Tools for Writing Better PHP Code
Version Control with Subversion

Prepared by
Jeff Knight (jeff dot knight at nyphp.org) &
Andrew Yochum (andrew at plexpod.com)
for New York PHP

Why Version Control?

Developing with Others

On projects with more than one developer, version control prevents them from overwriting each other's changes.

Almost Infinite Undo

A project in version control can be "rolled back" to any previous state.

Publishing

Most version control systems can also be used to publish and distribute the project to a wider audience.

Why Subversion (as opposed to cvs)?

SVN is faster & in general, needs much less server memory

Versions entire directories and structure instead of just files

Random metadata

Cheap copies

Works well weith binaries

Atomic transactions to the repo

CVS is based on RCS and inherits lots of legacy crap as a result

Popularly known as "cvs without the suck"

Installing Subversion

Subversion is built on a portability layer called APR (the Apache Portable Runtime library). This means Subversion should work on any operating system that the Apache httpd server runs on: Windows, Linux, all flavors of BSD, Mac OS X, Netware, and others.

The easiest way to get Subversion is to download a binary package built for your operating system. Subversion's website (`http://subversion.tigris.org`) often has these packages available for download, posted by volunteers. The site usually contains graphical installer packages for users of Microsoft operating systems. If you run a Unix-like operating system, you can use your system's native package distribution system (RPMs, DEBs, the ports tree, etc.) to get Subversion.

Alternately, you can build Subversion directly from source code. From the Subversion website, download the latest source-code release. After unpacking it, follow the instructions in the `INSTALL` file to build it. Note that a released source package contains everything you need to build a command-line client capable of talking to a remote repository (in particular, the apr, apr-util, and neon libraries). But optional portions of Subversion have many other dependencies, such as Berkeley DB and possibly Apache httpd. If you want to do a complete build, make sure you have all of the packages documented in the `INSTALL` file.

Excellent Manual: http://svnbook.red-bean.com

The O'Reilly Media published book *Version Control with Subversion* written by Ben Collins-Sussman, Brian W. Fitzpatrick, C. Michael Pilato is available for **free** on the web in HTML and PDF formats.

Repository Storage: Berkeley DB vs. FSFS

Just to keep things from being too easy, Subversion has two methods it uses to store its repositories. One type of repository stores everything in a Berkeley DB database; the other kind stores data in ordinary flat files they refer to as FSFS. Originally, it used the Berkeley DB method as the default, but has recently changed the default to FSFS.

In my opinion, it is easier to install and run subversion if you forget about Berkeley DB altogether. I also feel that it is easier to manage permissions.

Subversion Repository Data-Store Comparison

Feature	Berkeley DB	FSFS
Sensitivity to interruptions	very; crashes and permission problems can leave the database "wedged", requiring journaled recovery procedures.	quite insensitive.
Usable from a read-only mount	no	yes
Platform-independent storage	no	yes
Usable over network filesystems	no	yes
Repository size	slightly larger	slightly smaller
Scalability: number of revision trees	database; no problems	some older native filesystems don't scale well with thousands of entries in a single directory.
Scalability: directories with many files	slower	faster
Speed: checking out latest code	faster	slower
Speed: large commits	slower, but work is spread throughout commit	faster, but finalization delay may cause client timeouts
Group permissions handling	sensitive to user umask problems; best if accessed by only one user.	works around umask problems
Code maturity	in use since 2001	in use since 2004

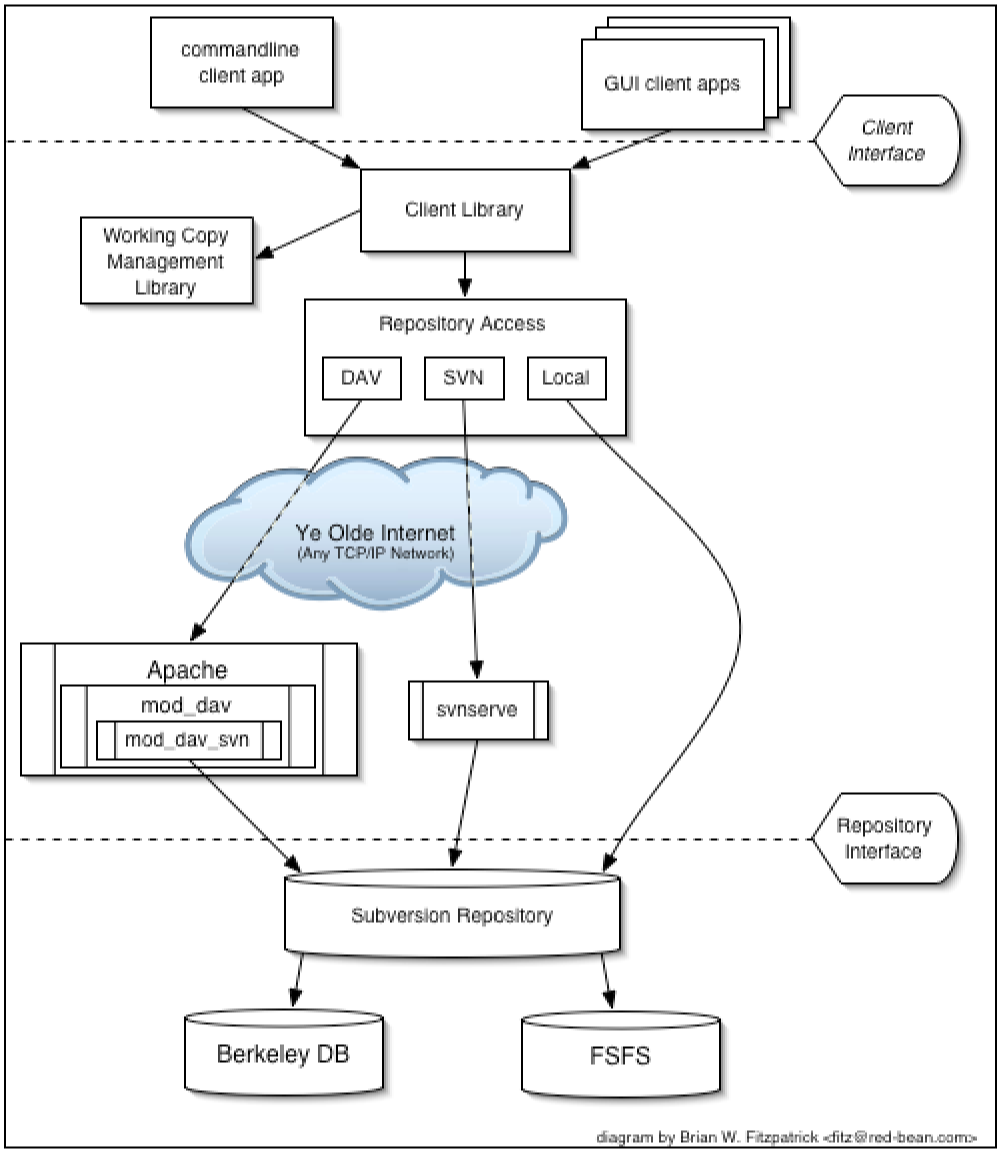
Repository Access Methods

Architecture Overview

At any given time, these processes may require read and write access to your repository:

regular system users using a Subversion client (as themselves) to access the repository directly via `file:///` URLs;

- regular system users connecting to SSH-spawned private **svnserve** processes (running as themselves) which access the repository;
- an **svnserve** process—either a daemon or one launched by **inetd**—running as a particular fixed user;
- an Apache **httpd** process, running as a particular fixed user.



svnserve

The **svnserve** program is a lightweight server, capable of speaking to clients over TCP/IP using a custom, stateful protocol. Clients contact an **svnserve** server by using URLs that begin with the `svn://` or `svn+ssh://` schema.

Apache 2

Via a custom module, **httpd** makes Subversion repositories available to clients via the

WebDAV/DeltaV protocol. The result is a standardized, robust system that is conveniently packaged as part of the Apache 2.0 software, is supported by numerous operating systems and third-party products, and doesn't require network administrators to open up yet another custom port. While an Apache-Subversion server has more features than **svnserve**, it's also a bit more difficult to set up.

That is Apache 2, by the way. Not 1. If you are one of the many people still running 1 and cannot chanage, consider runnning a copy of 2 as well on a different ports for your Subvresion & WebDAV needs.

Creating & Checking Out a Repository

`svn import [PATH] URL`

Recursively commit a copy of *PATH* to *URL*. If *PATH* is omitted "." is assumed. Parent directories are created in the repository as necessary.

Developer Foo creates a repository from the files `shape.php`, `square.php`, `junk.php`.

```
air:~/start foo$ ls
demo
air:~/start foo$ ls demo
junk.php        shape.php       square.php
air:~/start foo$ cat demo/junk.php
<?php
class Junk {
        public   $junk   ;
}

air:~/start foo$ cat demo/shape.php
<?php
class Shape {
        public   $height   ;
        public   $width   ;
}

air:~/start foo$ cat demo/square.php
<?php
class Square extends Shape {

}

air:~/start foo$ svn import http://svn.server.com/projects -m "Import"
Adding         demo
Adding         demo/shape.php
Adding         demo/square.php
Adding         demo/junk.php

Committed revision 1.
```

After importing data, note that the original tree is *not* under version control. To start working, you still need to **svn checkout** a fresh working copy of the tree.

svn checkout URL[@REV]... [PATH]

Check out a working copy from a repository. If *PATH* is omitted, the basename of the URL will be used as the destination. If multiple URLs are given each will be checked out into a subdirectory of PATH, with the name of the subdirectory being the basename of the URL.

Developer Foo checks out and shows files pulled down via a gui. There is nothing special about this gui in particular, and serveral are available for each OS. We are just demonstrating that there are other methods to access the repository other than via the command line.

Developer Bar checks out and shows files pulled down via command line.

```
air:~ bar$ cd Documents/
air:~ bar$ ls

air:~ bar$ svn checkout http://svn.server.com/repo/demo
A    demo/shape.php
A    demo/square.php
A    demo/junk.php
Checked out revision 1.
air:~ bar$ ls
demo
air:~ bar$ cd demo; ls
junk.php        shape.php       square.php
```

Changes & Updating

svn commit [PATH...]

Send changes from your working copy to the repository. If you do not supply a log message with your commit by using either the `--file` or `--message` switch, **svn** will launch your editor for you to compose a commit message.

Developer Bar adds area() method to `square.php` and commmits changes.

```
air:~/demo bar$ vi square.php
air:~/demo bar$ cat square.php
<?php
class Square extends Shape {
    function area() {
        return $this->height * $this->width;    }
}

air:~/demo bar$ svn commit -m "Added area() method to Square Class"
Sending        square.php
Transmitting file data .
Committed revision 2.
```

svn update [PATH...]

Brings changes from the repository into your working copy. If no revision given, it brings your working copy up-to-date with the HEAD revision. Otherwise, it synchronizes the working copy to the revision given by the `--revision` switch.

Developer Foo updates & shows changed file.

```
air:~/start/trunk foo$ cd ~/demo/
air:~/demo foo$ ls
junk.php        shape.php       square.php
air:~/demo foo$ cat square.php
<?php
class Square extends Shape {

}

air:~/demo foo$ svn update
U  square.php
Updated to revision 3.
air:~/demo foo$ cat square.php
<?php
class Square extends Shape {
    function area() {
        return $this->height * $this->width;
    }

}
```

Adding & Moving

svn add PATH...

Add files, directories, or symbolic links to your working copy and schedule them for addition to the repository. They will be uploaded and added to the repository on your next commit. If you add something and change your mind before committing, you can unschedule the addition using **svn revert**.

Developer Foo adds circle.php and commits.

```
air:~/demo foo$ vi circle.php
air:~/demo foo$ ls
junk.php        shape.php       square.php
air:~/demo foo$ vi circle.php
air:~/demo foo$ cat circle.php
<?php
class Circle extends Shape {
        public   $diameter   ;

    function radius() {
        return $this->diameter / 2 ;
    }
}

air:~/demo foo$ ls
circle.php      junk.php        shape.php       square.php
air:~/demo foo$ svn add circle.php
A         circle.php
air:~/demo foo$ svn commit -m "Added a Circle Class"
Adding         circle.php
Transmitting file data .
Committed revision 3.
```

svn delete PATH...
svn delete URL...

Items specified by *PATH* are scheduled for deletion upon the next commit. Files (and directories that have not been committed) are immediately removed from the working copy. The command will not remove any unversioned or modified items; use the `--force` switch to override this behavior.

Items specified by URL are deleted from the repository via an immediate commit. Multiple URLs are committed atomically.

Alternate Names:

- **svn del**
- **svn remove**
- **svn rm**

Developer Foo removes `junk.php` and commits.

```
air:~/demo foo$ ls
circle.php      junk.php        shape.php       square.php
air:~/demo foo$ svn delete junk.php
D         junk.php
air:~/demo foo$ ls
circle.php      shape.php       square.php
air:~/demo foo$ svn commit -m "Junk was thrown away"
Deleting       junk.php
```

```
Committed revision 4.
```

svn move SRC DST

This command moves a file or directory in your working copy or in the repository

Alternate Names:

- **svn mv**
- **svn rename**
- **svn ren**

This command is equivalent to an **svn copy** followed by **svn delete.**

Developer Foo renames filename to `rectangle.php`, changes Square class to Rectangle & commits.

```
air:~/demo foo$ svn move square.php rectangle.php
A         rectangle.php
D         square.php
air:~/demo foo$ ls
circle.php      rectangle.php   shape.php
air:~/demo foo$ vi rectangle.php
air:~/demo foo$ cat rectangle.php
<?php
class Rectangle extends Shape {
    function area() {
        return $this->height * $this->width;
    }

}

air:~/demo foo$ svn commit -m "Moved Square class to Rectangle"
Adding         rectangle.php
Deleting       square.php
Transmitting file data .
Committed revision 5.
```

Works for directories as well as files.

Developer Bar updates, then moves all files to a new, top level `/lib` folder & does not commit.

```
air:~/demo bar$ svn update
D    square.php
A    circle.php
A    rectangle.php
D    junk.php
Updated to revision 5.
air:~/demo bar$ ls
circle.php      rectangle.php   shape.php
air:~/demo bar$ mkdir lib
air:~/demo bar$ svn add lib
A         lib
air:~/demo bar$ svn move circle.php lib
```

```
A         lib/circle.php
D         circle.php
air:~/demo bar$ svn move rectangle.php lib
A         lib/rectangle.php
D         rectangle.php
air:~/demo bar$ svn move shape.php lib
A         lib/shape.php
D         shape.php
air:~/demo bar$ ls
lib
air:~/demo bar$ ls lib
circle.php      rectangle.php   shape.php
```

Logs

svn status [PATH...]

Print the status of working copy files and directories. With no arguments, it prints only locally modified items (no repository access). With `--show-updates`, add working revision and server out-of-date information. With `--verbose`, print full revision information on every item.

Developer Bar uses status to view recent changes.

```
air:~/demo bar$ svn status
D      shape.php
A      lib
A  +   lib/shape.php
A  +   lib/circle.php
A  +   lib/rectangle.php
D      circle.php
D      rectangle.php
```

svn log [PATH]

The default target is the path of your current directory. If no arguments are supplied, **svn log** shows the log messages for all files and directories inside of (and including) the current working directory of your working copy. You can refine the results by specifying a path, one or more revisions, or any combination of the two.

Developer Bar uses log to view recent comments, updates and shows changed file.

```
air:~/demo bar$ svn log
```

```
air:~/demo bar$ svn log
------------------------------------------------------------------------
r6 | bar | 2005-08-23 02:00:05 -0400 (Tue, 23 Aug 2005) | 1 line

moved all to /lib
------------------------------------------------------------------------
r5 | foo | 2005-08-23 01:54:16 -0400 (Tue, 23 Aug 2005) | 1 line

Moved Square class to Rectangle
------------------------------------------------------------------------
r4 | foo | 2005-08-23 01:52:34 -0400 (Tue, 23 Aug 2005) | 1 line

Junk was thrown away
------------------------------------------------------------------------
r3 | foo | 2005-08-23 01:51:38 -0400 (Tue, 23 Aug 2005) | 1 line

Added a Circle Class
------------------------------------------------------------------------
r2 | bar | 2005-08-23 01:47:23 -0400 (Tue, 23 Aug 2005) | 1 line

Added area() method to Square Class
------------------------------------------------------------------------
r1 | foo | 2005-08-23 01:37:20 -0400 (Tue, 23 Aug 2005) | 2 lines

Import
```

svn diff [-r N[:M]] [TARGET[@REV]...] svn diff [-r N[:M]] --old OLD-TGT[@OLDREV] [--new NEW-TGT[@NEWREV]] [PATH...] svn diff OLD-URL[@OLDREV] NEW-URL[@NEWREV]

Display the differences between two paths. The three different ways you can use **svn diff** are:

svn diff [-r N[:M]] [--old OLD-TGT] [--new NEW-TGT] [PATH...] displays the differences between *OLD-TGT* and *NEW-TGT*. If *PATH*s are given, they are treated as relative to *OLD-TGT* and *NEW-TGT* and the output is restricted to differences in only those paths. *OLD-TGT* and *NEW-TGT* may be working copy paths or *URL*[*@REV*]. *OLD-TGT* defaults to the current working directory and *NEW-TGT* defaults to *OLD-TGT*. *N* defaults to BASE or, if *OLD-TGT* is a URL, to HEAD. *M* defaults to the current working version or, if *NEW-TGT* is a URL, to HEAD. **svn diff -r N** sets the revision of *OLD-TGT* to *N*, **svn diff -r N:M** also sets the revision of *NEW-TGT* to *M*.

Developer Bar demonstrates some uses of diff...

```
Too much output to display here...
```

Developer Bar commits all changes.

```
air:~/demo bar$ svn commit -m "moved all to /lib"
Deleting       circle.php
Adding         lib
Adding         lib/circle.php
Adding         lib/rectangle.php
Adding         lib/shape.php
Deleting       rectangle.php
Deleting       shape.php
```

```
Committed revision 6.
```

Branching & Merging

svn copy SRC DST

Copy a file in a working copy or in the repository. *SRC* and *DST* can each be either a working copy (WC) path or URL:

WC -> WC
    Copy and schedule an item for addition (with history).
WC -> URL
    Immediately commit a copy of WC to URL.
URL -> WC
    Check out URL into WC, and schedule it for addition.
URL -> URL
    Complete server-side copy. This is usually used to branch and tag.

Cheap copies

Subversion's repository has a special design. When you copy a directory, you don't need to worry about the repository growing huge—Subversion doesn't actually duplicate any data. Instead, it creates a new directory entry that points to an *existing* tree. If you're a Unix user, this is the same concept as a hard-link. From there, the copy is said to be "lazy". That is, if you commit a change to one file within the copied directory, then only that file changes—the rest of the files continue to exist as links to the original files in the original directory.

Developer Foo updates (reinforce good workflow), then copies and switches to an "experimental" branch. Then does some refactoring by moving area() method up to `shape.php` and commits the changes.

```
air:~/demo foo$ svn update
D  circle.php
D  rectangle.php
A  lib
A  lib/shape.php
A  lib/circle.php
A  lib/rectangle.php
D  shape.php
Updated to revision 6.
```

Developer Foo creates an "experimental" branch by copying the current main trunk.

```
air:~/demo foo$ svn copy -m 'branching' \
   http://svn.server.com/repo/demo \
   http://svn.server.com/repo/foo_branch
Committed revision 7.
```

Developer Foo checks out the newly created branch.

```
air:~/branch foo$ cd ..
air:~ foo$ svn co http://svn.server.com/repo/foo_branch
A  foo_branch/lib
A  foo_branch/lib/shape.php
A  foo_branch/lib/circle.php
A  foo_branch/lib/rectangle.php
Checked out revision 7.
```

Developer Foo does some refactoring by moving area() method up to `shape.php` and commits the changes.

```
air:~ foo$ cd foo_branch/lib
air:~/foo_branch/lib foo$ vi shape.php
air:~/foo_branch/lib foo$ vi rectangle.php
air:~/foo_branch/lib foo$ vi circle.php
air:~/foo_branch/lib foo$ vi triangle.php
air:~/foo_branch/lib foo$ cat shape.php
<?php
class Shape {
        public  $height  ;
        public  $width   ;

        function area() {
                return $this->height * $this->width;
        }
}

air:~/foo_branch/lib foo$ cat rectangle.php
<?php
class Rectangle extends Shape {

}

air:~/foo_branch/lib foo$ cat circle.php
<?php
class Circle extends Shape {
        public $diameter ;

        function radius() {
                return $this->diameter / 2 ;
        }

        function area() {
                return pi()*pow($this->radius(),2);
        }
}

air:~/foo_branch/lib foo$ cat triangle.php
<?php
class Triangle extends Shape {
        function area() {
                return parent::area()/2 ;
        }
}

air:~/foo_branch/lib foo$ svn commit -m 'moved area up to Shape class'
```

```
Sending        lib/circle.php
Sending        lib/rectangle.php
Sending        lib/shape.php
Adding         lib/triangle.php
Transmitting file data ...
Committed revision 8.
```

Developer Bar uses status to show no changes after commit to branch. Note the revision number is higher, but no files change.

```
air:~/demo bar$ svn update
At revision 8.
```

svn merge sourceURL1[@N] sourceURL2[@M] [WCPATH]
svn merge sourceWCPATH1@N sourceWCPATH2@M [WCPATH]
svn merge -r N:M SOURCE[@REV] [WCPATH]

In the first form, the source URLs are specified at revisions *N* and *M*. These are the two sources to be compared. The revisions default to HEAD if omitted.

In the second form, *SOURCE* can be a URL or working copy item, in which case the corresponding URL is used. This URL, at revisions *N* and *M*, defines the two sources to be compared.

WCPATH is the working copy path that will receive the changes. If *WCPATH* is omitted, a default value of "." is assumed, unless the sources have identical basenames that match a file within ".": in which case, the differences will be applied to that file.

Unlike **svn diff**, the merge command takes the ancestry of a file into consideration when performing a merge operation. This is very important when you're merging changes from one branch into another and you've renamed a file on one branch but not the other.

Developer Foo merges changes from "experimental" to main trunk.

```
air:~/foo_branch/lib foo$ cd ../../demo/lib
air:~/demo foo$ svn merge --dry-run -r 6:HEAD
http://svn.server.com/repo/foo_branch
U  lib/shape.php
U  lib/circle.php
U  lib/rectangle.php
A  lib/triangle.php
air:~/demo foo$ svn status
air:~/demo foo$ svn merge  -r 6:HEAD  http://svn.server.com/repo/foo_branch
U  lib/shape.php
U  lib/circle.php
U  lib/rectangle.php
A  lib/triangle.php
air:~/demo foo$ svn status
M      lib/shape.php
M      lib/circle.php
M      lib/rectangle.php
A      lib/triangle.php
```

```
air:~/demo foo$ svn commit -m 'mergeed experimental branch back'
Sending        lib/circle.php
Sending        lib/rectangle.php
Sending        lib/shape.php
Adding         lib/triangle.php
Transmitting file data ...
Committed revision 9.
```

Developer Bar updates.

```
air:~/demo bar$ svn update
U    lib/shape.php
U    lib/circle.php
U    lib/rectangle.php
A    lib/triangle.php
Updated to revision 9.
```

Since copies are cheap, it is easy to support multiple developers with multiple branches, just merging into the main trunk whatever particluar revision you want.

Conflicts & Recovery

Developer Bar makes a change and commits.

```
air:~/demo/lib bar$ vi circle.php
air:~/demo/lib bar$ cat circle.php
<?php
class Circle extends Shape {
        public $diameter ;

        function radius() {
                return $this->diameter / 2 ;
        }

        function area() {
                return pi()*pow($this->radius(),2);
        }

        function circumference() {
                return pi() * $this->diameter ;
        }
}
air:~/demo/lib bar$ svn commit -m 'added circumference() to circle class'
Sending        lib/circle.php
Transmitting file data ...
Committed revision 10.
```

Developer Foo neglects to update before making changes to the same file and commits.

```
air:~/demo/lib foo$ vi circle.php
air:~/demo/lib foo$ cat circle.php
<?php
class Circle extends Shape {
        public $diameter ;

        function radius() {
                return $this->diameter / 2 ;
        }

        function area() {
                return pi()*pow($this->radius(),2);
        }

        function circumference() {
                return $this->diameter * pi() ;
        }
}
air:~/demo foo$ svn commit -m 'added circumference() to circle class'
Sending        lib/circle.php
Transmitting file data ...
Committed revision 11.
```

Merging Conflicts by Hand

`svn resolved PATH...`

Remove "conflicted" state on working copy files or directories. This routine does not semantically resolve conflict markers; it merely removes conflict-related artifact files and allows PATH to be committed again; that is, it tells Subversion that the conflicts have been "resolved".

The strings of less-than signs, equal signs, and greater-than signs are conflict markers, and are not part of the actual data in conflict. You generally want to ensure that those are removed from the file before your next commit. The text between the first two sets of markers is composed of the changes you made in the conflicting area. The text between the second and third sets of conflict markers is the text from the conflicting commit.

```
air:~/demo/lib bar$ cat circle.php
....
air:~/demo/lib bar$ svn resolved circle.php
air:~/demo/lib bar$ svn commit -m "Go ahead and use my cirle, discarding bar's
edits."
```

Copying a File Onto Your Working File

f you get a conflict and decide that you want to throw out your changes, you can merely copy one of the temporary files created by Subversion over the file in your working copy.

```
air:~/demo/lib bar$ svn update
C  circle.php
Updated to revision 12.
```

```
air:~/demo/lib bar$ ls circle.*
circle.php  circle.php.mine  circle.php.r10  circle.php.r11
air:~/demo/lib bar$ cp circle.php.r2 circle.php
air:~/demo/lib bar$ svn resolved circle.php
```

Punting: Using svn revert

`svn revert PATH...`

Reverts any local changes to a file or directory and resolves any conflicted states. `svn revert` will not only revert the contents of an item in your working copy, but also any property changes. Finally, you can use it to undo any scheduling operations that you may have done (e.g. files scheduled for addition or deletion can be "unscheduled").

If you get a conflict, and upon examination decide that you want to throw out your changes and start your edits again, just revert your changes.

```
air:~/demo/lib bar$ svn revert circle.php
Reverted 'circle.php'
air:~/demo/lib bar$ ls circle.*
circle.php
```

Good Workflow

- Update your working copy: **`svn update`**
- Make changes: **`svn add|delete|copy|move`**
- Examine Your changes: **`svn status|diff|revert`**
- Merge others' changes: **`svn merge|resolved`**
- Commit your changes: **`svn commit`**