第 1 回 Subversion ゼミ

ゼミ担当者　：　川崎 考蔵，木田 直人
指導院生　：　菅原 麻衣子，木浦 正博
開催日　：　2007 年 4 月 19 日

1　はじめに

ソフトウェアやシステムの開発は，ソースコードなどの修正を繰り返して改良していくことが活動の基本である．ただし，開発現場ではソースコードやドキュメントに修正を加えるのが一人だけとは限らず，複数の開発者が並行して開発を進める場合も存在する．このため，「いつ」，「どこを」，「どう」修正したのかを管理することが重要となってくる．また，個人で開発する場合も，改良の履歴を管理することは非常に有用である [1]．一般に，変更の履歴などをバージョンとして管理するが，本ゼミではバージョン管理システム「Subversion」を用いたバージョン管理の方法について学ぶ．

2　Subversion

Subversion とは，ファイルのバージョン管理をするアプリケーションソフトである．バージョン (version) とはファイルに対して変更を加え，その変更を確定するごとに上がっていく管理番号であり，リビジョン (revision) とも言う．Subversion は主にプログラムの開発現場などで使用されるが，プログラムのソースコードをはじめ，どんなファイルでも管理できる．また，複数人が同時に同じファイルを編集することができるが，この際手元で行った修正と他のユーザが行った修正が同じ箇所である場合，変更の衝突が発生する．これをコンフリクト (conflict) と呼ぶ．コンフリクトが発生した場合，ユーザ同士が話し合ってコンフリクトを解消する必要がある．また，コンフリクトが発生しない場合は両方の変更を自動的に統合することができる．

Subversion の概念図を Fig. 1 に示す．Subversion ではファイル群は一箇所にまとめられて管理される．このファイルをまとめたものをリポジトリ (repository) といい，この中にファイルの変更履歴も記録される．また，リポジトリにあるファイル群の編集は，リポジトリの内容をローカルにコピーし，そこで行う．これをワーキングコピー (working copy) という．

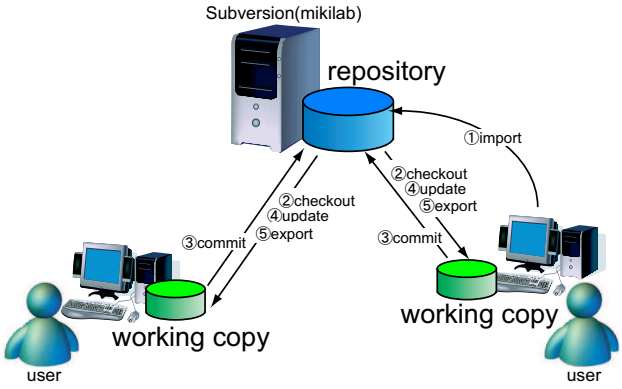

Fig. 1　Subversion の概念図 (出典：参考文献 [2])

3　基本用語

ここではバージョン管理システムで用いられる基本用語について簡単に説明する．

- バージョン (version)

  ファイルに対して変更を加え，それをバージョン管理システムに登録するごとにそのファイルのバージョンと呼ばれる番号が上がっていく．また，バージョンのことをリビジョン (revision) ともいう．

- リポジトリ (repository)

  バージョン管理システムでは，ファイル群は一箇所にまとめられて管理される．このファイルをまとめたものをリポジトリという．この中にファイルの変更履歴 (version history) も記録される．

- ワーキングコピー (working copy)

  リポジトリにあるファイル群を編集するためコピーしたもので，ファイル群のワーキングコピー内での変更履歴も，同じディレクトリ内で保存されている．なお，同じディレクトリを複数のリポジトリのワーキングコピーとすることはできない．

- チェックアウト (checkout)

  リポジトリからワーキングコピーを取り出すことをチェックアウトという．

- コミット (commit)

  ワーキングコピーでの更新結果を確定し，リポジトリに変更を反映させることをコミットという．

- アップデート (update)

  あるワーキングコピーでは編集してなくても，他の開発者のワーキングコピーからのコミットにより，リポジトリが更新されている可能性がある．リポジトリの更新状況をワーキングコピーに反映させることをアップデートという．

- コンフリクト (conflict)

  ファイルの変更をコミットする際に，手元で行った修正と他からコミットされた修正が，同じ箇所において異なった修正である場合，コンフリクトしているという．コンフリクトが検出された際，ユーザはワーキングコピーを編集しコンフリクトを解消する必要がある．

- インポート (import)

  最初にリポジトリにファイル群を登録することをインポートという．

- エクスポート (export)

  編集するつもりはなく，管理ファイルを除いて対象ファイルだけを取り出す操作のことをエクスポートという．ソフトウェアのリリースなどはこの一例である．

4　リポジトリの構成

システム開発において，システム A を主に進めている場合を例に説明する．開発を進めた結果，システム A が一旦完成したとする．まだシステム A にはバグが存在している可能性があるが，新しい機能の追加を平行して行う．その際は，今後バグ修正を進めていくシステム A を開発の主流として残しておき，機能追加や構成の変更などを行うための開発の流れをシステム A+α として作る．また，システム A をベースとした別のシステム B を開発する際に，システム A の完成時のプログラムを利用する．この場合，システム A の完成時のリビジョンを目印として記録しておくと，後で便利である．

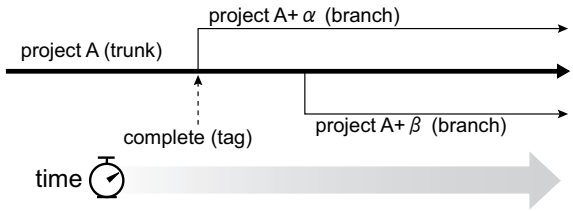

Fig. 2　開発の流れ (出典：参考文献 [3])

システム開発では上記のような場合が考えられるが，Subversion では，ディレクトリの構成を工夫することでこれを実現している．具体的には，リポジトリの構成において標準化されたものがあり，以下に示す 3 つのディレクトリをあらかじめ作成することが推奨される．

/trunk
/branches
/tags

システム A のような開発の主流となるデータは trunk ディレクトリで管理する．システム A+α のような主流の開発から分岐したものをブランチといい，branches の下にブランチを作成する．プログラムが完成したとき，またブランチを作ったときなどのリビジョンの目印をタグといい，タグを管理するためのディレクトリが tags である．

ブランチやタグの作成方法の詳細は後ほど説明するが，ブランチもタグもディレクトリをコピーすることで作成する．すなわち，ブランチとタグに違いはなく，システム開発者がコピーしたディレクトリをブランチとして，もしくはタグとして扱うというポリシーである．

また，リポジトリが複数プロジェクトを含む場合は，Fig.3 のようにプロジェクトごとに構成をインデックス化することが望ましい．

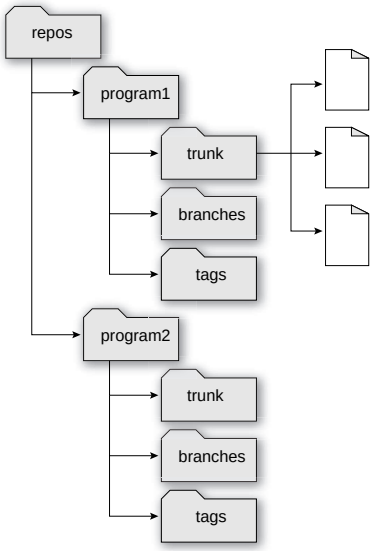

Fig. 3　リポジトリの構成 (出典：参考文献 [3])

5 Subversion コマンド

Subversion を用いてバージョン管理する際に用いるコマンドを紹介する.

- svnadmin create
  カレントディレクトリにリポジトリを作成して初期化する.

```
svnadmin create [リポジトリ名]
```

  Subversion を利用する際，このコマンドで，あらかじめリポジトリを作成しておく必要がある.

- import
  バージョン管理するファイル群をリポジトリに登録する.

```
svn import -m "コメント" URL[インポート先]
```

  カレントディレクトリにあるファイルが全て，リポジトリに登録される.

- checkout
  ワーキングコピーを取り出す.

```
svn checkout URL[チェックアウト元] ディレクトリ名
```

  リポジトリから指定した名前のディレクトリの中に，編集用にファイル群を取り出す．ディレクトリ名を指定しなければ，リポジトリと同じ名前のディレクトリが作成される．なお,「checkout」は「co」と略されたコマンドでも同様の動作となる.

- commit
  ワーキングコピーでの編集をリポジトリへ反映させる.

```
svn commit -m "コメント"
```

- add
  Subversion の管理下にファイルを追加する.

```
svn add [ファイル名]
```

  コミットするまでリポジトリには反映されない.

- delete
  ファイルをリポジトリから削除する.

```
svn delete ファイル (ディレクトリ)
```

  コミットするまでリポジトリには反映されない.

- move
  「move」には 2 通り使い方があり，リポジトリに対して UNIX のコマンドの「mv」と同じ操作が可能である.
  ファイル (ディレクトリ) 名の変更を行う.

```
svn move 古いファイル名　新しいファイル名
```

  ファイル (ディレクトリ) の移動

```
svn move ファイル名　移動先
```

  コミットするまでリポジトリには反映されない.

- status
  ワーキングコピーにあるファイルの更新状態を表示させる.

```
svn status
```

  行のはじめに表示される大文字英字がファイルの状態を示す (Table 1)．ワーキングコピーが変更されていない場合，何も表示されない.

Table 1　記号の表す状態

記号	ファイルの状態
?	Subversion の管理下にない
A	add してまだコミットされていない
M	修正部分がマージ出来た
D	ファイルが削除されている
C	コンフリクトしている

  「-v」オプションをつけることでより詳細を表示することができる.

```
svn status -v
```

  左から順に，ファイルの状態，ワーキングコピーのバージョン名，最後にコミットしたバージョン，コミットしたユーザ名，ファイル名が表示される.

- export
  ファイルを編集用ではなく取り出す.

```
svn export URL[エクスポート先]
```

  チェックアウトと違いエクスポートしたデータは,

編集用ではないのでバージョン管理用のファイルを含まない.

- diff
  ワーキングコピー内の変更された点を参照する.

```
svn diff
```

  ファイルに編集前後の削除および追加部分を知ることが出来る.

- log
  指定したファイルの作業履歴を調べる.

```
svn log ファイル名
```

  変更のあった際のリビジョン番号，変更したアカウント名，変更時刻，変更操作，が表示される.

- revert
  ワーキングコピーでの修正を取り消す.

```
svn revert ファイル名
```

  コミットする前の段階で，ワーキングコピーのファイルを修正前の状態に戻すことが出来る.

- mkdir
  リポジトリ内にディレクトリを作成する.

```
svn mkdir ディレクトリ名
```

  コミットするまでリポジトリには反映されない.

- cat
  指定したファイルまたは URL の内容をチェックアウトすることなしに表示する．なお，ディレクトリの表示については，前節で示した「svn list」を用いる.

```
svn cat URL[ディレクトリ名]
```

- copy
  作業コピーまたはリポジトリ中のファイルやディレクトリをコピーする．なお，コピー元 (SRC) とコピー先 (DST) は，作業コピー (WC) 中のパスでも URL でも構わない．ただし，ファイルは 1 つのリポジトリの内部でのみコピー可能であり，Subversion はリポジトリ間コピーをサポートしていない.

```
svn copy SRC[コピー元] DST[コピー先]
```

6 基礎演習

6.1 リポジトリの作成

まずリポジトリを作成する．リポジトリの作成は svnadmin create コマンドで行うが，ssh コマンドを用いてリポジトリを mikilab の作成したいディレクトリまで移動してから行う．

Step 1

「ssh [アカウント]@mikilab.doshisha.ac.jp」で mikilab にログイン

```
$ ssh nkida@mikilab.doshisha.ac.jp
```

Step 2

「/home/svn/project/svnsemi」に移動

```
nkida@mikilab: $ cd /home/svn/project/svnsemi
```

Step 3

「svnadmin create」を用いて「/home/svn/project/svnsemi」内に自分の名前のリポジトリを作成

```
hishida@mikilab:/home/svn/project/svnsemi$ svnadmin create nkida
```

6.2 インポート

リポジトリが作成されたので，その中にバージョン管理したいファイルを入れる (インポート)．ここでは test というディレクトリを作成し，その中に編集するファイル郡を入れる trunk，分岐した編集ファイルを入れる branches，完成したプログラムやブランチを作成したときにそのファイルを入れる tags を作成し，その 3 つのディレクトリを import コマンドによりインポートする．

また本来ならば，trunk の中に編集したいファイルを入れておくが，本ゼミでは簡単のためファイルを入れておかない．

Step 1

「logout」でログアウト

```
hishida@mikilab:/home/svn/project/svnsemi$ logout
Connection to mikilab.doshisha.ac.jp closed.
```

Step 2

「test」ディレクトリを作成（名前は何でもよい）

```
$ mkdir test
```

Step 3

「test」ディレクトリの中に「trunk」「branches」「tags」ディレクトリを作成

```
$ mkdir test/trunk

$ mkdir test/branches

$ mkdir test/tags
```

Step 4

「test」ディレクトリの中に移動

```
$ cd test
```

Step 5

「svn import」を用いて「/home/svn/project/svnsemi」内のリポジトリにインポート（ローカルから mikilab へのアクセスなので「svn+ssh:」を用いて URL 指定）

```
test$ svn import svn+ssh://nkida@mikilab.doshisha.ac.jp/home/svn/project/svnsemi/nkida -m 'initial import'

Adding trunk
Adding branches
Adding tags
Committed revision 1.
```

6.3 ワーキングコピーの作成

インポートが完了し，バージョン管理の下で編集したいファイルをリポジトリに登録することができたので，次に作業するためのディレクトリであるワーキングコピーを svn checkout コマンドで作成する．

リポジトリは mikilab にあるので，svn+ssh コマンドにより mikilab にアクセスしてワーキングコピーを作成する．

Step 1

「test」ディレクトリから出る

```
test$ cd ../
```

Step 2

「svn checkout」を用いてリポジトリからチェックアウト（ワーキングコピーの名前は何でもよい）

```
$ svn checkout svn+ssh://nkida@mikilab.doshisha.ac.jp/home/svn/project/svnsemi/nkida workingcopy

A workingcopy/trunk
A workingcopy/branches
A workingcopy/tags
Checked out revision 1.
```

6.4 ワーキングコピーへのファイルの追加

ワーキングコピーの作成が完了したので，次にワーキングコピー内でのファイルの作成を行う．ワーキングコピー内にファイルを作成しただけではそのファイルはリポジトリに登録されないので，svn add コマンドによりそのファイルをリポジトリに登録する (ただしコミットするまでその変更は確定されない)．

Step 1

ワーキングコピーの trunk ディレクトリ内に sample.txt を作成

```
$ cd workingcopy

workingcopy$ touch trunk/sample.txt
```

Step 2

「svn status」を行い，sample.txt のファイルの状態が「?」で，Subversion の管理下にないことを確認する

```
workingcopy$ svn status
? trunk/sample.txt
```

Step 3

「svn add」によって sample.txt を Subversion の管理下に置く

```
workingcopy$ svn add trunk/sample.txt
A trunk/sample.txt
```

6.5 コミット

リポジトリに新しくにファイル (sample.txt) を登録されたことを確定するために，svn commit コマンドによりコミットを行う．コミットを行うとリポジトリが更新され，バージョンが上がる．

Step 1

「svn status」を行い，sample.txt のファイルの状態が「A」で，ワーキングコピーに追加されていることを確認

```
workingcopy$ svn status
A trunk/sample.txt
```

Step 2

「svn commit」によってコミット

```
workingcopy$ svn commit -m 'add trunk/sample.txt'

Adding trunk/sample.txt
Transmitting file data .
Committed revision 2.
```

6.6 ファイルの変更

ワーキングコピー内のファイル (sample.txt) を変更する．ワーキングコピー内のファイルを変更するだけでは，リポジトリには反映されないので，コミットも行う．

Step 1

ファイルを以下の内容に変更

```
int a,b,c;
a = 3;
b = 5;
c = a*b;
```

```
workingcopy$ vi trunk/sample.txt
```

Step 2

「svn status」を行い，sample.txt のファイルの状態が「M」で，変更が加えられているのを確認する

```
workingcopy$ svn status
M trunk/sample.txt
```

Step 3

「svn commit」によってコミット

```
workingcopy$ svn commit -m 'edit trunk/sample.txt'

Sending trunk/sample.txt
Transmitting file data .
Committed revision 3.
```

7 追加演習

7.1 タグの付加

trunk ディレクトリを tags ディレクトリ内にコピーして，タグの付加を行うことにより，リビジョンの目印を作る．

Step 1

「svn copy」を用いて trunk ディレクトリを tags ディレクトリ内にコピー（その時にコピー先のディレクトリを「release1.0」といったような名前に変更し，ディレクトリ名によってタグ付けを行う）

```
workingcopy$ svn copy trunk tags/release1.0
A tags/release1.0
```

Step 2

変更をコミット

```
workingcopy$ svn commit -m 'tagging release1.0'

Adding tags/release1.0
Adding tags/release1.0/sample.txt
Committed revision 4.
```

7.2 作業の別ラインの作成

機能追加や構成の変更などを行うための branches ディレクトリに trunk ディレクトリの内容をコピーする．

Step 1

trunk ディレクトリを branches ディレクトリ内にコピー（コピー先のディレクトリの名前を変更し，以降は作業の別ラインとして trunk とは別の作業を行う）

```
workingcopy$ svn copy trunk branches/sampleBranch
A branches/sampleBranch
```

Step 2

変更をコミット

```
workingcopy$ svn commit -m 'copy trunk to branches/sampleBranch'

Adding branches/sampleBranch
Adding branches/sampleBranch/sample.txt
Committed revision 5.
```

7.3 ファイルの変更内容の確認

ワーキングコピー内の変更された点を参照するための diff コマンドを使ってみる．変更前よりも増えた行の前には＋が付き，減った行の前には－が付く．

Step 1

trunk ディレクトリ内の sample.txt を以下の内容に変更（下線部が追加箇所）

```
int a,b,c;
a = 3;
printf(a);
b = 5;
c = a*b;
```

```
workingcopy$ vi trunk/sample.txt
```

Step 2

「svn diff」によって変更内容を確認

```
workingcopy$ svn diff
Index: trunk/sample.txt
===================================================================
— trunk/sample.txt (revision 2)
+++ trunk/sample.txt (working copy)
@@ -1,4 +1,5 @@
int a,b,c;
a = 3;
+printf(a);
b = 5;
c = a*b;
```

7.4 ワーキングコピーの変更の取消

前節で変更を加えた内容を，revert コマンドで取り消す．

Step 1

「svn status」行い，trunk 内の sample.txt のファイルの状態が「M」で，変更が加えられているのを確認する

```
workingcopy$ svn status
M trunk/sample.txt
```

Step 2

「svn revert」によって変更を取消

```
workingcopy$ svn revert trunk/sample.txt
Reverted 'trunk/sample.txt'
```

7.5 コミット後のファイルの復元

Step 1

「svn delete」によって trunk 内の sample.txt を Subversion 管理下から削除

```
workingcopy$ svn delete trunk/sample.txt
D trunk/sample.txt
```

Step 2

変更をコミット

```
workingcopy$ svn commit -m 'delete trunk/sample.txt'

Deleting trunk/sample.txt
Committed revision 6.
```

Step 3

「-r [リビジョン番号]」のオプションを用いたコピーによって，ワーキングコピーに過去のリビジョンのファイルをコピーする（trunk 内の sample.txt はリビジョン 5 以前に存在）

```
workingcopy$ svn copy -r 5 svn+ssh://hishida@mikilab.doshisha.ac.jp/home/svn/project/svnsemi/hishida/
trunk/sample.txt trunk/

A trunk/sample.txt
```

Step 4

変更をコミット

```
workingcopy$ svn commit -m 'recover trunk/sample.txt'

Adding trunk/sample.txt
Committed revision 7.
```

7.6 過去のバージョンの復元

Step 1

「svn delete」によって trunk 内の sample.txt を Subversion 管理下から削除

```
workingcopy$ svn delete trunk/sample.txt
D trunk/sample.txt
```

Step 2

変更をコミット

```
workingcopy$ svn commit -m 'delete trunk/sample.txt'

Deleting trunk/sample.txt
Committed revision 8.
```

Step 3

リビジョン 7 からリビジョン 6 への過去へさかのぼる差分をワーキングコピーにマージすることによって，リビジョン 6 を復元

```
workingcopy$ svn merge trunk@7 trunk@6 trunk

A trunk/sample.txt
```

Step 4

変更をコミット

```
workingcopy$ svn commit -m 'recover rivision6 in trunk'

Adding trunk/sample.txt
Committed revision 9.
```

参考文献

1) 木田 清香，千田 智治，村上 耕平．2006 年度 UNIX ゼミ Subversion マニュアル．

2) 荒久田 博士，折戸 俊彦，市川 親司．2003 年度 LINUX 開発アプリケーションゼミ資料．

3) サブバージョンによるバージョン管理．http://subversion.bluegate.org/doc/book.html#svn.branchmerge.using.create