場合)
M	変更 (Modify)
U	更新 (Update)(リポジトリ上で行われた変更がローカルに反映されていない場合)
G	マージ (merGed)
C	コンフリクト
-	ワーキングディレクトリがなく、リポジトリにだけ存在するファイル/ディレクトリ

図18 コミット例

```
$ echo 111 > 1
$ svn rm 2
D 2
$ rm 2
$ svn mkdir sub
A sub
$ svn diff .....①

Index: ./1
-----
--- ./svn/text-base/1.svn-base Tue Jan 1 10:10:00 2002
+++ ./1 Tue Jan 1 10:15:00 2002
@@ -0,0 +1 @@
+111

$ svn status .....②
M ./1
D ./2
A ./sub
$ svn ci -m '1st commit'
Deleting /home/cz/cotest/test/2
Adding /home/cz/cotest/test/sub
Sending /home/cz/cotest/test/1
Commit succeeded.
```

図19 ファイルのリネーム

```
$ svn move 1 1-2
A 1-2
D 1
$ rm 1

$ svn st .....①
D ./1
A + ./1-2

$ svn ci -m '2nd commit'
Deleting /home/cz/cotest/test/1
Adding /home/cz/cotest/test/1-2
Commit succeeded.
```

図20 リビジョン2の取り出し

```
$ svn -r 2 up
A ./1
D ./1-2

$ ls
1 3 sub
```

HTTP 拡張プロトコルでバージョン管理をする

ピー」という概念が導入されました。コピーを行うのは非常に簡単で、図21のように“svn copy”を実行するだけです。カレントディレクトリにtestディレクトリが作成され、その下に1-2および3というファイル、subというディレクトリが作成されます。もちろんコピー先に適当なサブディレクトリを指定することもできます。あとは作成されたtestディレクトリ以下をCVSで言うブランチと同様に、自由に利用できます。

ネットワーク経由でのアクセス

svnコマンドのリポジトリの場所はURIで指定していることに気付いたでしょうか。ローカルでのアクセスもネットワーク経由でのアクセスも、クライアント側から見ればほとんど差はありません。説明を読むより実際の例を見た方が理解しやすいでしょうから、例によって実行例を図22（サーバ上で実行）、図23（クライアント上で実行）に示します。

ただしいくつかの注意点があります。SubversionサーバにはApacheを経由してアクセスするため、サーバ上のリポジトリをApacheが読み書きできなければなりません。そのため図22のようにして、svnadminコマンドで作成したリポジトリの所有権をApacheが動作しているユーザ（httpd.confのUserディレクティブの値）に変更してやる必要があります。あとはリポジトリの場所がローカル（file://）なのかリモート（http://）なのかの違いだけです。

参考までに言うと、Subversion内部ではローカルアクセスとリモートアクセスは明確に分けられていま

図 21 コピーの作成とコミット

```
$ svn copy file://$HOME/svn/test .
A ./1-2
A ./3
A ./sub
A ./test

$ svn ci -m 'created a branch'
Adding /home/cz/cotest/test/test
Commit succeeded.
```

図 22 ネットワーク経由でのリポジトリアクセスの準備（サーバ上）

```
$ su
# svnadmin create /home/svn
# chown -R nobody /home/svn
```

す。ローカルアクセスのときにはBerkeley DBおよびlibsvn_ra_localというライブラリを、リモートアクセスのときにはlibsvn_ra_davというライブラリを使用します（どちらも/usr/local/lib/にインストールされているはず）。

Subversion の活用例

最も有効かつ大本命の活用例は、CVSと同様に共同開発におけるプログラムソースおよびドキュメントの共有です。何度も述べている通り、Subversionのバージョン管理はツリー単位かつコミットがアトミックのため、ファイル単位で管理されるCVSとは異なり「あるファイルは正常にコミットされ、あるファイルはコンフリクトが発生したためにコミットされていない」という状態は発生しません。もちろんコミット時に注意していれば問題はありませんが、徹夜続きでの開発作業など頭が正常でないときなどには、コンフリクトに気付かずそのまま開発を続行してしまい、わけがわからない状態になることがよくあります。機械がまか

図 23 ネットワーク経由でのリポジトリアクセス（クライアント上）

```
$ touch 1 2 3
$ svn import http://サーバのIPアドレスあるいはFQDN/svn . test2
Username: ENTER
Password: ENTER

Adding ./test2
Adding ./test2/1
Adding ./test2/2
Adding ./test2/3
Commit succeeded.

$ mkdir ~/cotest2
$ cd ~/cotest2
$ svn co http://サーバのIPアドレスあるいはFQDN/svn/test2
Username: ENTER
Password: ENTER

A test2/3
A test2/2
A test2/1

$ cd test2
$ echo test >> 1
$ svn ci -m '1st commit'
Sending /home/cz/cotest2/test2/1
Commit succeeded.

$ svn -r 1 up
? ./svn_update.8044.00001.ra_dav
U ./1
$ cat 1
$
```

WebDAV バージョニング実装 Subversion

せられることは機械にまかせてしまうのが正義ではないでしょうか。)

筆者自身は、社内（あるいは個人専用の）の文書管理に利用しています。Subversion のもう一つの利点であるバイナリファイルを自動判別してくれる機能は、単純なようですが実際に利用してみるとそのありがたみがわかります。この原稿も最初はCVSで管理して執筆していたのですが、図を作成しバイナリファイルを追加するたびに -kb オプションを忘れてしまい何度も、ファイルを追加、削除、-kb で再追加、ということを繰り返していたため途中でSubversionに切り替えたとこ、たかだかファイルの種類について考えなくてもよくなったのですが、非常に楽になりました。

設定例

ドキュメント共有のために必要な設定は、ローカルのディレクトリ構造をそのままインポートするだけです。あまりな説明と思うかもしれませんが、CVSもSubversionもローカルにあるディレクトリ構造をそのままリポジトリに登録できるため、「設定 = ディレクトリ構造をどうするか」という話になってしまいます。ローカルにある図24のディレクトリ構造を192.168.10.1というSubversionサーバに登録する例を図25に示します。「インポートしてチェックアウトするだけ」というのがわかると思います。

さいごに

長い間、登場（と実用になること）を期待されていたSubversionですが、最新版においてはリモートアクセスがうまくいかなくなるなど、残念ながら安定して運用できるとはまだ言い難い状況です。ただローカル内での利用は問題ありませんし、過去にはリモートアクセスが可能だったことから、近日中に修正されることを期待しています。

SubversionクライアントのSSL機能が正常に動作するようになれば、現在CVSで管理されているプロジェクトからSubversionに置き換わるものも増えてくるのではないのでしょうか。ちなみにSubversionのプロジェクトは以前はCVSで管理されていましたが、今では

Subversion自身で管理されています。2002年はWebDAVやSubversionから目が離せません。ぜひ試してみてください。Web

参考URL

WebDAV Resources
<http://www.webdav.org/>
 IETF WEBDAV Working Group (WebDAV を実装しているサーバ・クライアントの一覧がある)
<http://www1.ics.uci.edu/pub/ietf/webdav/>
 WebDAV-jp メーリングリスト
<http://begi.net/webdav/>
 IETF Delta-V Working Group
<http://www.webdav.org/deltav/>
 Subversion
<http://subversion.tigris.org/>
 WebDAV Resources JP
<http://webdav.todo.gr.jp/>

図 24 サンプルディレクトリ構造

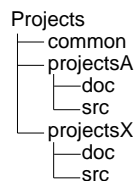


図 25 サンプルディレクトリ構造の準備手順

```

下記はサーバ上で実行
$ su
# svnadmin create /home/svn
# chown -R nobody /home/svn

下記はクライアント上で実行
$ cd /home/tmp/Projects
$ svn import http://192.168.10.1/svn . Projects
Username: [ENTER]
Password: [ENTER]

Adding      ./Projects
Adding      ./Projects/common
Adding      ./Projects/projectX
Adding      ./Projects/projectX/src
Adding      ./Projects/projectX/doc
Adding      ./Projects/projectA
Adding      ./Projects/projectA/doc
Adding      ./Projects/projectA/src
Commit succeeded.

$ mkdir ~/svn; cd ~/svn
$ svn co http://localhost/svn/Projects
Username: [ENTER]
Password: [ENTER]

A  Projects/projectX
A  Projects/projectX/src
A  Projects/projectX/doc
A  Projects/common
A  Projects/projectA
A  Projects/projectA/src
A  Projects/projectA/doc
  
```