

The COOPY Toolbox

0.6.4

Generated by Doxygen 1.8.1.2

Thu Sep 13 2012 15:43:55

Contents

1	COOPY Guide	1
1.1	Comparing tables	1
1.2	Coopy GUI tutorials	1
1.3	Version control	1
1.4	Command line tools reference	2
1.5	Formats	2
2	Specification of the highlighter diff format	2
2.1	Single sheet case, with fixed columns	2
2.2	Action column	3
2.3	Data columns	3
2.4	Compound values	4
2.5	String representation of values	4
2.6	Multiple sheet case, with fixed columns	5
2.7	Adding columns	6
2.8	Removing columns	7
2.9	Row and column order?	8
2.10	Implementations	8
3	Pulling in a repository with Coopy	8
4	Creating a new repository for Coopy	12
5	Pushing out a repository with Coopy	16
6	Forking a repository for Coopy	18
7	Resolving conflicts with Coopy	22
8	Comparing tables using ssdiff	24
8.1	Presenting differences flexibly	25
8.2	Merging differences flexibly	26
9	Applying highlighter diffs using sspatch	27
9.1	How to recognize a highlighter diff	27
9.2	Applying all changes from a highlighter diff	28
9.3	Applying just some changes from a highlighter diff	28
9.4	Converting to another format	28
9.5	Accessing diffs programmatically	29

10 Merging tables with diverging IDs	30
10.1 An example	30
10.2 Divergence	31
10.3 Convergence	34
10.4 Getting the material used in this example	37
11 Using ssmerge with git	37
11.1 Manage CSV files in git with COOPY	37
11.2 Worked CSV example	38
11.3 Troubleshooting COOPY with git	39
11.4 Useful commands for testing	40
11.5 Manage Sqlite databases in git with COOPY	40
11.6 Worked Sqlite example	40
12 ssdiff	43
12.1 Usage	43
12.2 Index	43
12.3 Option summary	43
12.4 Option details	44
12.5 Examples	45
12.5.1 Example 1	45
12.5.2 Example 2	45
12.5.3 Example 3	46
12.5.4 Example 4	46
12.6 Patch formats	46
12.7 Database/spreadsheet file formats	46
12.8 Version	49
13 sspatch	49
13.1 Usage	49
13.2 Index	49
13.3 Option summary	49
13.4 Option details	50
13.5 Examples	50
13.5.1 Example 1	50
13.5.2 Example 2	51
13.5.3 Example 3	51
13.5.4 Example 4	51

13.5.5 Example 5	51
13.5.6 Example 6	51
13.5.7 Example 7	51
13.5.8 Example 8	51
13.5.9 Example 9	51
13.5.10 Example 10	52
13.6 Patch formats	52
13.7 Database/spreadsheet file formats	52
13.8 Version	54
14 ssmerge	54
14.1 Usage	55
14.2 Index	55
14.3 Option summary	55
14.4 Option details	56
14.5 Examples	56
14.5.1 Example 1	57
14.5.2 Example 2	57
14.5.3 Example 3	57
14.5.4 Example 4	57
14.6 Patch formats	57
14.7 Database/spreadsheet file formats	58
14.8 Version	60
15 ssresolve	60
15.1 Usage	60
15.2 Index	60
15.3 Option summary	60
15.4 Option details	61
15.5 Examples	61
15.5.1 Example 1	61
15.5.2 Example 2	61
15.5.3 Example 3	61
15.5.4 Example 4	61
15.6 Patch formats	62
15.7 Database/spreadsheet file formats	62
15.8 Version	64

16 ssformat	64
16.1 Usage	64
16.2 Index	65
16.3 Option summary	65
16.4 Option details	65
16.5 Examples	66
16.5.1 Example 1	66
16.5.2 Example 2	66
16.5.3 Example 3	66
16.6 Patch formats	66
16.7 Database/spreadsheet file formats	66
16.8 Version	69
17 ssrediff	69
17.1 Usage	69
17.2 Index	69
17.3 Option summary	69
17.4 Option details	70
17.5 Examples	71
17.5.1 Example 1	71
17.5.2 Example 2	71
17.5.3 Example 3	71
17.5.4 Example 4	71
17.6 Patch formats	71
17.7 Database/spreadsheet file formats	72
17.8 Version	74
18 ssfossil	74
19 coopy	74
19.1 Usage	75
19.2 Index	75
19.3 Option summary	75
19.4 Option details	75
19.5 Examples	76
19.5.1 Example 1	76
19.5.2 Example 2	76
19.5.3 Example 3	76

19.5.4 Example 4	76
19.6 Version	76
20 Related projects	77
21 What are all these funny files in a Coopy directory?	77
22 Diff format	77
23 TDIFF tabular diff format	77
23.1 General structure	78
23.2 Comment blocks	78
23.3 Control blocks	78
23.4 Header control block	78
23.5 Hunk	79
23.6 Diff line	79
23.7 Column line	79
23.8 Keys versus identity	80
23.9 Appendix: Quoting	80
23.10Appendix: Grammar	80
23.11Appendix: Complete examples	81
23.12Appendix: Version history	82
24 CSV DTBL diff format	82
24.1 General structure	83
24.2 First row	83
24.3 Config rows	84
24.4 Operation rows	84
24.5 table name	84
24.6 column name	84
24.7 column move	85
24.8 column insert	85
24.9 column delete	85
24.10column rename	85
24.11link name	85
24.12link act	86
24.13row update	87
24.14row select	87
24.15row insert	87

24.16row delete	87
24.17CSV details	88
24.18Representing NULLs and other types	88
24.19Pending issues	88
25 Supported table formats	88
25.1 Specifying sources	89

1 COOPY Guide

The COOPY project contains programs and libraries to help you:

- Describe the difference between two tables (see [ssdiff](#)).
- Apply the difference between two tables to a table (see [sspatch](#)).
- Merge tables that came from the same source but have been modified independently (see [ssmerge](#)). Two- and three-way merges supported.
- Maintain a repository of tables with multiple active contributors (see [ssfossil](#) and the [coopy](#) program).
- Supported table formats include CSV, sqlite, mysql, Access (read-only), Excel, OpenOffice.

1.1 Comparing tables

- [Comparing tables using ssdiff](#)
- [Applying highlighter diffs using sspatch](#)
- [Merging tables with diverging IDs](#)

1.2 Coopy GUI tutorials

- [Pulling in a repository with Coopy](#)
- [Creating a new repository for Coopy](#)
- [Pushing out a repository with Coopy](#)
- [Forking a repository for Coopy](#)
- [Resolving conflicts with Coopy](#)
- [What are all these funny files in a Coopy directory?](#)

1.3 Version control

- [Using ssmerge with git](#)
- [ssfossil](#) reference

1.4 Command line tools reference

- [ssdiff](#)
- [sspatch](#)
- [ssmerge](#)
- [ssresolve](#)
- [ssformat](#)
- [ssrediff](#)
- [coopy](#)

1.5 Formats

- [Supported table formats.](#)
- [Diff format.](#)

And if COOPY isn't quite right for you?

- [Related projects.](#)

Code in the COOPY project is all free and open source (licensed under the GPL; LGPL port in the works). The [ssmerge](#) and [ssdiff](#) programs currently make use of [libcsv](#). The [ssfossil](#) program is a thin wrapper around [fossil](#). The [coopy](#) program uses [wxWidgets](#). Optionally, programs also make use of the spreadsheet library within [Gnumeric](#).

2 Specification of the highlighter diff format

The highlighter diff format is a format for expressing the difference between two spreadsheets.

A highlighter diff is designed to be represented in a spreadsheet format, using "common denominator" features of Excel, OpenOffice/LibreOffice, Gnumeric, etc. A highlighter diff can however be represented in any tabular format with ordered columns and rows. It is a simple format, easy to parse, yet expressive enough to capture an important set of possible differences. It is intended for spreadsheets containing data, rather than spreadsheets with formulae, charts, and the like. Highlighter diffs deal with cell content, not formatting.

2.1 Single sheet case, with fixed columns

Assume we are comparing two versions of a sheet/table, L and R (for Left/Local and Right/Remote). L is the "reference" sheet, and we describe the difference between it and R in terms of operations needed to change L into R. The difference is described in a "diff sheet" that looks like this:

@@	bridge	designer	length
	Brooklyn	J. A. Roebling	1595
+++	Manhattan	G. Lindenthal	1470
->	Williamsburg	D. Duck->L. L. Buck	1600
	Queensborough	Palmer & Hornbostel	1182
	Triborough	O. H. Ammann	1380,383
...
	Throgs Neck	O. H. Ammann	1800
	George Washington	O. H. Ammann	3500
---	Spamspan	S. Spamington	10000

A diff sheet consists of a single action column and a block of data columns, as follows.

2.2 Action column

The diff sheet has a column called the "action" column. The action column is the first column that does not contain integers; generally it will be exactly the first column. Here are cell values that can appear in this column:

@@	the header tag marks a header row, if one exists.
+++	the insert tag marks an added row (present in R, not present in L).
---	the delete tag marks a removed row (present in L, not present in R).
->	the update tag marks a row in which at least one cell changed. -->, --->, ----> etc. have the same meaning.
Blank	A blank string or NULL marks a row common to L and R.
...	the skip tag marks that rows common to L and R are being skipped.
!	The schema tag marks a change in the sheet structure.

2.3 Data columns

The columns after the action column are called the data columns. Each column in L and each column in R map onto a data column. If the diff contains no schema row ("!"), then L and R have the same columns, and the data columns map straightforwardly onto columns in L and R. Here is what the data columns contain, for each action tag:

@ @	cells contain column names.
+++	cells shown are from R, and not present in L.
---	cells shown are from L, and not present in R.
->	cells shown are common to both L and R. When L and R cells differ, a compound "Vl->Vr" cell is shown.
Blank	cells shown are common to both L and R.
...	skipping rows common to both L and R.

In added rows, the data columns are copied from R, and in deleted rows, the data columns are copied from R. For updates, data columns that are common to L and R are filled with that common value. Data columns that differ are filled with a "compound value".

2.4 Compound values

When a cell in L and R differs, we need to show both values (call them Vl and Vr) in a single cell without introducing ambiguity. We keep this as simple as possible, as follows. For data where the substring ">" will not occur, and there are no ambiguities in the string representation of values, compounds are simple:

```
compound(Vl,Vr) = string(Vl) + ">" + string(Vr)
```

So we can get (the string representation of) Vl and Vr by splitting the compound value at ">". But in general, ">" might occur in data. We optimize ease of interpretation of the diff (rather than ease of producing it), by requiring that the update action tag be chosen to avoid collision with the data (either per row or over the entire sheet, at the discretion of the implementor). So the rule becomes:

```
compound(Vl,Vr) = string(Vl) + action_tag + string(Vr)
```

So as before we can get (the string representation of) Vl and Vr by splitting the compound value at occurrence of the action_tag.

2.5 String representation of values

The string representations of Vl and Vr in compound values require some care, in the case of data that contains NULL values and for which NULLs and blank strings are distinct. If the diff contains any string ending in "NULL" then "NULL-quoting" is in effect, and any such string should be interpreted as follows:

STRING	VALUE
"NULL"	This string represents a NULL value
"_NULL"	This represents the string "NULL"
"__NULL"	This represents the string "_NULL"
...	...

When possible, NULL values may be represented in the diff directly through empty cells, rather than as strings.

2.6 Multiple sheet case, with fixed columns

Easy! Just the same as we've done so far, repeated, with appropriate sheet names. For example:

locations

@@	id	street	city
	1	305 Memorial Drive	Cambridge
	2	Big Crater	Moon
+++	3	10 Ten Street	Denver

org2loc

@@	org_id	loc_id
	1	2
	2	1
	3	1
	3	2
+++	5	3

organizations

@@	id	name
	1	Dracula Inc
	2	The Firm
	3	Omni Cooperative
	4	Nonexistence Unlimited
+++	5	Alice and Company

2.7 Adding columns

Highlighter diffs can represent changes to the sheet's structure, by adding an initial "schema" row (identified by having an exclamation mark in the first column). Columns to be added are identified with a "+++". For example:

!			+++	
@@	bridge	designer	quark	length
+	Brooklyn	J. A. Roebling	strange	1595
+++	Manhattan	G. Lindenthal	charm	1470
->	Williamsburg	D. Duck->L. L. Buck	up	1600
+	Queensborough	Palmer & Hornbostel	down	1182
+	Triborough	O. H. Ammann	strange	1380,383
+	Bronx Whitestone	O. H. Ammann	charm	2300
+	Throgs Neck	O. H. Ammann	hairy	1800
+	George Washington	O. H. Ammann	moody	3500
---	Spamspan	S. Spamington		10000

A new action called "+" has been added, to mark rows that have no change other than added cells.

2.8 Removing columns

Columns to be removed are identified with a "—" in the schema row. For example:

!			---	
@@	bridge	designer	quark	length
	Brooklyn	J. A. Roebling	strange	1595
---	Manhattan	G. Lindenthal	charm	1470
->	Williamsburg	L. L. Buck->D. Duck	up	1600
	Queensborough	Palmer & Hornbostel	down	1182
	Triborough	O. H. Ammann	strange	1380,383
...
	Throgs Neck	O. H. Ammann	hairy	1800
	George Washington	O. H. Ammann	moody	3500
+++	Spamspan	S. Spamington		10000

2.9 Row and column order?

This spec doesn't deal with order changes. Suggestions welcome!

2.10 Implementations

The COOPY toolbox can generate highlighter diffs, apply them as patches, and convert them to other patch formats (such as SQL and tdiff). See: [Applying highlighter diffs using sspatch](#). Using COOPY, highlighter diffs can be read in C++, Ruby (via wrappers), and Python (via wrappers). It would be great to see native implementations for reading and writing highlighter diffs in various languages and (crucially) spreadsheet plugins.

3 Pulling in a repository with Coopy

The **coopy** program is a graphical interface for "pulling" new or updated repositories onto one's computer, and for "pushing" modified repositories out to a server.

It is specialized for spreadsheets and databases. Advanced users who need maximum flexibility may wish to try to the [ssfoossil](#) command-line tool rather than the **coopy** program. It is fine to use both on the same repository.

When you start the **coopy** program, you see something like this:

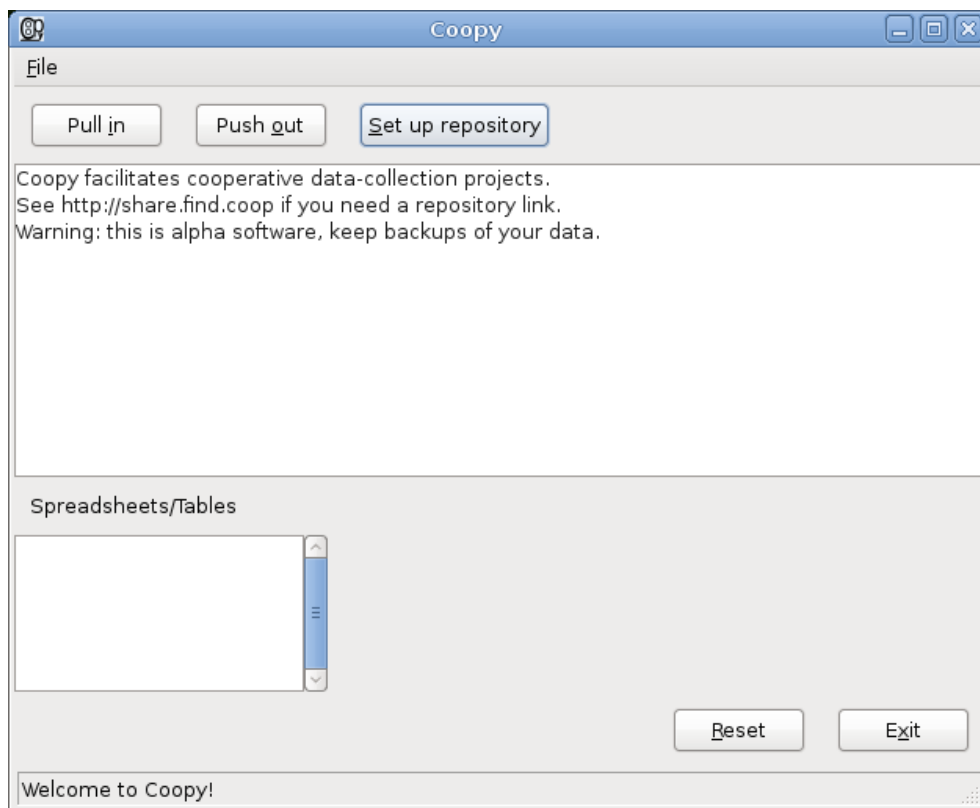


Figure 1: Here's Coopy!

Here are the main parts of the interface:

- Buttons!
- A big text box, called the "activity log". When we start working with Coopy, messages will roll past in this box. This is a user interface designed by a computer scientist, sorry...
- A box labeled "Spreadsheets/Tables". A list of available databases appears here. Just now, there are none. We need to set up a repository first.

To do anything useful with Coopy, we need to get a repository. Click on the "Pull in" button to pull data in to your computer from the web, and you'll see:

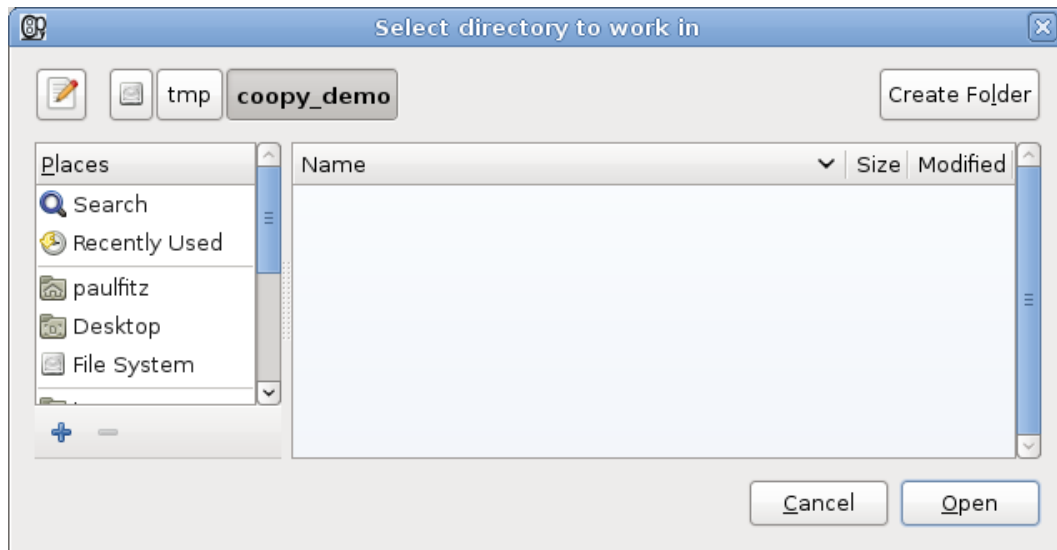


Figure 2: Select or create an empty directory.

What Coopy is looking for here is a fresh, clean, empty directory in which to store a bunch of repository files. It is a good idea to create a new folder for this purpose.

Once you've given Coopy an empty directory to fool around in, you're prompted for an address of a repository:

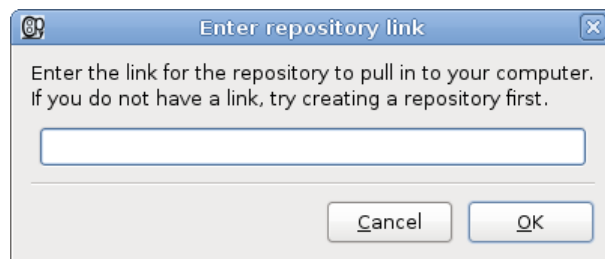


Figure 3: What to put here?

What the heck do we put here? Repositories live on the web, and have addresses like any webpage. We could make our own (see [Creating a new repository for Coopy](#)), but for now let's use one I've prepared earlier - this:

https://chiselapp.com/user/coopy/repository/coopy_demo/index

If you click on that link, you'll see something that has a little logo of a fossil. Don't bother poking around there yet, we'll come back to it later. For now, just paste in the link to Coopy:

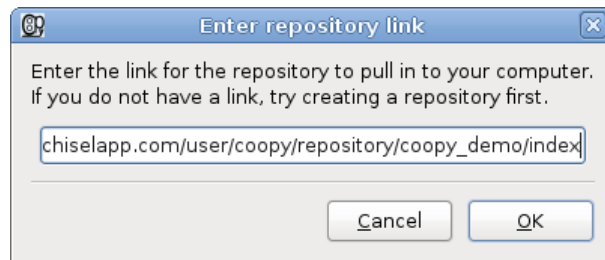


Figure 4: Fill in demo repository link.

Click OK, and Coopy chirps away merrily to itself, throwing messages into the activity log:

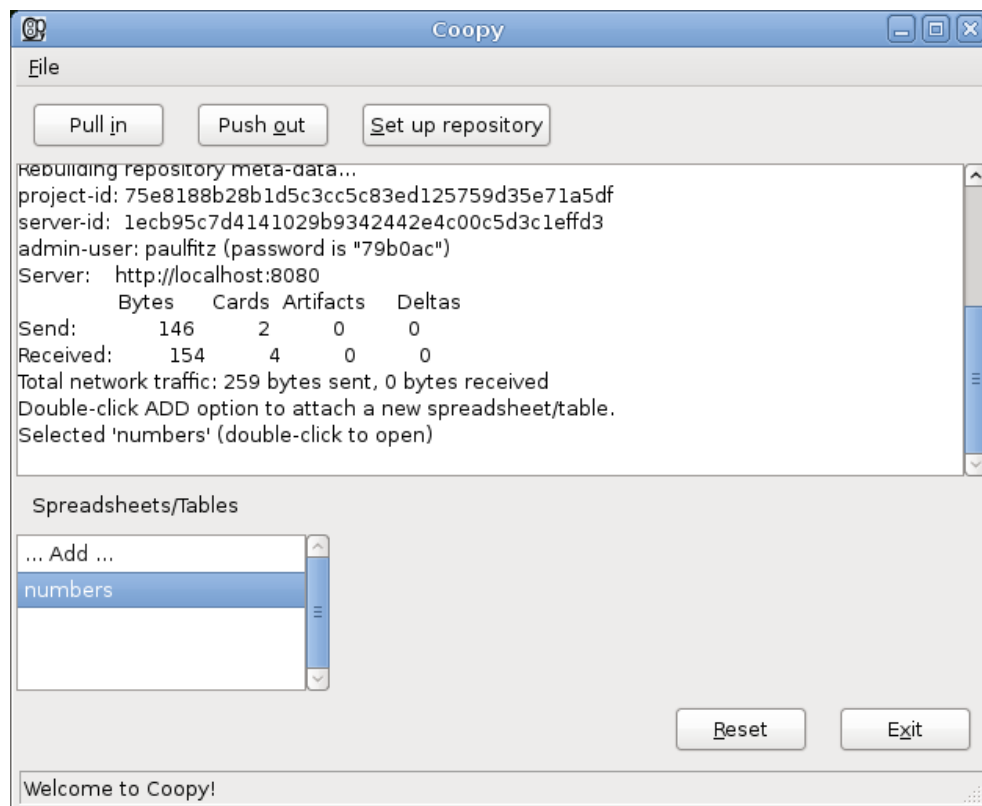


Figure 5: Our first repository has arrived!

No error messages, that's good. The "Spreadsheets/Tables" area has changed. It gives a list of databases in the repository. There is in fact just one, called "numbers". Double-click on it, and you're asked where you want to save the database, and in what format:

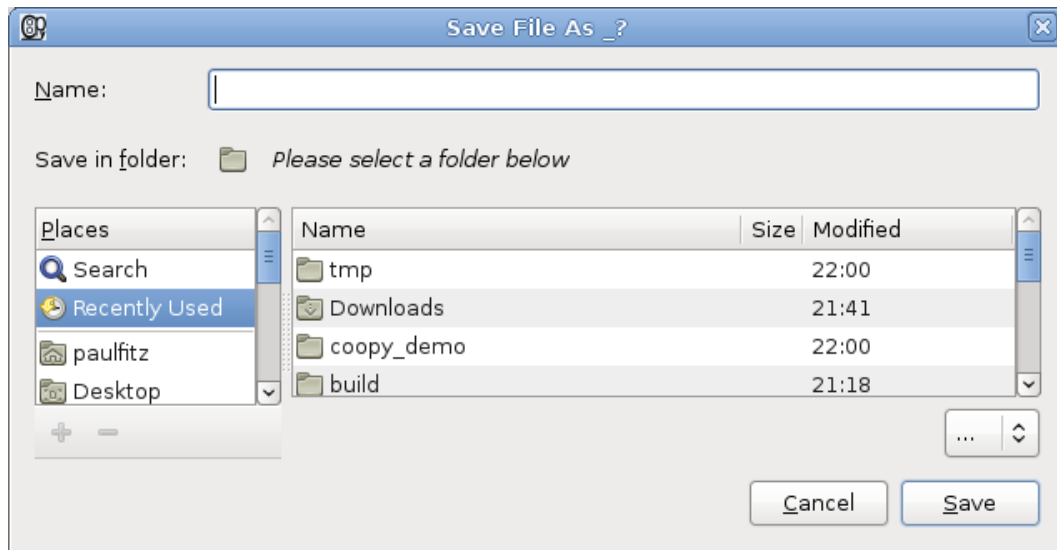


Figure 6: Save database as...

You can choose to save the database in a number of formats: principally Excel, Sqlite, or CSV (for single-table databases). The choice is yours. For now, save the database as an *.xls file, and it will open in whatever you use to view spreadsheets on your computer (if you have nothing, install OpenOffice, Gnumeric, or something along those lines).

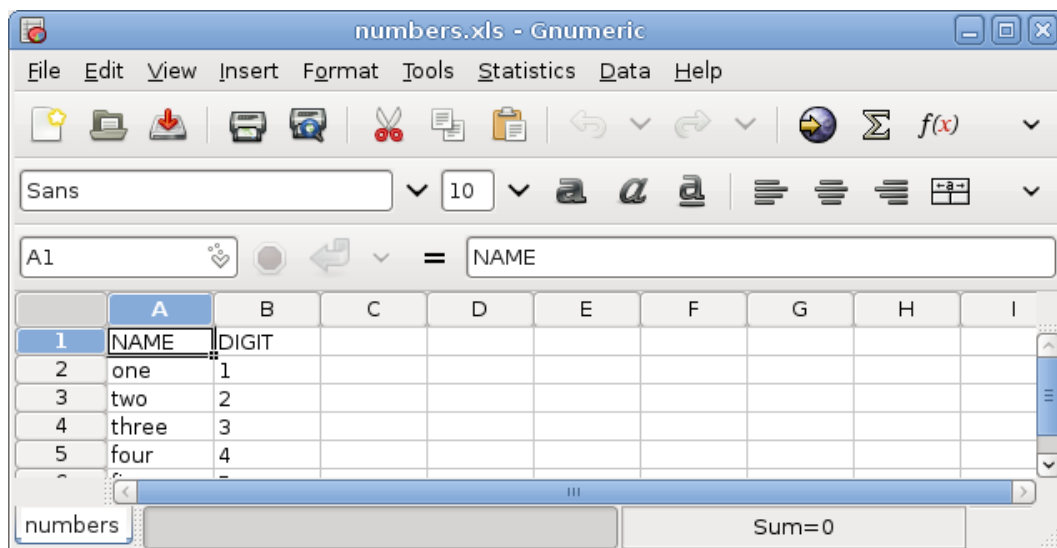


Figure 7: The numbers spreadsheet.

What happens when we start making changes in the spreadsheet? That's the topic of this tutorial: [Pushing out a repository with Coopy](#).

4 Creating a new repository for Coopy

A repository is a collection of files.

For Coopy, collections of files are managed by [fossil](#), a distributed version control system. What does this mean?

- A revision history is kept for all files in the collection.
- The repository can be "cloned" on another computer or another location, in such a way that changes to the repository can be passed on easily.
- Clones of the repository can be changed by different people, and then have those changes merged intelligently.

It is possible to set up a repository using the fossil or [ssfossil](#) program. Alternatively, you can find or create a fossil repository online. Here are known hosting services for fossil repositories:

- [chiselapp.com](#)
- [share.find.coop](#)

You can find this list by starting [coopy](#), then clicking on "Set up repository":

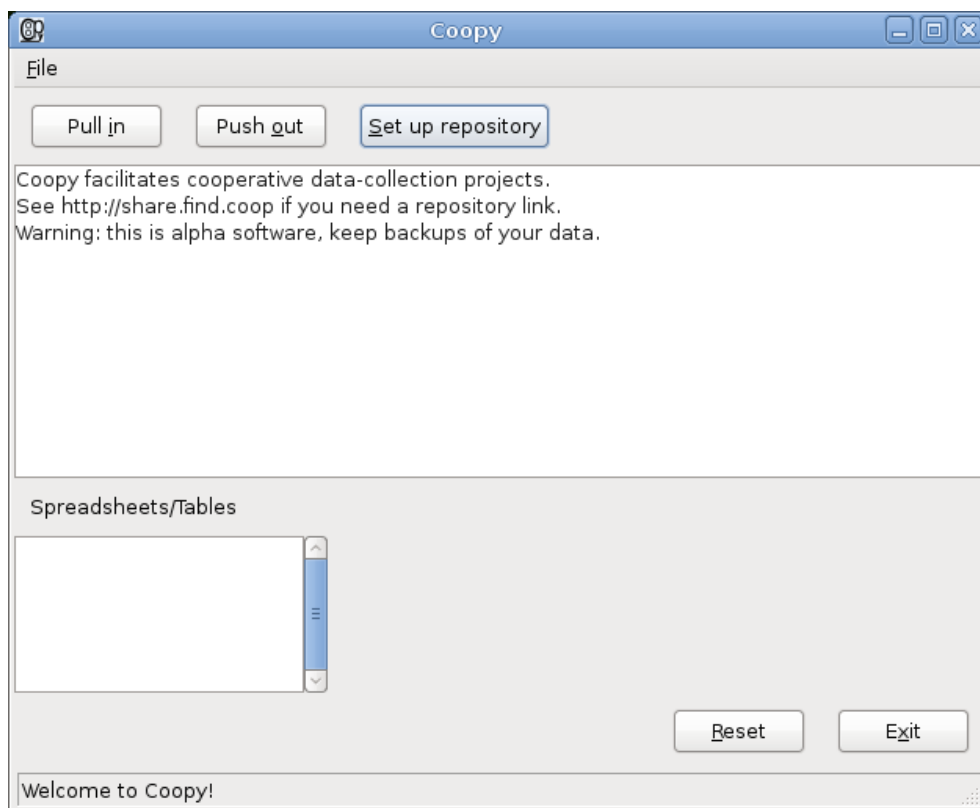


Figure 8: Click on: Set up repository

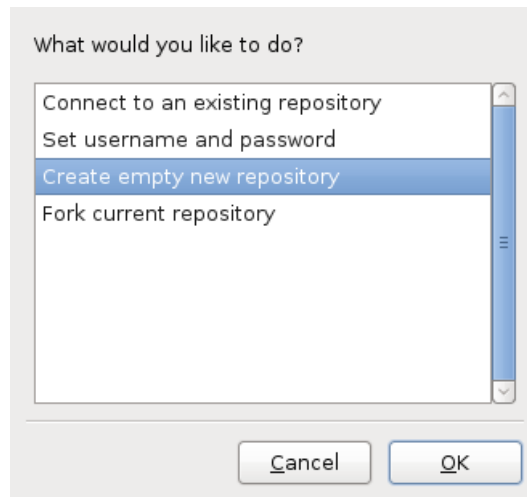


Figure 9: Select: Create new repository

When we choose the new repository option, we are offered a list of hosts:

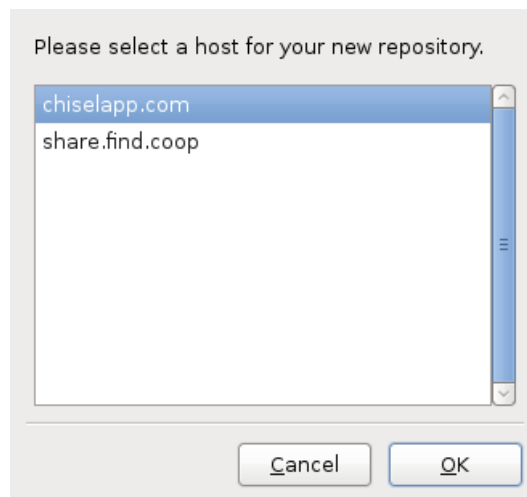


Figure 10: Select a host.

Choose a website to host your repository. You can also have your own website to host a fossil repository (see [self-hosting instructions](#)), you don't need to stick with this list. For now, let's go with [chiselapp.com](#).

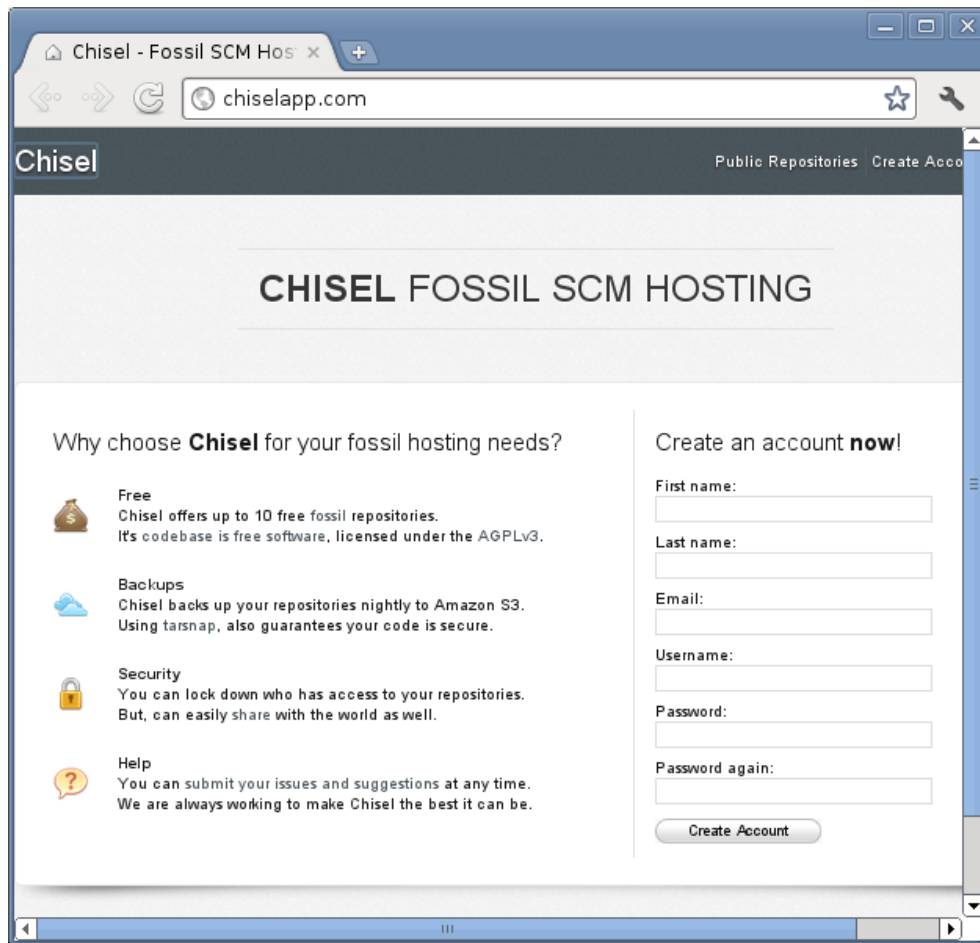


Figure 11: Chisel

After you register and log in, you'll see a dashboard that looks like this (except emptier):

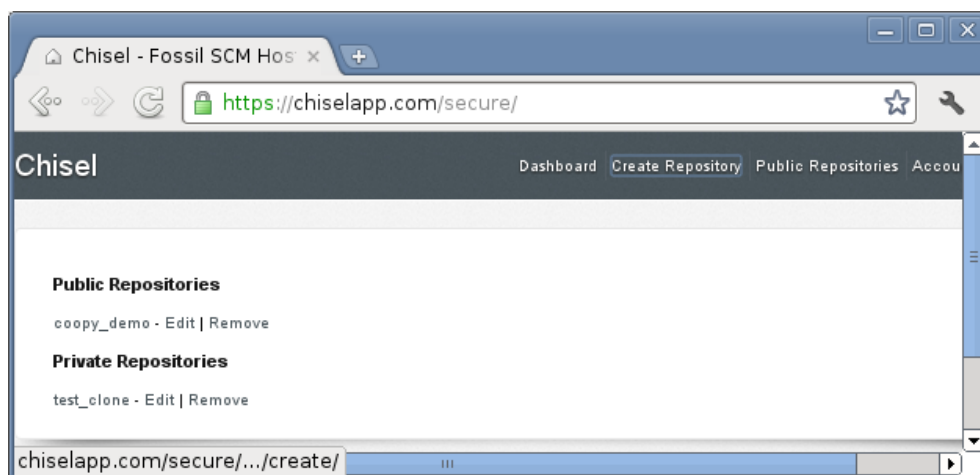
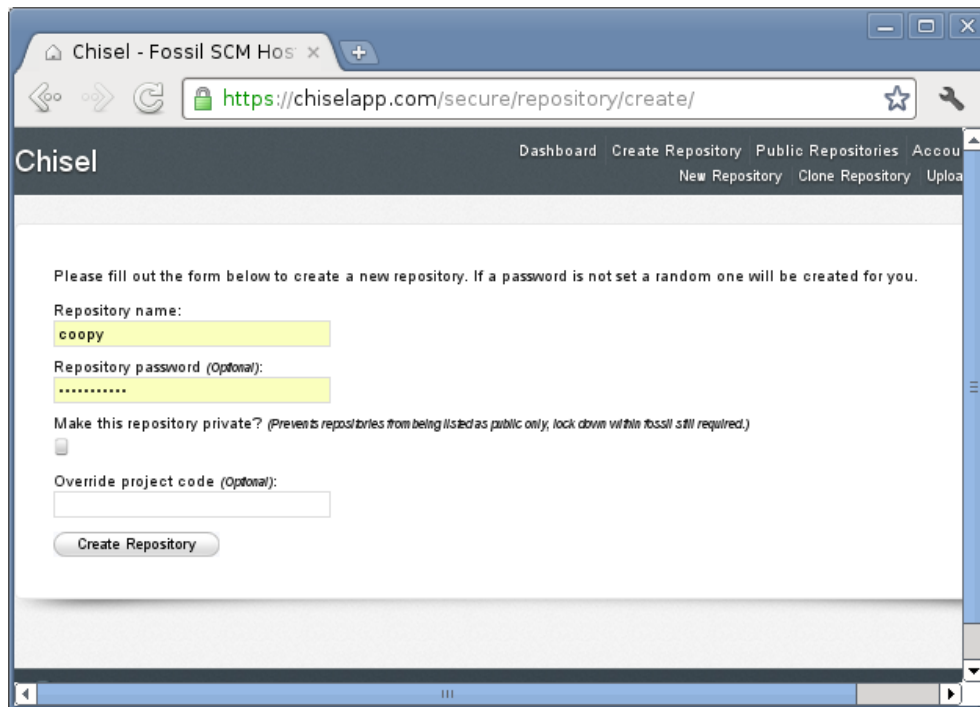


Figure 12: Chisel dashboard

To make a repository, click on "Create repository".



The screenshot shows a web browser window with the address bar displaying `https://chiselapp.com/secure/repository/create/`. The page title is "Chisel". The navigation bar includes links for "Dashboard", "Create Repository", "Public Repositories", and "Account". Below the navigation bar, there is a form to create a new repository. The form contains the following fields and options:

- A message: "Please fill out the form below to create a new repository. If a password is not set a random one will be created for you."
- "Repository name:" with a text input field containing "coopy".
- "Repository password (Optional):" with a text input field containing "*****".
- A checkbox labeled "Make this repository private? (Prevents repositories from being listed as public only; lock down within fossil still required.)" which is currently unchecked.
- "Override project code (Optional):" with a text input field.
- A "Create Repository" button.

Figure 13: Create repository

Set the repository up as you like. The "Override project code" field is not needed when setting up a fresh, empty repository. Once you've set up a repository, on the dashboard find a link to it, like the one shown in a box here:

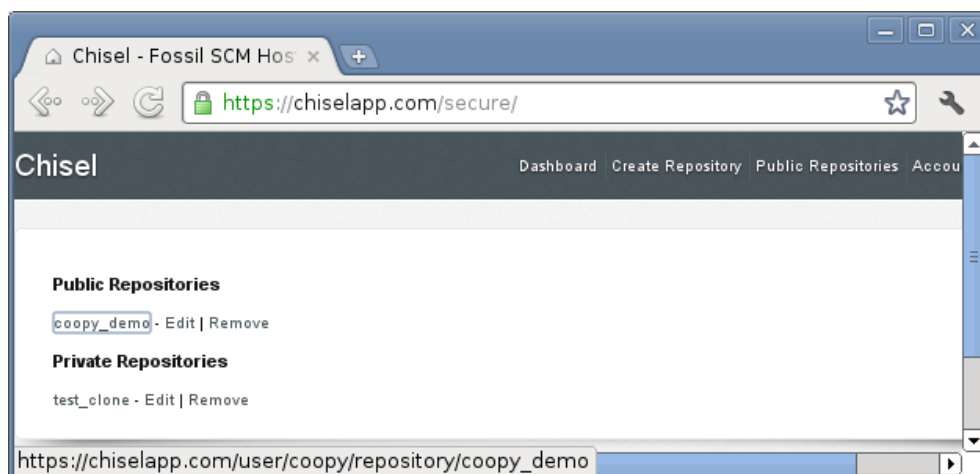


Figure 14: Find the repository link

When you click that link, you should see a page in a different style with a picture of a fossil. This is your repository, and the link shown in your browser is the "repository link" that Coopy needs.



Figure 15: Fossil

You could now follow the instructions in this tutorial: [Pulling in a repository with Coopy](#). Just replace the demo repository link with your own. You can also follow this tutorial: [Pushing out a repository with Coopy](#), using your Chiselapp username, and the password you chose for the repository.

5 Pushing out a repository with Coopy

Do this tutorial first: [Pulling in a repository with Coopy](#)

Now, change some number in your *.xls file, then save it.

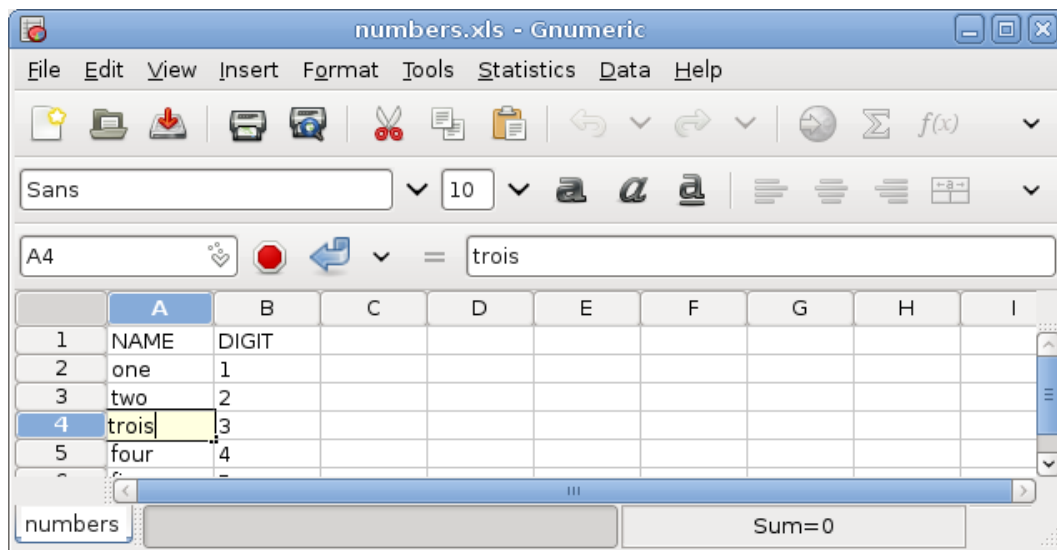


Figure 16: Making a change...

Now to send the updates to the repository, we click the "Push out" button. We're prompted to make a brief description of our changes (called a commit message). Write anything you like here. Typically this should be a brief message that will let a collaborator understand your change quickly.

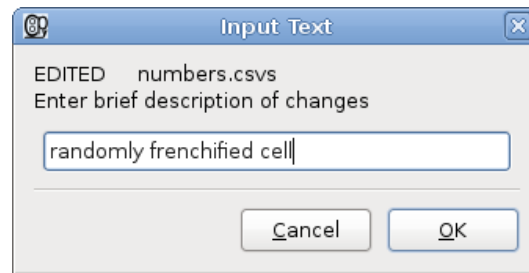


Figure 17: Add a commit message.

The next thing that is likely to happen is an access denied message. Repositories can be set up so that random folks can commit changes to them, but normally you need a username and password. Let's see how things work if you have a username/password (otherwise consider following: [Forking a repository for Coopy](#)).

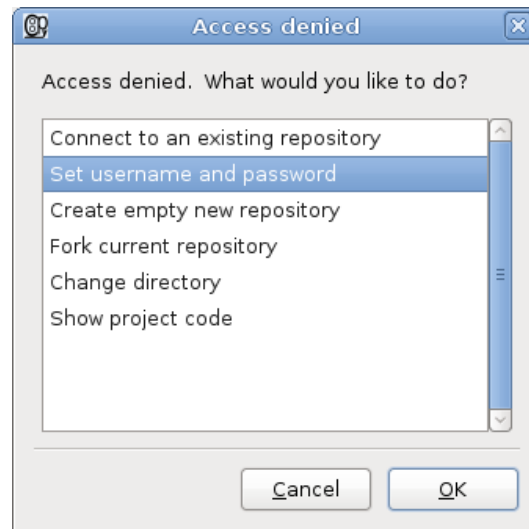


Figure 18: Access denied.

Fill out a username and password:

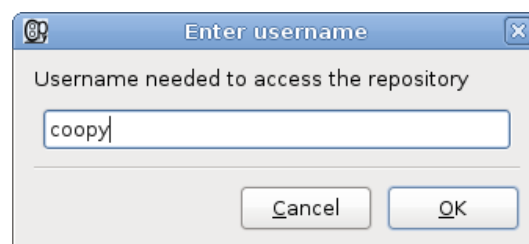


Figure 19: Username.

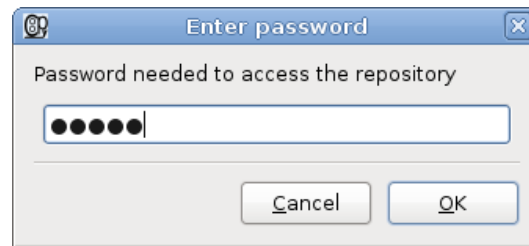


Figure 20: Password.

And we're done:

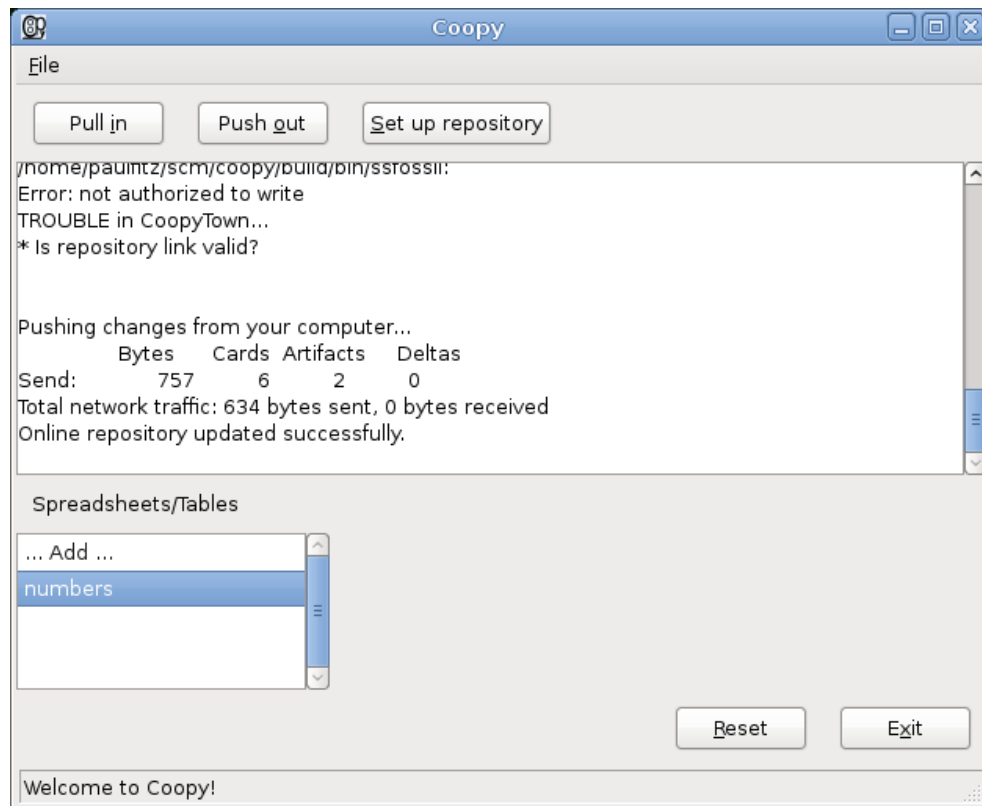


Figure 21: A successful push.

If you want to push out to a new repository, follow: [Forking a repository for Coopy](#).

6 Forking a repository for Coopy

Suppose you pulled in the demo repository in [Pulling in a repository with Coopy](#), then made modifications to it, but didn't want to (or couldn't) push those modifications back to the original repository.

One option is to fork the repository and store it elsewhere.

The procedure is just like [Creating a new repository for Coopy](#), except with one wrinkle: when making your new repos-

itory, you need to give it a "project code" that is identical to the one you pulled in. You can find that code by clicking on the "Set up repository" link:

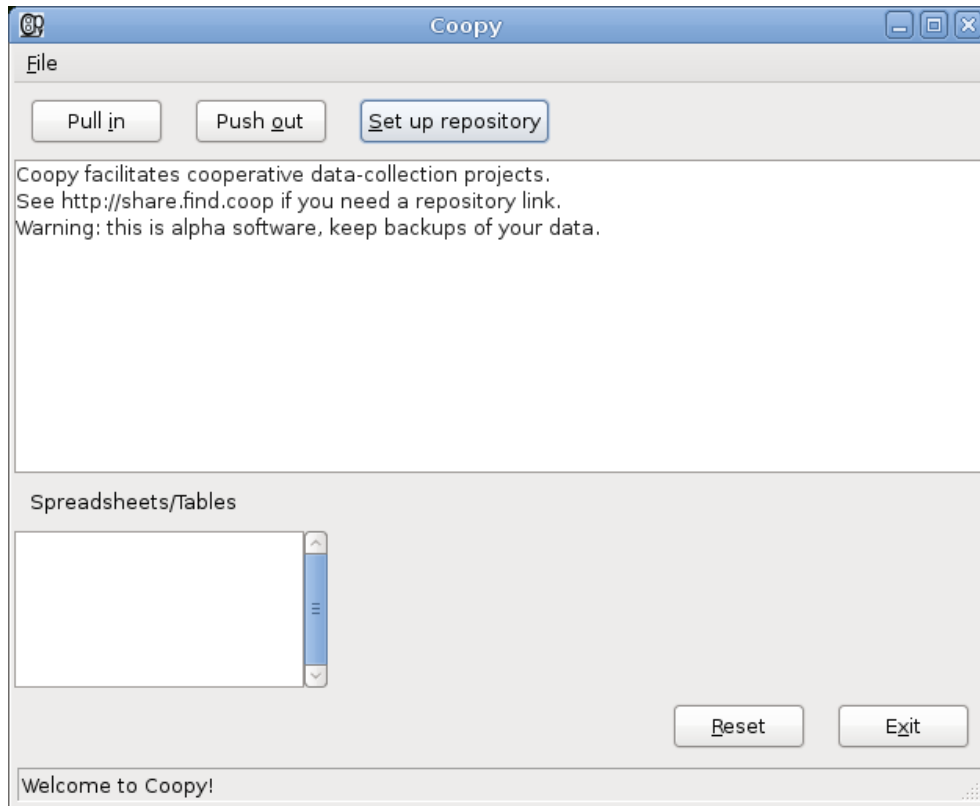


Figure 22: Click on: Set up repository

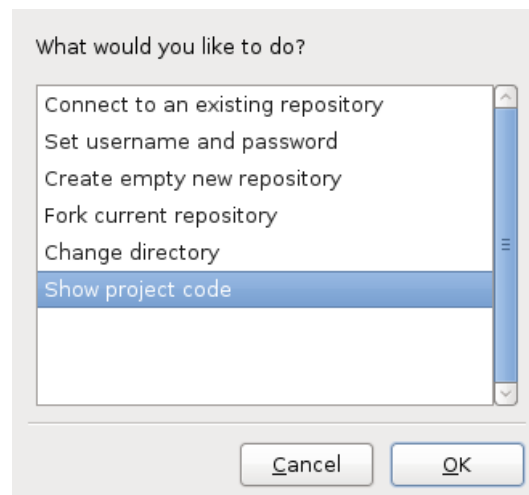


Figure 23: Select: Show project code

When we choose the project code option, Coopy gives us a big string of numbers and letters:

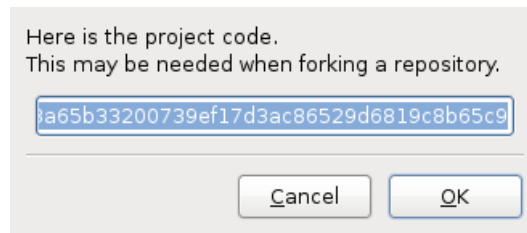


Figure 24: Project code

We will need the code a little later, don't worry about it for now, just be aware of where to find it. Clicking on the "Set up repository" link again and this time select the "fork repository" option:

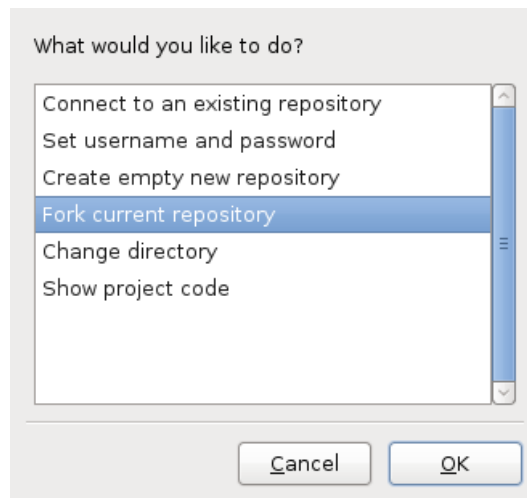


Figure 25: Fork repository option

We get a list of possible hosts for our repository:

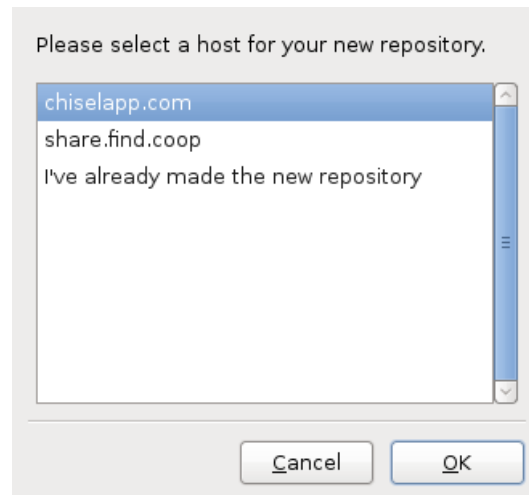


Figure 26: Fork host

Suppose we pick the first one. Then we follow the same sequence as in [Creating a new repository for Coopy](#), up to this point:

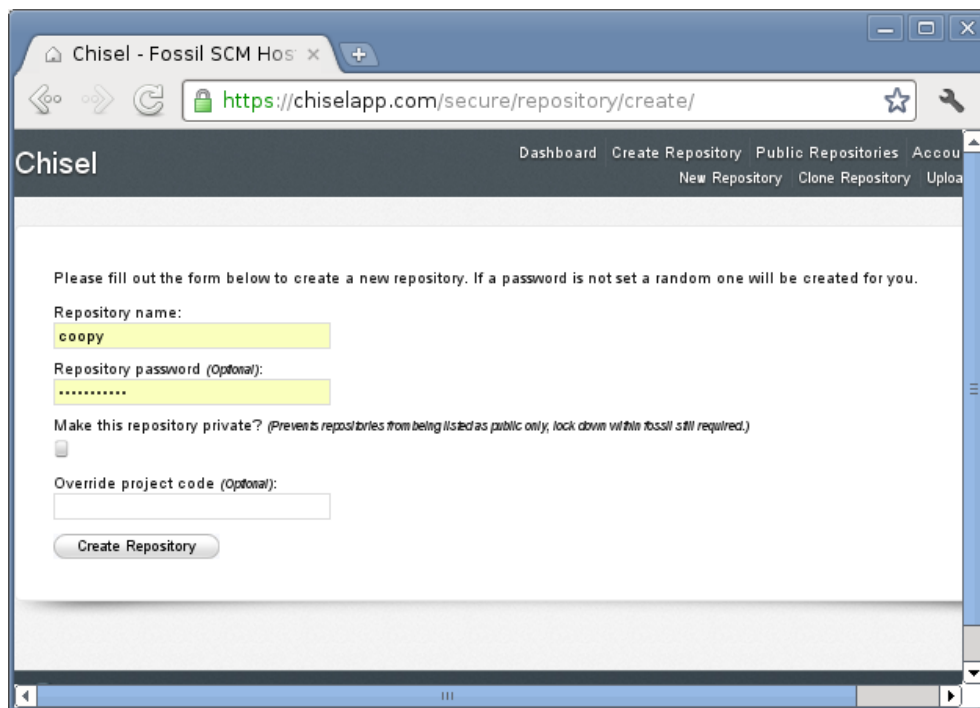


Figure 27: Create repository

Here we get the project code from Coopy and fill it in in the "Override project code" slot. Then, we get the repository link just like in [Creating a new repository for Coopy](#), and give it to Coopy:

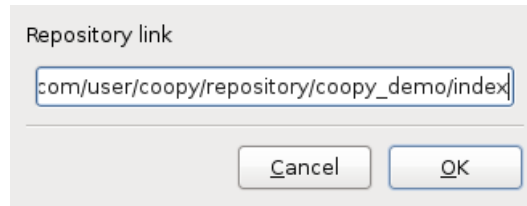


Figure 28: Provide new repository link

Now, we continue on to provide a username and password to Coopy as in [Pushing out a repository with Coopy](#). Done!

7 Resolving conflicts with Coopy

When working collaboratively, conflicts may occasionally happen.

Suppose Alice and Bob both share the following table with Coopy:

	A	B	C	D	E	F	G	H	I
1	NAME	DIGIT							
2	one	1							
3	two	2							
4	three	3							
5	four	9							
6	five	5							
7									
8									

numbers Sum=0

Figure 29: The starting point.

Both independently notice that the number "9" in this table is wrong, and fix it. Bob accidentally goofs:

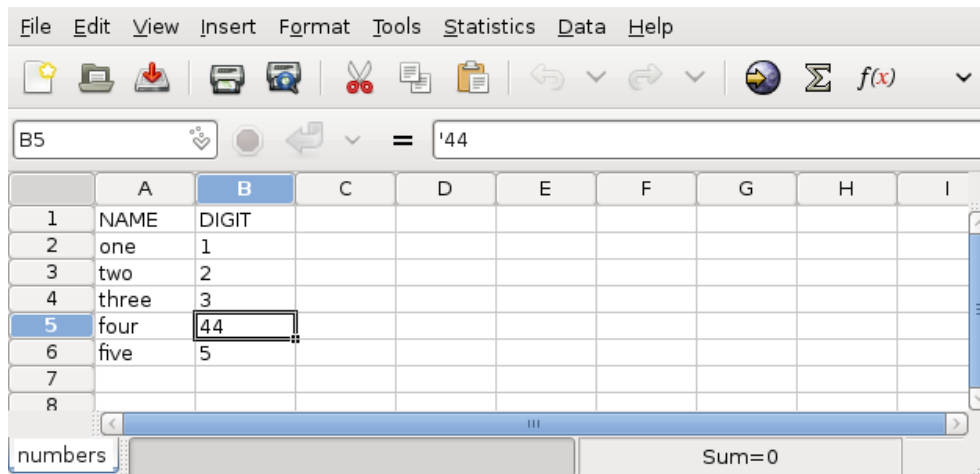


Figure 30: Bob's fix

Alice gets it right:

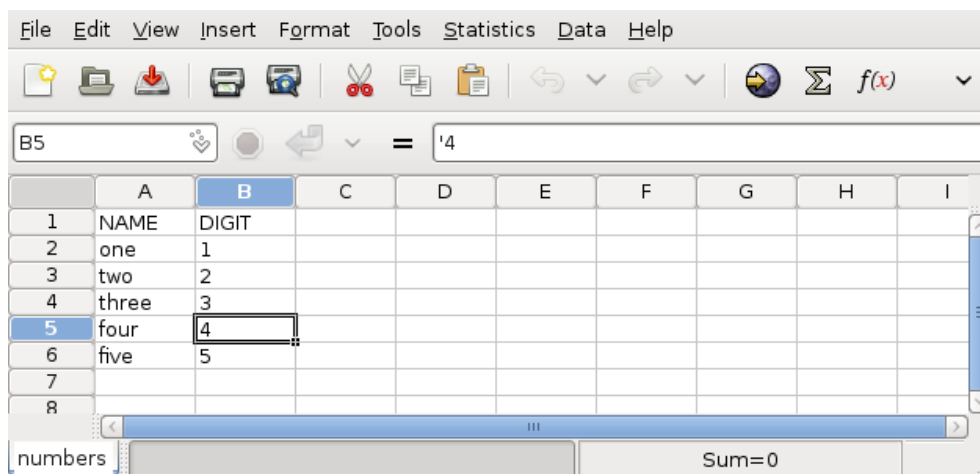


Figure 31: Alice's fix

Suppose Bob pushes his "fix" first to their shared repository. When Alice tries to push her fix, Coopy gives this message:

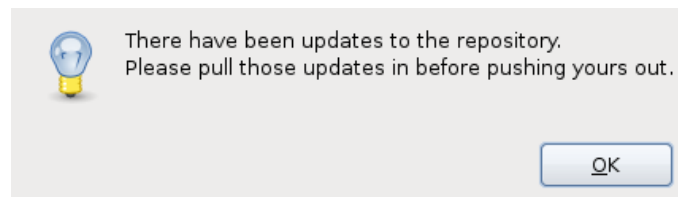


Figure 32: Pull needed.

Alice goes ahead and pulls. Compatible changes would get merged, but Coopy doesn't know what to do with the conflicting change. So it leaves Alice's table like this:

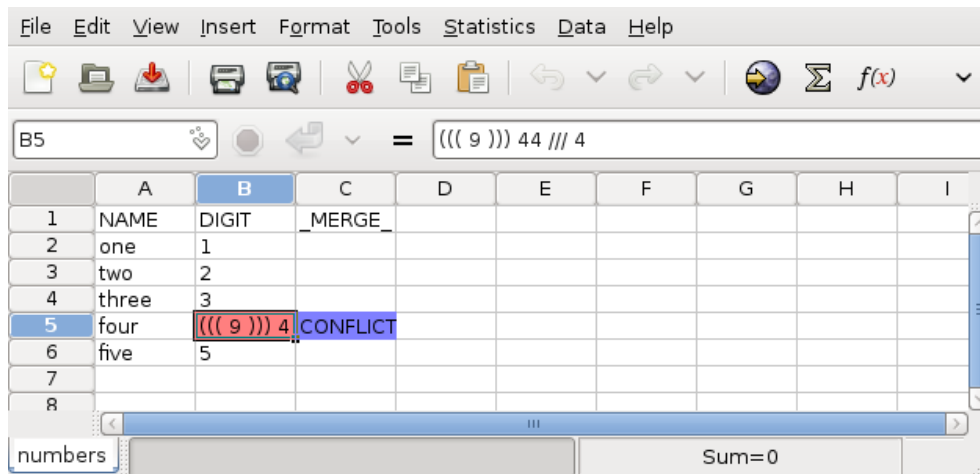


Figure 33: Conflicted table

Alice can recognize that "9" was replaced by either "4" (her choice) or "44" (Bob's choice). Using her magical human intelligence, she decides "4" is the right choice, and simply rewrites the conflicted cell, and deletes the "_MERGE_" column. Now she can push without trouble.

The Coopy toolbox has a [ssresolve](#) tool to speed up conflict resolving. Currently this works from the command-line only. Lots of graphical options should be integrated in the Coopy GUI soon.

8 Comparing tables using ssdiff

You can use [ssdiff](#) to summarize changes you've made between two versions of a table (or tables).

Suppose you are a bridge geek, and have been working with this table of New York bridges compiled by a friend:

bridge	designer	length
Brooklyn	J. A. Roebling	1595
Williamsburg	D. Duck	1600
Queensborough	Palmer & Hornbostel	1182
Triborough	O. H. Ammann	1380,383
Bronx Whitestone	O. H. Ammann	2300
Throgs Neck	O. H. Ammann	1800
George Washington	O. H. Ammann	3500
Spamspan	S. Spamington	10000

That table has some problems (D. Duck? Spamington?). So you fix them.

bridge	designer	length
Brooklyn	J. A. Roebling	1595
Manhattan	G. Lindenthal	1470
Williamsburg	L. L. Buck	1600
Queensborough	Palmer & Hornbostel	1182
Triborough	O. H. Ammann	1380,383
Bronx Whitestone	O. H. Ammann	2300
Throgs Neck	O. H. Ammann	1800
George Washington	O. H. Ammann	3500

As a public-spirited type, you'd like to offer your fixes to your friend. If you both happen to run the same software, chances are it has some way of tracking and highlighting your revisions, and everything is hunky-dorey. If not, then this is where sadness often begins. Some sad outcomes:

- You send your edited table to your friend, they say thanks, but since they don't have a quick way to see what you changed, or it is not in their preferred format, they never get around to actually merging in your changes.
- You try to convince your friend to use the same software as you, but they are not convinced. You argue, and pretty soon you are no longer friends.
- You try to convince your friend to use the same software as you, and they agree. The world becomes just a smidgeon more boring, and the status quo crusts over just a little bit more. Also, every problem your friend ever has with that software becomes your fault.
- You get fed up and don't send the table - too much hassle, too little reward.

COOPY can help in several ways, depending on your friend's preferences.

8.1 Presenting differences flexibly

The most basic way that COOPY can help is by letting you quickly prepare a summary of the changes you've made, in the style that is most suited to your friend. For example, if your friend is a spreadsheet user, you can use COOPY to prepare a report like this:

@@	bridge	designer	length
	Brooklyn	J. A. Roebling	1595
+++	Manhattan	G. Lindenthal	1470
->	Williamsburg	D. Duck->L. L. Buck	1600
	Queensborough	Palmer & Hornbostel	1182
	Triborough	O. H. Ammann	1380,383
...
	Throgs Neck	O. H. Ammann	1800
	George Washington	O. H. Ammann	3500
---	Spamspan	S. Spamington	10000

(produced with `ssdiff -format hilite -output report.xls FILE1 FILE2`)

This makes it very clear what you've changed, and if your friend's software understands this format, it could go ahead and apply these changes. Currently, only COOPY does, but we're working to get it more widely supported (see [Specification of the highlighter diff format](#)).

If your friend uses something that speaks SQL, COOPY can produce an equivalent report for them, along the lines of:

```
INSERT INTO bridges (bridge, designer, length) VALUES ('Manhattan', 'G.
Lindenthal', '1470');
UPDATE bridges SET designer='L. L. Buck' WHERE bridge='Williamsburg' AND
designer='D. Duck' AND length='1600';
DELETE FROM bridges WHERE bridge='Spamspan' AND designer='S. Spamington' AND
length='10000';
```

(produced with `ssdiff -format sql -default-table bridges FILE1 FILE2`)

If your friend happens to already be a COOPY fan, you could also send COOPY's own preferred format for representing differences, called **TDIFF**. This is modeled loosely after a format used for contributing to software projects.

```
* |bridge=Brooklyn|
+ |bridge:->Manhattan|designer:->G. Lindenthal|length:->1470|
= |bridge=Williamsburg|designer=D. Duck->L. L. Buck|
- |bridge=Spamspan|
```

(produced with `ssdiff FILE1 FILE2`)

With any of these formats, your friend can quickly review the changes you made.

8.2 Merging differences flexibly

Visualizing differences is already useful. Being able to selectively apply those differences is even more useful.

Let's switch perspective now. Suppose we've received an email from a friend, saying they've made some fixes to our bridge list, and they've attached this spreadsheet:

@ @	bridge	designer	length
	Brooklyn	J. A. Roebling	1595
+++	Manhattan	G. Lindenthal	1470
->	Williamsburg	D. Duck->L. L. Buck	1600
	Queensborough	Palmer & Hornbostel	1182
	Triborough	O. H. Ammann	1380,383
...
	Throgs Neck	O. H. Ammann	1800
	George Washington	O. H. Ammann	3500
---	Spamspan	S. Spamington	10000

That's handy, it is easy to see what has changed. The L.L.Buck fix is important, and how did we forget the Manhattan bridge - but the Spamspan bridge shouldn't be deleted (let's imagine we live in a parallel universe where S. Spamington is a bona fide designer, reknowned for making very long bridges).

It is simple to go ahead and make those changes manually, but we feel like doing some yak-shaving (look it up). The report is apparently in something called "hilite diff" format, supported by a vast list of programs that currently appears to be exactly one: COOPY. So we download the COOPY program, and quickly find out that we can just edit the report in anything, remove the lines we don't want applied, save the result, and then apply those changes to our table with [sspatch](#). Our information is in a table called "bridges" in an Sqlite database, ny.sqlite. So we change the sheet name in the report to "bridges" and do:

```
sspatch --inplace ny.sqlite report.xls/csv/...
```

Done!

9 Applying highlighter diffs using sspatch

The COOPY toolbox can report the difference between tables in a number of formats, one of which is called the "hilite" format.

If someone sends you such a report, what can you do with it?

9.1 How to recognize a highlighter diff

A good clue that you are looking at a highlighter diff is if you see annotations like "+++", "—", or "->" in the first column of one or more tables in a spreadsheet. Here is such a table:

@@	bridge	designer	length
	Brooklyn	J. A. Roebling	1595
+++	Manhattan	G. Lindenthal	1470
->	Williamsburg	D. Duck->L. L. Buck	1600
	Queensborough	Palmer & Hornbostel	1182
	Triborough	O. H. Ammann	1380,383
...
	Throgs Neck	O. H. Ammann	1800
	George Washington	O. H. Ammann	3500
---	Spamspan	S. Spamington	10000

A highlighter diff shows the differences between two versions of a spreadsheet. It says what should be done to the first version of the spreadsheet to produce the second - by adding rows, removing rows, changing cell values, etc. The diff does not generally show all of the original spreadsheet, just parts that have changed and their immediate surroundings. See: [Specification of the highlighter diff format](#).

9.2 Applying all changes from a highlighter diff

Highlighter diffs can be applied to a variety of database and spreadsheet formats using the `sspatch` program available in the COOPY toolbox. Suppose we have received the highlighter diff as a file "diff.xls". We can do any of:

```
sspatch --inplace my_table.xlsx diff.xls
sspatch --inplace my_table.ods diff.xls
sspatch --inplace my_table.gnumeric diff.xls
sspatch --inplace my_table.csv diff.xls
sspatch --inplace my_table.sqlite diff.xls
sspatch --inplace dbi:mysql:my_database:username=root diff.xls
...
```

See [sspatch](#) documentation for options.

9.3 Applying just some changes from a highlighter diff

Delete any rows from the diff that you don't want (or set the first column to empty). Then [apply as normal](#).

9.4 Converting to another format

A highlighter diff can be converted to other formats, using [ssrediff](#). Here's an SQL translation of the [above example](#):

```
INSERT INTO sheet (bridge, designer, length) VALUES ('Manhattan', 'G. Lindenthal', '1470');
UPDATE sheet SET designer='L. L. Buck' WHERE bridge='Williamsburg' AND designer='D. Duck' AND length='1600';
DELETE FROM sheet WHERE bridge='Spamspan' AND designer='S. Spamington' AND length='10000';
```

(produced with `ssrediff -format sql diff.xls`).

Here's a translation to TDIFF, the preferred format for the COOPY toolbox:

```
@@@ sheet

@ |bridge=|designer=length=|
* |Brooklyn|J. A. Roebling|1595|
+ |Manhattan|G. Lindenthal|1470|
= |Williamsburg|D. Duck->L. L. Buck|1600|
- |Spamspan|S. Spamington|10000|
```

(produced with `ssrediff -format tdiff diff.xls`).

Here's a translation to "ROWOPS" format, a simple tabular representation of changes this can be useful if you don't have to worry about NULLs.

name	op	bridge0	designer0	length0	bridge1	designer1	length1
sheet	context	Brooklyn	J. A. Roebling	1595			
sheet	insert				Manhattan	G. Lindenthal	1470
sheet	update	Williamsburg	D. Duck	1600		L. L. Buck	
sheet	delete	Spamspan	S. Spamington	10000			

(produced with `ssrediff -format ops diff.xls`).

There is a more elaborate tabular representation (`-format csv`) that deals with NULLs, schema changes, etc. but is less easy to read.

9.5 Accessing diffs programmatically

The programs in the COOPY toolbox are a thin wrapper around a library written in C++ that can be accessed from other languages. See the top level BUILD.txt file of the source code. Currently Python and Ruby are supported. For example, this code in python:

```
import coopy

class TableDiff(coopy.Patcher):

    def changeRow(self, change):
        print("Row change of type '%s'" % change.modeString())
        for c in change.cond.keys():
            print("* condition: %s = %s" % (c, change.cond[c].toString()))
        for v in change.val.keys():
            print("* value: %s -> %s" % (v, change.val[v].toString()))
        print("")
        return True

flags = coopy.CompareFlags()
differ = TableDiff()
parser = coopy.PatchParser(differ, "diff.xls", flags)
parser.applyPatch()
```

Will produce:

```
Row change of type 'context'
```

```

* condition: bridge = Brooklyn
* condition: designer = J. A. Roebling
* condition: length = 1595

Row change of type 'insert'
* value: bridge -> Manhattan
* value: designer -> G. Lindenthal
* value: length -> 1470

Row change of type 'update'
* condition: bridge = Williamsburg
* condition: designer = D. Duck
* condition: length = 1600
* value: designer -> L. L. Buck

Row change of type 'delete'
* condition: bridge = Spanspan
* condition: designer = S. Spamington
* condition: length = 10000

```

The equivalent code in Ruby is:

```

require 'coopy'

class TableDiff < Coopy::Patcher

  def changeRow(change)
    puts "Row change of type '%s' "%change.modeString()
    for c in change.cond.keys()
      puts "* condition: %s = %s" %[c, change.cond[c].toString()]
    end
    for c in change.val.keys()
      puts "* value: %s -> %s" %[c, change.val[c].toString()]
    end
    puts ""
    true
  end

end

flags = Coopy::CompareFlags.new()
differ = TableDiff.new()
parser = Coopy::PatchParser.new(differ, "diff.xls", flags)
parser.applyPatch()

```

The same code can read TDIFF format diff files.

10 Merging tables with diverging IDs

It is common to create numeric identifiers for records (rows) when inserting them in a table.

Merging two versions of a table modified in this way requires special attention, to avoid duplicate identifiers.

10.1 An example

Consider an Sqlite database `directory.sqlite` with the following schema:

```

CREATE TABLE locations (id INTEGER PRIMARY KEY,street,city);
CREATE TABLE org2loc (org_id INTEGER,loc_id INTEGER,
  FOREIGN KEY(org_id) REFERENCES org2loc(org_id),
  FOREIGN KEY(loc_id) REFERENCES locations(id));
CREATE TABLE organizations (id INTEGER PRIMARY KEY,name);

```

That's a table listing locations, a table listing organizations, and a table linking organizations and locations (`org2loc`). Suppose the database holds the following data:

locations

id	street	city
1	305 Memorial Drive	Cambridge
2	Big Crater	Moon

org2loc

org_id	loc_id
1	2
2	1
3	1
3	2

organizations

id	name
1	Dracula Inc
2	The Firm
3	Omni Cooperative
4	Nonexistence Unlimited

See [Getting the material used in this example](#) for how to get a copy of this database to play with. This toy database contains `organizations` and `locations`, where each organization may have zero, one, or many locations. The link between organizations and locations is set up in the `org2loc` table. For example, "The Firm" is located in Cambridge, "Omni Cooperative" is in Cambridge and on the Moon, and "Nonexistence Unlimited" is not linked to any location.

10.2 Divergence

Suppose Alice and Bob both have a copy of `directory.sqlite`. Alice adds in a new organization to her copy, "Alice and Company", with a location in Denver:

locations

@@	id	street	city
	1	305 Memorial Drive	Cambridge
	2	Big Crater	Moon
+++	3	10 Ten Street	Denver

org2loc

@@	org_id	loc_id
	1	2
	2	1
	3	1
	3	2
+++	5	3

organizations

@@	id	name
	1	Dracula Inc
	2	The Firm
	3	Omni Cooperative
	4	Nonexistence Unlimited
+++	5	Alice and Company

Bob, in his copy, adds "Bob's World" located on Planet Bob and in Cambridge, and notes that "Omni Cooperative" also has a branch on Planet Bob:

locations

@@	id	street	city
	1	305 Memorial Drive	Cambridge
	2	Big Crater	Moon
+++	3	42 The Boblands	Planet Bob

org2loc

@@	org_id	loc_id
	1	2
	2	1
	3	1
	3	2
+++	3	3
+++	5	1
+++	5	3

organizations

@@	id	name
	1	Dracula Inc
	2	The Firm
	3	Omni Cooperative
	4	Nonexistence Unlimited
+++	5	Bob's World

Bob then sends a "diff" bob.tdiff to Alice with his changes, either in the highlighter format shown above, or as a tdiff file:

```
@@@ locations
+ |id:->3|street:->42 The Boblands|city:->Planet Bob|

@@@ org2loc

@ |org_id|loc_id|
+ |3|3|
+ |5|1|
+ |5|3|

@@@ organizations
```

```
+ |id:->5|name:->'Bob''s World'|
```

(produced with `ssdiff -output bob.tdiff directory.sqlite directory_bob.sqlite`).

10.3 Convergence

A naive application of Bob's changes to Alice's version of the database would result in garbage, due to conflicting IDs. But when Alice applies Bob's changes using [sspatch](#), she gets a happier result:

locations

id	street	city
1	305 Memorial Drive	Cambridge
2	Big Crater	Moon
3	10 Ten Street	Denver
4	42 The Boblands	Planet Bob

org2loc

org_id	loc_id
1	2
2	1
3	1
3	2
5	3
3	4
6	1
6	4

organizations

id	name
1	Dracula Inc
2	The Firm
3	Omni Cooperative
4	Nonexistence Unlimited
5	Alice and Company
6	Bob's World

(Command: `sspatch -inplace directory.sqlite bob.tdiff`)

Notice how IDs have been modified appropriately. They are allocated by Sqlite rather than slavishly copied. If this is not the desired behavior, turn on the "–native" flag when calling [sspatch](#).

If Bob didn't make a patch but instead sent a copy of his database, Alice could either use [ssdiff](#) followed by [sspatch](#), or combine the two steps with [ssmerge](#). Doing:

```
ssmerge --inplace directory.sqlite directory_alice.sqlite directory_bob.sqlite
```

Alice will get:

locations

id	street	city
1	305 Memorial Drive	Cambridge
2	Big Crater	Moon
3	10 Ten Street	Denver
4	42 The Boblands	Planet Bob

org2loc

org_id	loc_id
1	2
2	1
3	1
3	2
5	3
3	4
6	1
6	4

organizations

id	name
1	Dracula Inc
2	The Firm
3	Omni Cooperative
4	Nonexistence Unlimited
5	Alice and Company
6	Bob's World

For merging, it is important that Alice has a copy of the last "common ancestor" between her own database and Bob's.

10.4 Getting the material used in this example

If you want to follow along, you can get a copy of the database used in this example by doing:

```
sspatch --test-file directory.sqlite
```

If you have the sqlite3 tool installed, you can view the database's content with:

```
sqlite3 directory.sqlite .dump      # how to make the database
sqlite3 directory.sqlite .schema   # structural part of database only
```

If you don't, you can use COOPY:

```
ssdiff directory.sqlite      # how to make the database
ssformat directory.sqlite    # non-structural part of database only
```

11 Using ssmerge with git

You can configure git to use [ssmerge](#) when merging tables in a repository.

- [Manage CSV files in git with COOPY](#)
- [Worked CSV example](#)
- [Troubleshooting COOPY with git](#)
- [Useful commands for testing](#)
- [Manage Sqlite databases in git with COOPY](#)
- [Worked Sqlite example](#)

The basic steps are:

- Add "custom merge driver" lines to your .gitconfig file, to add coopy's [ssmerge](#) command as a merge option.
- Add format handlers to a .gitattributes file, to make sure the files you want are merged using ssmerge.

11.1 Manage CSV files in git with COOPY

Find or create a .gitconfig file in your home directory, OR find the file .git/config in a repository. Add these lines to the end of this file, to create a "custom merge manager" for CSV files:

```
[merge "coopy-merge-csv"]
  name = coopy CSV merge
  driver = ssmerge --output dbi:csv::file=%A dbi:csv::file=%O dbi:csv::file=%A dbi:csv::file=%B
```

If ssmerge is not in your path, add the complete path to "ssmerge" in this file. This step needs to be done by each collaborator who wants to use COOPY.

Now, find or create a .gitattributes file in the same directory as the files you want COOPY to handle (there are other options for where to put this file, read the gitattributes documentation for details). Place these lines in .gitattributes:

```
*.csv merge=coopy-merge-csv
```

The .gitattributes file may be placed under version control, so this only needs to get set up once.

11.2 Worked CSV example

Let's make an empty git repository:

```
mkdir -p coopy_test/repo
cd coopy_test/repo
git init
```

Now, let's place a table in the repository. Let's start with a CSV file called "numbers.csv" with content like this:

```
NAME,DIGIT
one,1
two,2
thre,33
four,4
five,5
```

There are two intentional typos on the "thre" line. Add "numbers.csv" to the repository:

```
git add numbers.csv
git commit -m "add csv example"
```

Now, let's tell git to use a custom merge driver for .csv files. In the same directory as "numbers.csv", create a file called ".gitattributes" containing this:

```
*.csv merge=coopy-merge-csv
```

Let's add this to the repository too:

```
git add .gitattributes
git commit -m "add coopy rule"
```

Now, at the end of \$HOME/.gitconfig (create this file if it doesn't already exist), add on the following:

```
[merge "coopy-merge-csv"]
  name = coopy CSV merge
  driver = ssmerge --output dbi:csv::file=%A dbi:csv::file=%O dbi:csv::file=%A dbi:csv::file=%B
```

Now let's set up a clone of this repository for testing:

```
cd .. # should be in coopy_test directory now
git clone repo repo2
cd repo2
ls -a # should see numbers.csv and .gitattributes
```

Good. Now, to test, we'll make two non-conflicting changes on the same row, and see if they get merged without a problem. Regular text-based merges will choke on this. So, in repo2, modify "numbers.csv" to make "thre" be "three", and commit:

```
git commit -m "fix three" numbers.csv
```

Now, in repo, modify "numbers.csv" to make "33" be "3", and commit:

```
cd ../repo
git commit -m "fix 3" numbers.csv
```

Now try merging:

```
git pull ../repo2
```

Happy result:

```
From ../repo2
  * branch          HEAD      -> FETCH_HEAD
Auto-merging numbers.csv
Merge made by recursive.
 numbers.csv | 12 ++++++-----
 1 files changed, 6 insertions(+), 6 deletions(-)
```

And the contents of numbers.csv should be:

```
NAME,DIGIT
one,1
two,2
three,3
four,4
five,5
```

11.3 Troubleshooting COOPY with git

Things to check

- That the .gitconfig exists in your home directory and contains the needed rules.
- That there is a .gitattributes file in the same directory as your tables, and that it has the needed lines.
- That the "ssmerge" command is in your path. If you run "ssmerge" you should see a help message.
- That "ssmerge" is a recent version.
- That the "ssformat" command is in your path. If you run "ssformat" you should see a help message.
- That "ssformat" is a recent version.

If for some reason git doesn't use the coopy merge rule, then something like the following message will be shown during CSV merges:

```
remote: Counting objects: 5, done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From ../repo2
  * branch          HEAD      -> FETCH_HEAD
Auto-merging numbers.csv
CONFLICT (content): Merge conflict in numbers.csv
Automatic merge failed; fix conflicts and then commit the result.
```

and numbers.csv will contain the following:

```
NAME,DIGIT
one,1
two,2
<<<<<<< HEAD
three,3
=====
three,33
>>>>>>> b37613ebac50b552b4dd967c0f134930361c9070
four,4
five,5
```

This is the regular text merging algorithm. Undo the merge as follows:

```
git reset --hard HEAD # remove any uncommitted changes
```

And run through the checks listed earlier in this section. See also [Useful commands for testing](#).

11.4 Useful commands for testing

```
git reset --hard HEAD # remove any uncommitted changes, such as a bad merge
git reset --hard HEAD^ # remove any uncommitted changes, and revert the last
                        # commit
git pull ../path      # merge in changes from another version of repo
git pull              # merge in changes from same repo as last time
```

11.5 Manage Sqlite databases in git with COOPY

There are a few ways to version-control an Sqlite database using COOPY and git. A good option is to use git's filtering capabilities to keep the database in a text format in the repository (for meaningful diffs), while checked-out versions are in Sqlite's native binary format (for fast, easy access).

COOPY can deal with a variety of text formats, including Sqlite's text dump format. COOPY calls this "sqlitext" format. To preserve all Sqlite metadata, this is the best repository format to use.

Here's what we need in .gitconfig to convert Sqlite databases to and from text format (using [ssformat](#)), and to do sensible merges (using [ssmerge](#)).

```
[filter "coopy-filter-sqlite"]
  smudge = ssformat dbi:sqlitext:file=- dbi:sqlite:file=-
  clean = ssformat dbi:sqlite:file=- dbi:sqlitext:file=-

[merge "coopy-merge-sqlite"]
  name = coopy sqlite merge
  driver = ssmerge --named --unordered --output dbi:sqlitext::file=%A dbi:sqlitext::file=%O dbi:sqlitext::file=%A
```

Now, find or create a .gitattributes file in the same directory as the files you want COOPY to handle (there are other options for where to put this file, read the gitattributes documentation for details). Place these lines in .gitattributes:

```
*.sqlite filter=coopy-filter-sqlite
*.sqlite merge=coopy-merge-sqlite
```

The .gitattributes file may be placed under version control, so this only needs to get set up once.

11.6 Worked Sqlite example

Let's make an empty git repository:

```
mkdir -p coopy_test/repo
cd coopy_test/repo
git init
```

Now, let's place a database in the repository. It should end in the .sqlite extension and be an Sqlite database. For concreteness, we'll generate a test database here (but feel free to use your own):

```
ssformat --test-file numbers.sqlite
```

Add "numbers.sqlite" to the repository:

```
git add numbers.sqlite
git commit -m "add sqlite example"
```

Now, follow the setup for .gitconfig and .gitattributes in [Manage Sqlite databases in git with COOPY](#). In summary, you need to add this to \$HOME/.gitconfig:

```
[filter "coopy-filter-sqlite"]
    smudge = ssformat dbi:sqlitext:file=- dbi:sqlite:file=-
    clean = ssformat dbi:sqlite:file=- dbi:sqlitext:file=-

[merge "coopy-merge-sqlite"]
    name = coopy sqlite merge
    driver = ssmerge --named --unordered --output dbi:sqlitext::file=%A dbi:sqlitext::file=%O dbi:sqlitext::file=%A
```

And you should make a `.gitattributes` file in the same directory as the files you want COOPY to handle with these lines in it:

```
*.sqlite filter=coopy-filter-sqlite
*.sqlite merge=coopy-merge-sqlite
```

Let's add this to the repository:

```
git add .gitattributes
git commit -m "add coopy sqlite rule"
```

Now let's set up a clone of this repository for testing:

```
cd .. # should be in coopy_test directory now
git clone repo repo2
cd repo2
ls -a # should see numbers.sqlite and .gitattributes
```

In the clone, the `numbers.sqlite` should be a valid Sqlite database. It will have been translated to and from a text representation, so it is worth checking this. If there's a problem, see [Troubleshooting COOPY with git](#).

Now, to test, we'll make two non-conflicting changes on the same row, and see if they get merged without a problem. In `repo2`, modify "`numbers.sqlite`" to make "three" be "threepio", using either your favorite sqlite editor (e.g. the `sqlite3` command-line tool) or `sspatch`:

```
sspatch numbers.sqlite --inplace --cmd "= |NAME:three->threepio|"
ssformat numbers.sqlite # check change
git commit -m "scramble three" numbers.sqlite
```

Now, in `repo`, modify "`numbers.sqlite`" make "3" be "33", and commit:

```
cd ../repo
sspatch numbers.sqlite --inplace --cmd "= |three|3->33|"
ssformat numbers.sqlite # check change
git commit -m "scramble 3" numbers.sqlite
\endve
```

```
Now try merging:
\verbatimim
git pull ../repo2
```

Here's what a happy result looks like, if everything is configured well:

```
From ../repo2
 * branch          HEAD          -> FETCH_HEAD
Auto-merging numbers.sqlite
Merge made by recursive.
 numbers.sqlite | 2 +-
 1 files changed, 1 insertions(+), 1 deletions(-)
```

And the contents of `numbers.sqlite` should be:

```
== sheet ==
NAME,DIGIT
```

```

-----
one,1
two,2
threepio,33
four,4
five,5

```

If there's a problem, see [Troubleshooting COOPY with git](#).

Let's look now at what happens if there is a conflict. Let's change "2" to different values in the different repositories:

```

cd ../repo2; git pull ../repo master
sspatch numbers.sqlite --inplace --cmd "= |two|2->22|"
git commit -m "conflict 22" numbers.sqlite
cd ../repo
sspatch numbers.sqlite --inplace --cmd "= |two|2->222|"
git commit -m "conflict 222" numbers.sqlite
git pull ../repo2

```

We get this message from git:

```

From ../repo2
 * branch          HEAD          -> FETCH_HEAD
# conflict: {{222}} vs {{22}} from {{2}}
Conflict detected.
Auto-merging numbers.sqlite
CONFLICT (content): Merge conflict in numbers.sqlite
Automatic merge failed; fix conflicts and then commit the result.

```

And the content of numbers.sqlite is:

```

== sheet ==
NAME,DIGIT,_MERGE_
-----
one,1,NULL
two,"((( 2 ))) 222 /// 22",CONFLICT
threepio,33,NULL
four,4,NULL
five,5,NULL

```

There's a new column marking where conflicts occurred. We use Sqlite's willingness to put any kind of value in any column regardless of its official type to squeeze information into the conflicting cell. If you've ideas on other ways to present this data, please let the COOPY developers know.

Suppose we decide that our version was best. We resolve the conflict either by editing the table or using [ssresolve](#) as follows:

```
ssresolve --ours numbers.sqlite
```

This gives us a non-conflicted table again:

```

== sheet ==
NAME,DIGIT
-----
one,1
two,222
threepio,33
four,4
five,5

```

Then we tell git:

```

git add numbers.sqlite
git commit -m "resolved conflict"

```

Done!

12 ssdiff

Show the difference between two tables/databases/spreadsheets.

12.1 Usage

- `ssdiff [options] FILE1 FILE2`

12.2 Index

- [Option summary](#)
- [Option details](#)
- [Examples](#)
- [Patch formats](#)
- [Database/spreadsheet file formats](#)
- [Version](#)

12.3 Option summary

- `-act=ACT`
- `-apply`
- `-bid=COLUMN`
- `-context=N`
- `-default-table=TABLE`
- `-fixed-columns`
- `-format=FORMAT`
- `-git`
- `-head-trimmed`
- `-headerless`
- `-help`
- `-id=COLUMN`
- `-input-formats`
- `-low-memory`
- `-named`
- `-omit-format-name`
- `-omit-sheet-name`
- `-output=OUTPUTFILE`

- `-parent=PARENT`
- `-patch-formats`
- `-scan-for-patch`
- `-table=TABLE`
- `-tail-trimmed`
- `-unordered`
- `-variant=VARIANT`

12.4 Option details

`-act=ACT`

filter for an action of a particular type (update, insert, delete, none, schema)

`-apply`

apply difference between FILE1 and FILE2 immediately to FILE1

`-bid=COLUMN`

boost a column (repeat option for multiple columns)

`-context=N`

Number of rows of context before and after changes for highlighter diffs ("all" to include all rows)

`-default-table=TABLE`

name to use when a table name is needed and not supplied

`-fixed-columns`

ignore new or removed columns

`-format=FORMAT`

set difference format for output

`-git`

expect git-compatible parameters (path old-file old-hex old-mode new-file new-hex new-mode)

`-head-trimmed`

ignore rows removed at the beginning of a table (such as a log file)

`-headerless`

treat any embedded column names as regular parts of the table (for formats like CSV)

`-help`

show how to use this program

`-id=COLUMN`

set primary key (repeat option for multi-column key)

`-input-formats`

list supported input database formats

`-low-memory`

prioritize low memory usage over speed

–named

trust names of columns, omitting checks for column renames

–omit-format-name

omit any version-dependent header from diff

–omit-sheet-name

omit any sheet/table name from diff

–output=OUTPUTFILE

direct output to this file (default is standard output)

–parent=PARENT

use named workbook/database as common ancestor in difference calculations

–patch-formats

list supported patch formats

–scan-for-patch

check if FILE2 looks like a patch, and if so, apply it

–table=TABLE

filter for a named table of a workbook/database (repeat option for multiple tables)

–tail-trimmed

ignore rows removed at the end of a table (such as a log file)

–unordered

treat order of rows as unimportant

–variant=VARIANT

set the desired dialect when using a poorly defined output format (currently for SQL, available variants are: sqlite, access)

12.5 Examples

You can generate test file(s) for the examples that follow:

```
ssdiff --test-file numbers.csv
ssdiff --test-file numbers.sqlite
ssdiff --test-file numbers_buggy.csv
ssdiff --test-file numbers_buggy.sqlite
```

12.5.1 Example 1

```
ssdiff numbers_buggy.csv numbers.csv
```

Compare two tables. Output goes to standard output.

12.5.2 Example 2

```
ssdiff --unordered numbers_buggy.csv numbers.csv
```

Compare two tables, neglecting row order.

12.5.3 Example 3

```
ssdiff --format sql numbers_buggy.sqlite numbers.sqlite
```

Compare two databases, with output in SQL format.

12.5.4 Example 4

```
ssdiff --format hilite --output review.csv numbers_buggy.csv numbers.csv
```

Generate tabular diff for eyeballing. If ssdiff is compiled with gnumeric support, and output format is *.xls, color highlighting is added.

12.6 Patch formats

- **tdiff**: *[default]* reminiscent of the standard unix diff format for text
- **csv**: csv-compatible diff format
- **hilite**: colorful spreadsheet format
- **index**: tabular output showing relationship between rows and columns
- **novel**: mark all shared rows - remaining rows are unmatched
- **ops**: summarize modified rows in a table
- **raw**: verbose diff format for debugging
- **review**: spreadsheet diff format suitable for quickly accepting or rejecting changes
- **sql**: SQL format (data diffs only)
- **stats**: produce statistics on table changes

12.7 Database/spreadsheet file formats

CSV: plain-text delimiter-separated family of formats

- **.csv**: Comma-separated values
- **.tsv**: Tab-separated values
- **.ssv**: Semicolon-separated values
- **.json**: {
 "type": "csv", // CSV family
 "file": "fname.dsv", // File name
 "delimiter": "|" // Delimiter character
 }

SQLITE: file-based database

- **.sqlite**: Sqlite database file
- **.json**: {
 "type": "sqlite", // *Sqlite family*
 "file": "fname.db" // *File name*
}
- **dbi:sqlite:fname.db** (Force sqlite interpretation)

SQLITEXT: sqlite-format sql dump

- **.sqlitext**: SQL dump of Sqlite database
- **.json**: {
 "type": "sqlitext", // *Sqlitext family*
 "file": "fname.sql" // *File name*
}
- **dbi:sqlitext:fname.sql** (Force sqlitext interpretation)

JSONBOOK: Spreadsheet formats in json

- **.jsonbook**: Json spreadsheet
- **.json**: {
 "type": "jsonbook", // *Json spreadsheet*
 "file": "fname.sheet" // *File name*
}
- **dbi:jsonbook:fname.sheet** (Force Json spreadsheet interpretation)

GNUMERIC: Spreadsheet formats (via gnumeric)

- **.xls**: Excel spreadsheet
- **.xlsx**: Excel spreadsheet
- **.gnumeric**: Gnumeric spreadsheet
- **.ods**: OpenOffice/LibreOffice spreadsheet
- **.json**: {
 "type": "gnumeric", // *Gnumeric family*
 "file": "fname.sheet" // *File name*
}
- **dbi:gnumeric:fname.sheet** (Force Gnumeric interpretation)

MDB: Access database format (via Mdbtools, READ-ONLY)

- **.mdb**: Access database file
- **.json**: {
 "type": "access", // Access family
 "file": "fname.db" // File name
 }
- **dbi:access:fname.db** (Force Access interpretation)

JMDB: Access database format (via Jackcess)

- **.mdb**: Access database file
- **.json**: {
 "type": "access", // Jackcess/access family
 "file": "fname.db" // File name
 }
- **dbi:jackcess:fname.db** (Force Access interpretation)

MYSQL: database connector

- **.json**: {
 "type": "mysql", // MYSQL connector
 "database": "db_name", // Database name
 "host": "localhost", // Host name
 "port": "1111", // Port number
 "username": "root", // Username
 "password": "*****" // Password
 }
- **dbi:mysql:database_name** (Use default port, username, etc)
- **dbi:mysql:database_name:username=USER:password=PASS**
- **dbi:mysql:database_name:host=HOST:port=PORT**

SOCIALCALC: SocialCalc format (via mozjs)

- **.socialcalc**: SocialCalc spreadsheet file
- **.sc**: SocialCalc spreadsheet file
- **.json**: {
 "type": "socialcalc", // SocialCalc family
 "file": "sheet.txt" // File name
 }
- **dbi:socialcalc:sheet.txt** (Force SocialCalc interpretation)

12.8 Version

ssdiff version 0.6.4

13 sspatch

Modify a table/database/spreadsheet to integrate the changes described in a pre-computed difference.

13.1 Usage

- `sspatch [options] DATAFILE PATCHFILE`
- `sspatch [options] --cmd PATCHSTRING DATAFILE`

13.2 Index

- [Option summary](#)
- [Option details](#)
- [Examples](#)
- [Patch formats](#)
- [Database/spreadsheet file formats](#)
- [Version](#)

13.3 Option summary

- `--act=ACT`
- `--cmd=CMD`
- `--default-table=TABLE`
- `--help`
- `--inplace`
- `--input-formats`
- `--neither`
- `--ours`
- `--output=OUTPUTFILE`
- `--patch-formats`
- `--table=TABLE`
- `--theirs`

13.4 Option details

-act=ACT

filter for an action of a particular type (update, insert, delete, none, schema)

-cmd=CMD

specify an action inline in tdiff format

-default-table=TABLE

name to use when a table name is needed and not supplied

-help

show how to use this program

-inplace

if modifications are made, make them in place without a copy

-input-formats

list supported input database formats

-neither

in case of conflict use cell value from common ancestor

-ours

in case of conflict use cell value that was the local choice

-output=OUTPUTFILE

direct output to this file (default is standard output)

-patch-formats

list supported patch formats

-table=TABLE

filter for a named table of a workbook/database (repeat option for multiple tables)

-theirs

in case of conflict use cell value that wasn't the local choice

13.5 Examples

You can generate test file(s) for the examples that follow:

```
sspatch --test-file directory.sqlite
sspatch --test-file numbers.csv
sspatch --test-file numbers_buggy.csv
sspatch --test-file numbers_buggy.sqlite
```

13.5.1 Example 1

```
sspatch numbers_buggy.csv numbers_patch.tdiff
```

Apply a patch to a table. Output goes to standard output. Input file is untouched.

13.5.2 Example 2

```
sspatch --inplace numbers_buggy.csv numbers_patch.tdiff
```

Apply a patch to a table. Input file is modified.

13.5.3 Example 3

```
sspatch --tmp tmp.sqlite numbers_buggy.sqlite numbers_patch.tdiff
```

Apply a patch to a sqlite database. Input file is not modified. Space for a temporary database is needed to do this. If not supplied, sspatch will ask for it.

13.5.4 Example 4

```
sspatch - numbers_patch.tdiff < numbers_buggy.csv
```

Apply a patch to a table read from standard input.

13.5.5 Example 5

```
sspatch numbers_buggy.csv - < numbers_patch.tdiff
```

Apply a patch read from standard input.

13.5.6 Example 6

```
sspatch --cmd "+ |two|2|" numbers_buggy.csv
```

Add a new row to a table.

13.5.7 Example 7

```
sspatch --cmd "- |NAME=four|" numbers.csv
```

Remove a row from a table

13.5.8 Example 8

```
sspatch --cmd "= |NAME=four|DIGIT:*->4|" numbers_buggy.csv
```

Change the DIGIT column on a row with NAME=four.

13.5.9 Example 9

```
sspatch --inplace --table organizations \  
--cmd "+ |42|The New Organization|" directory.sqlite
```

Add an organization. The specified organization ID is replaced by an autoincremented ID generated by sqlite. Use 'ssformat directory.sqlite' to view result.

13.5.10 Example 10

```
sspatch --native --inplace --table organizations \
--cmd "+ |42|The New Organization|" directory.sqlite
```

Add an organization. The specified organization ID is used (due to `--native` flag). Use `'ssformat directory.sqlite'` to view result.

13.6 Patch formats

- **tdiff**: *[default]* reminiscent of the standard unix diff format for text
- **csv**: csv-compatible diff format
- **hilite**: colorful spreadsheet format
- **index**: tabular output showing relationship between rows and columns
- **novel**: mark all shared rows - remaining rows are unmatched
- **ops**: summarize modified rows in a table
- **raw**: verbose diff format for debugging
- **review**: spreadsheet diff format suitable for quickly accepting or rejecting changes
- **sql**: SQL format (data diffs only)
- **stats**: produce statistics on table changes

13.7 Database/spreadsheet file formats

CSV: plain-text delimiter-separated family of formats

- **.csv**: Comma-separated values
- **.tsv**: Tab-separated values
- **.ssv**: Semicolon-separated values
- **.json**: {
 "type": "csv", // CSV family
 "file": "fname.dsv", // File name
 "delimiter": "|" // Delimiter character
 }

SQLITE: file-based database

- **.sqlite**: Sqlite database file
- **.json**: {
 "type": "sqlite", // Sqlite family
 "file": "fname.db" // File name
 }

- **dbi:sqlite:fname.db** (Force sqlite interpretation)

SQLTEXT: sqlite-format sql dump

- **.sqltext**: SQL dump of Sqlite database
- **.json**: {
 "type": "sqltext", // *Sqltext family*
 "file": "fname.sql" // *File name*
}
- **dbi:sqltext:fname.sql** (Force sqltext interpretation)

JSONBOOK: Spreadsheet formats in json

- **.jsonbook**: Json spreadsheet
- **.json**: {
 "type": "jsonbook", // *Json spreadsheet*
 "file": "fname.sheet" // *File name*
}
- **dbi:jsonbook:fname.sheet** (Force Json spreadsheet interpretation)

GNUMERIC: Spreadsheet formats (via gnumeric)

- **.xls**: Excel spreadsheet
- **.xlsx**: Excel spreadsheet
- **.gnumeric**: Gnumeric spreadsheet
- **.ods**: OpenOffice/LibreOffice spreadsheet
- **.json**: {
 "type": "gnumeric", // *Gnumeric family*
 "file": "fname.sheet" // *File name*
}
- **dbi:gnumeric:fname.sheet** (Force Gnumeric interpretation)

MDB: Access database format (via Mdbtools, READ-ONLY)

- **.mdb**: Access database file
- **.json**: {
 "type": "access", // *Access family*
 "file": "fname.db" // *File name*
}
- **dbi:access:fname.db** (Force Access interpretation)

JMDB: Access database format (via Jackcess)

- **.mdb**: Access database file
- **.json**: {
 "type": "access", // *Jackcess/access family*
 "file": "fname.db" // *File name*
 }
- **dbi:jackcess:fname.db** (Force Access interpretation)

MYSQL: database connector

- **.json**: {
 "type": "mysql", // *MYSQL connector*
 "database": "db_name", // *Database name*
 "host": "localhost", // *Host name*
 "port": "1111", // *Port number*
 "username": "root", // *Username*
 "password": "*****" // *Password*
 }
- **dbi:mysql:database_name** (Use default port, username, etc)
- **dbi:mysql:database_name:username=USER:password=PASS**
- **dbi:mysql:database_name:host=HOST:port=PORT**

SOCIALCALC: SocialCalc format (via mozjs)

- **.socialcalc**: SocialCalc spreadsheet file
- **.sc**: SocialCalc spreadsheet file
- **.json**: {
 "type": "socialcalc", // *SocialCalc family*
 "file": "sheet.txt" // *File name*
 }
- **dbi:socialcalc:sheet.txt** (Force SocialCalc interpretation)

13.8 Version

sspatch version 0.6.4

14 ssmerge

Merge table/database/spreadsheets.

The first file must be a common ancestor of the remaining two.

14.1 Usage

- `ssmerge [options] FILE1 FILE2 FILE3`

14.2 Index

- [Option summary](#)
- [Option details](#)
- [Examples](#)
- [Patch formats](#)
- [Database/spreadsheet file formats](#)
- [Version](#)

14.3 Option summary

- `-bid=COLUMN`
- `-default-table=TABLE`
- `-fixed-columns`
- `-head-trimmed`
- `-headerless`
- `-help`
- `-id=COLUMN`
- `-inplace`
- `-input-formats`
- `-named`
- `-output=OUTPUTFILE`
- `-patch-formats`
- `-table=TABLE`
- `-tail-trimmed`
- `-unordered`

14.4 Option details

-bid=COLUMN

boost a column (repeat option for multiple columns)

-default-table=TABLE

name to use when a table name is needed and not supplied

-fixed-columns

ignore new or removed columns

-head-trimmed

ignore rows removed at the beginning of a table (such as a log file)

-headerless

treat any embedded column names as regular parts of the table (for formats like CSV)

-help

show how to use this program

-id=COLUMN

set primary key (repeat option for multi-column key)

-inplace

if modifications are made, make them in place without a copy

-input-formats

list supported input database formats

-named

trust names of columns, omitting checks for column renames

-output=OUTPUTFILE

direct output to this file (default is standard output)

-patch-formats

list supported patch formats

-table=TABLE

filter for a named table of a workbook/database (repeat option for multiple tables)

-tail-trimmed

ignore rows removed at the end of a table (such as a log file)

-unordered

treat order of rows as unimportant

14.5 Examples

You can generate test file(s) for the examples that follow:

```
ssmerge --test-file numbers.csv
ssmerge --test-file numbers_buggy.csv
ssmerge --test-file numbers_buggy_add.csv
ssmerge --test-file numbers_conflict.csv
```

14.5.1 Example 1

```
ssmerge numbers_buggy.csv numbers.csv numbers_buggy_add.csv
```

Merge two CSV tables (numbers.csv and numbers_buggy_add.csv) with a common ancestor (numbers_buggy.csv).

14.5.2 Example 2

```
ssmerge --theirs numbers_buggy.csv numbers.csv numbers_conflict.csv
```

Merge numbers.csv and numbers_conflict.csv (with common ancestor numbers_buggy.csv), deferring to numbers_conflict.csv in the case of conflict.

14.5.3 Example 3

```
ssmerge --ours numbers_buggy.csv numbers.csv numbers_conflict.csv
```

Merge numbers.csv and numbers_conflict.csv (with common ancestor numbers_buggy.csv), deferring to numbers.csv in the case of conflict.

14.5.4 Example 4

```
ssmerge --inplace --theirs numbers_buggy.csv numbers.csv numbers_conflict.csv
```

Merge directly into numbers.csv. Without `--inplace`, output goes to standard output.

14.6 Patch formats

- **tdiff**: *[default]* reminiscent of the standard unix diff format for text
- **csv**: csv-compatible diff format
- **hilite**: colorful spreadsheet format
- **index**: tabular output showing relationship between rows and columns
- **novel**: mark all shared rows - remaining rows are unmatched
- **ops**: summarize modified rows in a table
- **raw**: verbose diff format for debugging
- **review**: spreadsheet diff format suitable for quickly accepting or rejecting changes
- **sql**: SQL format (data diffs only)
- **stats**: produce statistics on table changes

14.7 Database/spreadsheet file formats

CSV: plain-text delimiter-separated family of formats

- **.csv**: Comma-separated values
- **.tsv**: Tab-separated values
- **.ssv**: Semicolon-separated values
- **.json**: {
 "type": "csv", // *CSV family*
 "file": "fname.dsv", // *File name*
 "delimiter": "|" // *Delimiter character*
}

SQLITE: file-based database

- **.sqlite**: Sqlite database file
- **.json**: {
 "type": "sqlite", // *Sqlite family*
 "file": "fname.db" // *File name*
}
- **dbi:sqlite:fname.db** (Force sqlite interpretation)

SQLITEXT: sqlite-format sql dump

- **.sqlitext**: SQL dump of Sqlite database
- **.json**: {
 "type": "sqlitext", // *Sqlitext family*
 "file": "fname.sql" // *File name*
}
- **dbi:sqlitext:fname.sql** (Force sqlitext interpretation)

JSONBOOK: Spreadsheet formats in json

- **.jsonbook**: Json spreadsheet
- **.json**: {
 "type": "jsonbook", // *Json spreadsheet*
 "file": "fname.sheet" // *File name*
}
- **dbi:jsonbook:fname.sheet** (Force Json spreadsheet interpretation)

GNUMERIC: Spreadsheet formats (via gnumeric)

- **.xls**: Excel spreadsheet
- **.xlsx**: Excel spreadsheet
- **.gnnumeric**: Gnumeric spreadsheet
- **.ods**: OpenOffice/LibreOffice spreadsheet
- **.json**: {
 "type": "gnnumeric", // *Gnumeric family*
 "file": "fname.sheet" // *File name*
 }
- **dbi:gnnumeric:fname.sheet** (Force Gnumeric interpretation)

MDB: Access database format (via Mdbtools, READ-ONLY)

- **.mdb**: Access database file
- **.json**: {
 "type": "access", // *Access family*
 "file": "fname.db" // *File name*
 }
- **dbi:access:fname.db** (Force Access interpretation)

JMDB: Access database format (via Jackcess)

- **.mdb**: Access database file
- **.json**: {
 "type": "access", // *Jackcess/access family*
 "file": "fname.db" // *File name*
 }
- **dbi:jackcess:fname.db** (Force Access interpretation)

MYSQL: database connector

- **.json**: {
 "type": "mysql", // *MYSQL connector*
 "database": "db_name", // *Database name*
 "host": "localhost", // *Host name*
 "port": "1111", // *Port number*
 "username": "root", // *Username*
 "password": "*****" // *Password*
 }
- **dbi:mysql:database_name** (Use default port, username, etc)
- **dbi:mysql:database_name:username=USER:password=PASS**

- **dbi:mysql:database_name:host=HOST:port=PORT**

SOCIALCALC: SocialCalc format (via mozjs)

- **.socialcalc**: SocialCalc spreadsheet file
- **.sc**: SocialCalc spreadsheet file
- **.json**: {
 "type": "socialcalc", // *SocialCalc family*
 "file": "sheet.txt" // *File name*
}
- **dbi:socialcalc:sheet.txt** (Force SocialCalc interpretation)

14.8 Version

ssmerge version 0.6.4

15 ssresolve

Resolve a file with conflicts from ssmerge.

15.1 Usage

- ssresolve [options] FILE

15.2 Index

- [Option summary](#)
- [Option details](#)
- [Examples](#)
- [Patch formats](#)
- [Database/spreadsheet file formats](#)
- [Version](#)

15.3 Option summary

- [-default-table=TABLE](#)
- [-dry-run](#)
- [-help](#)
- [-neither](#)
- [-ours](#)
- [-theirs](#)

15.4 Option details

--default-table=TABLE

name to use when a table name is needed and not supplied

--dry-run

make no changes, just describe what would happen

--help

show how to use this program

--neither

in case of conflict use cell value from common ancestor

--ours

in case of conflict use cell value that was the local choice

--theirs

in case of conflict use cell value that wasn't the local choice

15.5 Examples

You can generate test file(s) for the examples that follow:

```
ssresolve --test-file numbers.csv
ssresolve --test-file numbers_buggy.csv
ssresolve --test-file numbers_conflict.csv
```

15.5.1 Example 1

```
ssresolve numbers_muddle.csv
```

Check if file is resolved.

15.5.2 Example 2

```
ssresolve --ours numbers_muddle.csv
```

Resolve conflicts in favor of local/left values.

15.5.3 Example 3

```
ssresolve --theirs numbers_muddle.csv
```

Resolve conflicts in favor of remote/right values.

15.5.4 Example 4

```
ssresolve --neither numbers_muddle.csv
```

Resolve conflicts in favor of ancestral values.

15.6 Patch formats

- **tdiff**: *[default]* reminiscent of the standard unix diff format for text
- **csv**: csv-compatible diff format
- **hilite**: colorful spreadsheet format
- **index**: tabular output showing relationship between rows and columns
- **novel**: mark all shared rows - remaining rows are unmatched
- **ops**: summarize modified rows in a table
- **raw**: verbose diff format for debugging
- **review**: spreadsheet diff format suitable for quickly accepting or rejecting changes
- **sql**: SQL format (data diffs only)
- **stats**: produce statistics on table changes

15.7 Database/spreadsheet file formats

CSV: plain-text delimiter-separated family of formats

- **.csv**: Comma-separated values
- **.tsv**: Tab-separated values
- **.ssv**: Semicolon-separated values
- **.json**: {
 "type": "csv", // *CSV family*
 "file": "fname.dsv", // *File name*
 "delimiter": "|" // *Delimiter character*
 }

SQLITE: file-based database

- **.sqlite**: Sqlite database file
- **.json**: {
 "type": "sqlite", // *Sqlite family*
 "file": "fname.db" // *File name*
 }
- **dbi:sqlite:fname.db** (Force sqlite interpretation)

SQLITEXT: sqlite-format sql dump

- **.sqlitext**: SQL dump of Sqlite database

- **.json:** {
 "type": "sqlite", // *Sqlite family*
 "file": "fname.sql" // *File name*
}
- **dbi:sqlite:fname.sql** (Force sqlite interpretation)

JSONBOOK: Spreadsheet formats in json

- **.jsonbook:** Json spreadsheet
- **.json:** {
 "type": "jsonbook", // *Json spreadsheet*
 "file": "fname.sheet" // *File name*
}
- **dbi:jsonbook:fname.sheet** (Force Json spreadsheet interpretation)

GNUMERIC: Spreadsheet formats (via gnumeric)

- **.xls:** Excel spreadsheet
- **.xlsx:** Excel spreadsheet
- **.gnumeric:** Gnumeric spreadsheet
- **.ods:** OpenOffice/LibreOffice spreadsheet
- **.json:** {
 "type": "gnumeric", // *Gnumeric family*
 "file": "fname.sheet" // *File name*
}
- **dbi:gnumeric:fname.sheet** (Force Gnumeric interpretation)

MDB: Access database format (via Mdbtools, READ-ONLY)

- **.mdb:** Access database file
- **.json:** {
 "type": "access", // *Access family*
 "file": "fname.db" // *File name*
}
- **dbi:access:fname.db** (Force Access interpretation)

JMDB: Access database format (via Jackcess)

- **.mdb:** Access database file

- **.json:** {
 "type": "access", // *Jackcess/access family*
 "file": "fname.db" // *File name*
 }
- **dbi:jackcess:fname.db** (Force Access interpretation)

MYSQL: database connector

- **.json:** {
 "type": "mysql", // *MYSQL connector*
 "database": "db_name", // *Database name*
 "host": "localhost", // *Host name*
 "port": "1111", // *Port number*
 "username": "root", // *Username*
 "password": "****" // *Password*
 }
- **dbi:mysql:database_name** (Use default port, username, etc)
- **dbi:mysql:database_name:username=USER:password=PASS**
- **dbi:mysql:database_name:host=HOST:port=PORT**

SOCIALCALC: SocialCalc format (via mozjs)

- **.socialcalc:** SocialCalc spreadsheet file
- **.sc:** SocialCalc spreadsheet file
- **.json:** {
 "type": "socialcalc", // *SocialCalc family*
 "file": "sheet.txt" // *File name*
 }
- **dbi:socialcalc:sheet.txt** (Force SocialCalc interpretation)

15.8 Version

ssresolve version 0.6.4

16 ssformat

Reformat tables/databases/spreadsheets.

16.1 Usage

- ssformat [options] FILE
- ssformat [options] FILE1 FILE2

16.2 Index

- [Option summary](#)
- [Option details](#)
- [Examples](#)
- [Patch formats](#)
- [Database/spreadsheet file formats](#)
- [Version](#)

16.3 Option summary

- [--default-table=TABLE](#)
- [--header](#)
- [--help](#)
- [--index](#)
- [--input-formats](#)
- [--paint](#)
- [--table=TABLE](#)

16.4 Option details

--default-table=TABLE

name to use when a table name is needed and not supplied

--header

extract column names only

--help

show how to use this program

--index

extract content of key columns only

--input-formats

list supported input database formats

--paint

add color highlighting appropriate for highlighter diffs

--table=TABLE

operate on a single named table of a workbook/database

16.5 Examples

You can generate test file(s) for the examples that follow:

```
ssformat --test-file numbers.csv  
ssformat --test-file numbers.sqlite
```

16.5.1 Example 1

```
ssformat numbers.csv numbers_converted.sqlite
```

Convert CSV format table to an Sqlite database table.

16.5.2 Example 2

```
ssformat numbers.sqlite numbers_converted.csv
```

Convert Sqlite database table to a CSV format table.

16.5.3 Example 3

```
ssformat numbers.sqlite -
```

Display contents of an Sqlite database table.

16.6 Patch formats

- **tdiff**: *[default]* reminiscent of the standard unix diff format for text
- **csv**: csv-compatible diff format
- **hilite**: colorful spreadsheet format
- **index**: tabular output showing relationship between rows and columns
- **novel**: mark all shared rows - remaining rows are unmatched
- **ops**: summarize modified rows in a table
- **raw**: verbose diff format for debugging
- **review**: spreadsheet diff format suitable for quickly accepting or rejecting changes
- **sql**: SQL format (data diffs only)
- **stats**: produce statistics on table changes

16.7 Database/spreadsheet file formats

CSV: plain-text delimiter-separated family of formats

- **.csv**: Comma-separated values
- **.tsv**: Tab-separated values

- **.ssv**: Semicolon-separated values
- **.json**: {
 "type": "csv", // *CSV family*
 "file": "fname.dsv", // *File name*
 "delimiter": "|" // *Delimiter character*
}

SQLITE: file-based database

- **.sqlite**: Sqlite database file
- **.json**: {
 "type": "sqlite", // *Sqlite family*
 "file": "fname.db" // *File name*
}
- **dbi:sqlite:fname.db** (Force sqlite interpretation)

SQLITEXT: sqlite-format sql dump

- **.sqlitext**: SQL dump of Sqlite database
- **.json**: {
 "type": "sqlitext", // *Sqlitext family*
 "file": "fname.sql" // *File name*
}
- **dbi:sqlitext:fname.sql** (Force sqlitext interpretation)

JSONBOOK: Spreadsheet formats in json

- **.jsonbook**: Json spreadsheet
- **.json**: {
 "type": "jsonbook", // *Json spreadsheet*
 "file": "fname.sheet" // *File name*
}
- **dbi:jsonbook:fname.sheet** (Force Json spreadsheet interpretation)

GNUMERIC: Spreadsheet formats (via gnumeric)

- **.xls**: Excel spreadsheet
- **.xlsx**: Excel spreadsheet
- **.gnumeric**: Gnumeric spreadsheet
- **.ods**: OpenOffice/LibreOffice spreadsheet

- **.json:** {
 "type": "gnumeric", // *Gnumeric family*
 "file": "fname.sheet" // *File name*
 }
 • **dbi:gnumeric:fname.sheet** (Force Gnumeric interpretation)

MDB: Access database format (via Mdbtools, READ-ONLY)

- **.mdb:** Access database file
- **.json:** {
 "type": "access", // *Access family*
 "file": "fname.db" // *File name*
 }
 • **dbi:access:fname.db** (Force Access interpretation)

JMDB: Access database format (via Jackcess)

- **.mdb:** Access database file
- **.json:** {
 "type": "access", // *Jackcess/access family*
 "file": "fname.db" // *File name*
 }
 • **dbi:jackcess:fname.db** (Force Access interpretation)

MYSQL: database connector

- **.json:** {
 "type": "mysql", // *MYSQL connector*
 "database": "db_name", // *Database name*
 "host": "localhost", // *Host name*
 "port": "1111", // *Port number*
 "username": "root", // *Username*
 "password": "****" // *Password*
 }
 • **dbi:mysql:database_name** (Use default port, username, etc)
 • **dbi:mysql:database_name:username=USER:password=PASS**
 • **dbi:mysql:database_name:host=HOST:port=PORT**

SOCIALCALC: SocialCalc format (via mozjs)

- **.socialcalc:** SocialCalc spreadsheet file

- **.sc**: SocialCalc spreadsheet file
- **.json**: {
 "type": "socialcalc", // *SocialCalc family*
 "file": "sheet.txt" // *File name*
}
- **dbi:socialcalc:sheet.txt** (Force SocialCalc interpretation)

16.8 Version

ssformat version 0.6.4

17 ssrediff

Reformat a tabular diff file.

Converting to hilite/review formats will require supplying the original spreadsheet/database. Not every diff format supported as output by ssdiff can be read as input by ssrediff.

17.1 Usage

- ssrediff [options] PATCHFILE
- ssrediff [options] DATAFILE PATCHFILE

17.2 Index

- [Option summary](#)
- [Option details](#)
- [Examples](#)
- [Patch formats](#)
- [Database/spreadsheet file formats](#)
- [Version](#)

17.3 Option summary

- [-act=ACT](#)
- [-context=N](#)
- [-default-table=TABLE](#)
- [-format=FORMAT](#)
- [-help](#)
- [-low-memory](#)

- `-neither`
- `-omit-format-name`
- `-omit-sheet-name`
- `-ours`
- `-output=OUTPUTFILE`
- `-patch-formats`
- `-table=TABLE`
- `-theirs`
- `-variant=VARIANT`

17.4 Option details

`-act=ACT`

filter for an action of a particular type (update, insert, delete, none, schema)

`-context=N`

Number of rows of context before and after changes for highlighter diffs ("all" to include all rows)

`-default-table=TABLE`

name to use when a table name is needed and not supplied

`-format=FORMAT`

set difference format for output

`-help`

show how to use this program

`-low-memory`

prioritize low memory usage over speed

`-neither`

in case of conflict use cell value from common ancestor

`-omit-format-name`

omit any version-dependent header from diff

`-omit-sheet-name`

omit any sheet/table name from diff

`-ours`

in case of conflict use cell value that was the local choice

`-output=OUTPUTFILE`

direct output to this file (default is standard output)

`-patch-formats`

list supported patch formats

`-table=TABLE`

filter for a named table of a workbook/database (repeat option for multiple tables)

–theirs

in case of conflict use cell value that wasn't the local choice

–variant=VARIANT

set the desired dialect when using a poorly defined output format (currently for SQL, available variants are: sqlite, access)

17.5 Examples

You can generate test file(s) for the examples that follow:

```
ssrediff --test-file numbers.csv
ssrediff --test-file numbers_buggy.csv
```

17.5.1 Example 1

```
ssrediff --format sql numbers_patch.tdiff
```

Convert tdiff format file to SQL

17.5.2 Example 2

```
ssrediff --format sql --act update numbers_patch.tdiff
```

Convert tdiff format file to SQL, showing updates only

17.5.3 Example 3

```
ssrediff --format csv numbers_patch.tdiff
```

Convert tdiff format file to a CSV-readable diff format

17.5.4 Example 4

```
ssrediff --format hilite --output review.csv numbers_buggy.csv numbers_patch.tdiff
```

Generate tabular form of diff for eyeballing. If ssrediff is compiled with gnumeric support, and output format is *.xls, color highlighting is added.

17.6 Patch formats

- **tdiff**: *[default]* reminiscent of the standard unix diff format for text
- **csv**: csv-compatible diff format
- **hilite**: colorful spreadsheet format
- **index**: tabular output showing relationship between rows and columns

- **novel**: mark all shared rows - remaining rows are unmatched
- **ops**: summarize modified rows in a table
- **raw**: verbose diff format for debugging
- **review**: spreadsheet diff format suitable for quickly accepting or rejecting changes
- **sql**: SQL format (data diffs only)
- **stats**: produce statistics on table changes

17.7 Database/spreadsheet file formats

CSV: plain-text delimiter-separated family of formats

- **.csv**: Comma-separated values
- **.tsv**: Tab-separated values
- **.ssv**: Semicolon-separated values
- **.json**: {
 "type": "csv", // *CSV family*
 "file": "fname.dsv", // *File name*
 "delimiter": "|" // *Delimiter character*
 }

SQLITE: file-based database

- **.sqlite**: Sqlite database file
- **.json**: {
 "type": "sqlite", // *Sqlite family*
 "file": "fname.db" // *File name*
 }
- **dbi:sqlite:fname.db** (Force sqlite interpretation)

SQLITEXT: sqlite-format sql dump

- **.sqlitext**: SQL dump of Sqlite database
- **.json**: {
 "type": "sqlitext", // *Sqlitext family*
 "file": "fname.sql" // *File name*
 }
- **dbi:sqlitext:fname.sql** (Force sqlitext interpretation)

JSONBOOK: Spreadsheet formats in json

- **.jsonbook**: Json spreadsheet

- **.json:** {
 "type": "jsonbook", // *Json spreadsheet*
 "file": "fname.sheet" // *File name*
}
- **dbi:jsonbook:fname.sheet** (Force Json spreadsheet interpretation)

GNUMERIC: Spreadsheet formats (via gnumeric)

- **.xls:** Excel spreadsheet
- **.xlsx:** Excel spreadsheet
- **.gnumeric:** Gnumeric spreadsheet
- **.ods:** OpenOffice/LibreOffice spreadsheet
- **.json:** {
 "type": "gnumeric", // *Gnumeric family*
 "file": "fname.sheet" // *File name*
}
- **dbi:gnumeric:fname.sheet** (Force Gnumeric interpretation)

MDB: Access database format (via Mdbtools, READ-ONLY)

- **.mdb:** Access database file
- **.json:** {
 "type": "access", // *Access family*
 "file": "fname.db" // *File name*
}
- **dbi:access:fname.db** (Force Access interpretation)

JMDB: Access database format (via Jackcess)

- **.mdb:** Access database file
- **.json:** {
 "type": "access", // *Jackcess/access family*
 "file": "fname.db" // *File name*
}
- **dbi:jackcess:fname.db** (Force Access interpretation)

MYSQL: database connector

- **.json:** {
 "type": "mysql", // *MYSQL connector*
 "database": "db_name", // *Database name*
 "host": "localhost", // *Host name*
 "port": "1111", // *Port number*
 "username": "root", // *Username*
 "password": "****" // *Password*
 }
- **dbi:mysql:database_name** (Use default port, username, etc)
- **dbi:mysql:database_name:username=USER:password=PASS**
- **dbi:mysql:database_name:host=HOST:port=PORT**

SOCIALCALC: SocialCalc format (via mozjs)

- **.socialcalc:** SocialCalc spreadsheet file
- **.sc:** SocialCalc spreadsheet file
- **.json:** {
 "type": "socialcalc", // *SocialCalc family*
 "file": "sheet.txt" // *File name*
 }
- **dbi:socialcalc:sheet.txt** (Force SocialCalc interpretation)

17.8 Version

ssrediff version 0.6.4

18 ssfossil

This is a thin wrapper around the [fossil](#) version control system.

It modifies the **merge** operation within fossil to use the algorithm behind the [ssmerge](#) command.

For usage information, see the [fossil reference](#).

The ssfossil client can interoperate with fossil used as a server, or vice versa.

19 coopy

Manage a repository of spreadsheets and databases.

Usually run without options, for a graphical interface.

19.1 Usage

- `coopy [options]`
- `coopy [options] DIRECTORY`

19.2 Index

- [Option summary](#)
- [Option details](#)
- [Examples](#)
- [Version](#)

19.3 Option summary

- [-add=FILE](#)
- [-clone=URL](#)
- [-export=FILE](#)
- [-gui](#)
- [-help](#)
- [-key=KEY](#)
- [-message=MESSAGE](#)
- [-new](#)
- [-pull](#)
- [-push](#)
- [-silent](#)

19.4 Option details

-add=FILE

attach the given spreadsheet/database to the repository

-clone=URL

clone the given repository

-export=FILE

export the given spreadsheet/database from the repository

-gui

force GUI to be shown

-help

show how to use this program

-key=KEY

use specified key when adding or exporting a spreadsheet/database

-message=MESSAGE

use the specified message as a log entry

-new

create a new, empty repository

-pull

pull in data from remote repository to local clone

-push

push out data to remote repository from local clone

-silent

keep output to a minimum

19.5 Examples

19.5.1 Example 1

```
coopy
```

run the coopy GUI from the current directory. All the actions in these examples can be achieved from the GUI.

19.5.2 Example 2

```
coopy --new
```

create a new empty repository in the current directory.

19.5.3 Example 3

```
coopy --key=people --add=people.xls
```

add people.xls to the repository, with key 'people'.

19.5.4 Example 4

```
coopy --key=orgs --export=organizations.sqlite
```

export organizations.sqlite from the repository, with key 'orgs'.

19.6 Version

coopy version 0.6.4

20 Related projects

(limited to free and open source projects)

- [datacouch](#), turn spreadsheets into APIs, fork datasets.
- [google-refine](#), a power tool for working with messy data.
- [diffkit](#), diff for tables, targets enterprises.
- [google-diff-match-patch](#), for synchronizing plain text.
- [SpreadSheet Compare](#), an Excel plugin.

21 What are all these funny files in a Coopy directory?

There are some special files that fossil uses to store information about a repository.

They are:

- *FOSSIL*: this file identifies a directory as a fossil repository.
- *manifest*: this is a list of all managed files in the repository.
- *manifest.uuid*: this is a unique identifier. In addition, coopy will place this file in the same directory:
- *repository.coopy*: this is a database containing information about the repository. In fossil terms, this is the "real" repository, and the rest is just a "view" or "source tree". So as not to confuse people who have not used a version control system before, the coopy program does not emphasize this distinction. The fossil or [ssfossil](#) program can be used to create multiple views of the same fossil repository.

22 Diff format

- [TDIFF tabular diff format](#) - this is the main supported format. It has the aesthetics of classic diffs.
- [Highlighter format](#) - a visual diff format suitable for presentation and review in a spreadsheet program.
- [CSV DTBL diff format](#) - this is another supported format. It is a CSV readable file.
- *patch_format_csv_v_0_2* - an older format, deprecated but still supported.
- Raw format - a verbose format useful for debugging. Cannot be read for patching.

23 TDIFF tabular diff format

Version

0.3

This page describes the TDIFF diff format. This is the main format to be used by the COOPY toolbox for representing differences between tables. It is based on a draft specification with Joe Panico, see [version history](#).

23.1 General structure

TDIFF documents use the UTF-8 character encoding.

TDIFF documents comprise any number of [comment blocks](#), [control blocks](#), and [diff hunks](#), interleaved in any order. Each diff hunk describes a related set of differences between two tables. Each hunk could stand on its own as an independent TDIFF document. When there is a choice in how to decompose differences between two tables as a sequence of hunks, generators are encouraged to choose a decomposition that minimizes ordering effects between hunks.

Example:

```
# tdiff version 0.3

/*
 * fix some goofs
 */
* |bridge=Brooklyn|designer:'J.A.Roebling'|length:1595|
= |bridge=Williamsburg|designer:'D.D.Duck'-'>'L.L.Buck'|length:1600|
* |bridge=Queensborough|designer:'Palmer & Hornbostel'|length:1182|

/*
 * remove spam and add a missing bridge
 */

- |bridge=Spamspan|designer:'S.Spamington'|length:10000|
+ |bridge=Manhattan|designer:'G.Lindenthal'|length:1470|

/*
 * we are done!
 */
```

23.2 Comment blocks

Comment blocks are delimited using:

```
/* */
```

(C style). Any content can occur within a comment block. Examples:

```
/* This is an example single-line comment */
/* This is an
   example multi-line
   comment */
```

23.3 Control blocks

Control blocks begin with "# ", and are delimited by a newline or linefeed. Control blocks may hold meta information about diffs, or environmental information that might be useful to an interpreter. Apart from the special [header control block](#), they lie outside of the scope of this specification.

23.4 Header control block

A TDIFF document should begin with a special [control block](#) called the header. The header begins with the characters "# tdiff". It is there to aid in rapid identification of tdiff documents. Example:

```
# tdiff version 0.3
```

23.5 Hunk

A hunk is a series of one or more adjacent [diff lines](#), optionally preceded by a [column line](#), where each diff line represents the differences between the source tables for a single row. The lines within a hunk should be separated by only the newline characters that terminate each diff line, so that they all appear as adjacent lines within a text editor. Within a TDIFF document, hunks are delimited from each other via intermediate filler or comment blocks.

Example hunk:

```
- |bridge=Spamspan|designer:'S.Spamington'|length:10000|
= |bridge=Williamsburg|designer:'D.D.Duck'->'L.L.Buck'|length:1600|
+ |bridge=Manhattan|designer:'G.Lindenthal'|length:1470|
```

23.6 Diff line

A diff line describes differences in a single row of the two tables that were compared. One table is designated the left or local table (called **L**) and the other table is designated the right or remote table (called **R**).

There are three types of diff lines:

- **MISSING** line: describes a row that is present in **L** but absent in **R**.
- **EXTRA** line: describes a row that is absent in **L** but present in **R**.
- **CHANGE** line: describes a row that is present in both tables, but differs in some specific column values.

Each diff line occupies its own line in the document, and begins with one of three characters. These three characters are called "line type" characters:

- **MISSING** lines begin with plus: '+'. In order to make **R** look like **L** we would have to add the missing row to **R**.
- **EXTRA** lines begin with minus: '-'. In order to make **R** look like **L** we would remove the extra row from **R**.
- **CHANGE** lines begin with equals: '='. In order to make **R** look like **L** we would update some of its column values.

The line type character can be left or right padded with any amount of whitespace, for readability. The line type character is followed by any number of name-value pairs, where the names represent column names, and the values are the values for the corresponding column name in that particular row. The name is separated from the value by an equals ('=') sign for identifying columns (usually part of the primary key, but see [Keys versus identity](#)) or a colon (':') sign for all other columns. The name-value pairs, as well as the line type character, are delimited by a pipe '|' character.

Example diff line:

```
= |bridge=Williamsburg|designer:'D.D.Duck'->'L.L.Buck'|length:1600|
```

23.7 Column line

Optionally, key names can be "factored out" of diff lines and placed in a special column line. A column line lists column names, followed by "=" for identifying columns. New columns that were not present in the input should have "->" appended, to flag that cells in such columns have no prior values.

Here's a column line example:

```
@ |bridge=|designer|length|
```

This establishes bridge as an identifying column that appears first, followed by designer and length columns. We can now rewrite this:

```
= |bridge=Williamsburg|designer:'D.D.Duck'->'L.L.Buck'|length:1600|
```

as this:

```
= |Williamsburg|'D.D.Duck'->'L.L.Buck'|1600|
```

The effect of column lines should be limited to within a single hunk.

In the case of column diffs, for each cell that was different between **L** and **R**, both the old and new values are displayed. The old value must come first, followed by '->' (dash greater than), followed by the new value. For all three diff line types, the generator may include **L** name-value pairs that are not strictly needed, but may help with row identification.

23.8 Keys versus identity

Determining whether a row is present in **L** and **R** requires a judgment about row identity. This judgment may be simple. For example, the identity of a row may simply be the value of its primary key. However, it is possible that the identity of a row is distinct from its primary key. Consider for example a table with an auto-incrementing integer primary key, rather than something derived from the row data. Comparison of that key between separately maintained copies of that table will be meaningless. For meaningful comparison, an alternate row identity would need to be constructed.

This issue lies outside the TDIFF specification, but it is important that implementors be aware that columns used for identification may or may not be part of the primary key.

23.9 Appendix: Quoting

Names or values may be quoted in a TDIFF document. Quoting is done as follows:

- All instances of the single-quote character are duplicated into pairs.
- All control characters and the backslash character are escaped as for C literals.
- The name or value is wrapped in single-quotes

It is always safe to single-quote a name or value. Names or values *must* be quoted in any of the following conditions:

- A name or value conflicts with the reserved word: NULL
- A name or value contains any character in the 7-bit ASCII range that is *not* in the following set: [A-Za-z0-9+.]
- A *name* begins with any of the characters [0-9+..].

23.10 Appendix: Grammar

```
document ::= header ((space)? block)*
block ::= hunk | control | comment | filler

hunk ::= (hunk_header)? ((space)? diff_line)+
hunk_header ::= '@' (space)+ divider (column divider)* break
diff_line ::= ('-' | '+' | '=') (space)+ divider (term divider)* break
term ::= (name ('='|':'))? (value '->')? value
column ::= name ('='|'->')?
break ::= divider? linebreak

control ::= "#" (divider value)* break

comment ::= ("/" comment_body "/" )
```

```

filler ::= (linebreak | divider)+

header ::= "# tdiff" ([^\r\n])* break

divider ::= '|'
linebreak ::= ('\r\n' | '\r' | '\n')

```

The linebreak non-terminal needs to be handled more carefully than the grammar suggests, since the number of linebreaks is significant in the grammar. The comment_body non-terminal is as for the C language.

23.11 Appendix: Complete examples

In example one, both tables are in an RDBMS, both tables have the same column names, and the rows are identified using column1.

Example 1:

L:	column1, column2, column3, column4	R:	column1, column2, column3, column4
1,	0000, x, aaaa	-----	
3,	2222, x, aaaa	2,	1111, x, aaaa
4,	3333, x, aaaa	3,	2222, y, aaaa
5,	4444, x, aaaa	4,	0000, z, bbbb
6,	5555, x, aaaa	5,	4444, z, bbbb
-----		6,	5555, u, aaaa
-----		7,	0000, v, aaaa
-----		8,	1111, x, aaaa

Example 1 diff, variant 1:

```

# tdiff version 0.3

/*
 * this is the tDiff document for example 1, using 1 hunk only and no context.
 * Note the "|" usage varies from previous examples in this document.
 * "|" plays the same role as spaces and tabs in the spec, so varying
 * styles are possible.
 */
- | column1=1
+ | column1=2| column2:1111| column3:x| column4:aaaa
= | column1=3| column3:x->y
= | column1=4| column2:3333->0000| column3:x->z| column4:aaaa->bbbb
= | column1=5| column3:x->z| column4:aaaa->bbbb
= | column1=6| column3:x->u
+ | column1=7| column2:0000| column3:v| column4:aaaa
+ | column1=8| column2:1111| column3:x| column4:aaaa
/*
 * end of tDiff document
 */

```

Example 1 diff, variant 2:

```

# tdiff version 0.2

/*
 * here is a tDiff document that is equivalent to the document above, except
 * that it uses 8 hunks, more comments, and adds in some context
 */
/*
 * hunk 1: notice that columns 2,3,4 are context-- not strictly necessary

```

```

* to specify a remove
*/
- | column1=1| column2:0000| column3:x| column4:aaaa

/*
* hunk 2: notice that the hunks are separated by standalone newline
*/
+ | column1=2| column2:1111| column3:x| column4:aaaa

/*
* hunk 3: notice that column2 and column4 are merely context
*/
= | column1=3| column2:2222| column3:x->y| column4:aaaa

/*
* hunk 4: notice that the column diff line is surrounded by context rows, and
* that the context rows describe the values on the RHS.
*/
* | column1=3| column2:2222| column3:x| column4:aaaa
= | column1=4| column2:3333->0000| column3:x->z| column4:aaaa->bbbb
* | column1=5| column2:4444| column3:x| column4:aaaa

/*
* hunk 5
*/
= | column1=5| column3:x->z| column4:aaaa->bbbb

/*
* hunk 6
*/
= | column1=6| column3:x->u

/*
* hunk 7
*/
+ | column1=7| column2:0000| column3:v| column4:aaaa

/*
* hunk 8
*/
+ | column1=8| column2:1111| column3:x| column4:aaaa

```

23.12 Appendix: Version history

TDIFF version 0.2 was co-developed by COOPY author Paul Fitzpatrick and [diffkit](#) author Joe Panico ([TDIFF 0.2 draft](#), [diffkit-user thread](#)). Version 0.3 contains extensions to deal with schema changes and the like, and hasn't been sanity checked by Joe.

Differences between version 0.2 and 0.3:

- 0.3 drops specification of ROW pseudo column
- 0.3 drops specification of Column Numbers
- 0.3 drops specification of Context

24 CSV DTBL diff format

Version

0.5

This page describes the diff format (called DTBL) generated by "ssdiff –format-csv". This is the main format used in the COOPY toolbox for representing differences between tables. The diff format is itself tabular, so that diff files can be loaded by spreadsheet and database tools without writing a special parser. Within this document, we assume the diff is represented as a CSV file. Some simplification is possible if the diff can be stored directly in a format that allows NULLs and other types, see [Representing NULLs and other types](#).

24.1 General structure

A diff file consists of a series of lines in CSV format, following [RFC4180](#) (see [CSV details](#)). The first two cells should evaluate to the strings "dtbl" and "csv" (see [First row](#)).

Here's an example:

```
dtbl,csv,version,0.5
column,name,number,digits
column,move,digits,number
```

In words, this would mean:

- Start with a table whose columns are called "number" and "digits".
- Reorder the columns, so the "number" column comes after "digits".

The width of the diff file in cells has no significance. Extra empty columns can be added to the end of rows without affecting the interpretation of the diff.

24.2 First row

The first row contains at least four non-empty cells.

- The first cell of the first row should have the value "dtbl" (a four-character string). The representation of that value may be quoted or unquoted.
- The second cell of the first row should have the value "csv" (a three-character string). The representation of that value may be quoted or unquoted.
- The third cell of the first row should have the value "version" (a seven-character string). The representation of that value may be quoted or unquoted.
- The fourth cell of the first row may have any non-empty value. The representation of that value may be quoted or unquoted.

This means that a CSV format diff file will begin with one of the following sequence of bytes (ascii/utf8 encoding):

- dtbl,csv,...
- "dtbl","csv",...
- "dtbl",csv,...
- dtbl,"csv"...

This set is effectively the format's "magic number".

Here's an example of a first row:

```
dtbl,csv,version,0.5,, ,
```

24.3 Config rows

The first row may be followed by a series of rows, each of which has the value "config" in its first cell. The second and third cells on such rows are interpreted as key-value pairs. The values of keys may affect the interpretation of the rest of the diff. For example:

```
config,param1,val1,,
```

There are currently no specified configuration parameters. This functionality is reserved for possible future use.

24.4 Operation rows

The remaining rows describe properties and transformations of a table. These rows are called "operations". Operations assume that the rows of the diff file are considered in order from the beginning of the file to the end.

The values of the first and second cells may be:

- [table name](#)
- [column name](#)
- [column insert](#)
- [column delete](#)
- [column move](#)
- [column rename](#)
- [link name](#)
- [link act](#)
- [row update](#)
- [row select](#)
- [row insert](#)
- [row delete](#)

24.5 table name

Specifies the table/sheet name. Useful for working with multi-table differences and patches.

```
table,name,"Outcome Measurement",,,
```

24.6 column name

This gives arbitrary labels for the columns in the table. The labels are those used to refer to the columns within the patch file. Trailing blank cells are ignored. For example:

```
column,name,number,digits,,,
```

This specifies that two columns are expected, and those columns are labeled "number" and "digits".

Note that for some tables, such as those from CSV files, column names may be arbitrary or "guessed". This should be borne in mind in tools that apply diffs as patches to tables.

A "column name" row is mandatory if there are operations on columns to be performed ("column move", "column insert", or "column delete"). Otherwise, it is optional.

24.7 column move

This operation specifies the movement of a single column. The move is specified by giving the resultant order of columns after the move. For example:

```
column,name,[0],[1],,
column,move,[1],[0],,
```

All operations of the form "column *" are specified by giving the resultant order of columns after that operation.

24.8 column insert

This operation specifies the insertion of a single column. It has the same format as a "column move" row. The point of insertion is specified by giving the resulting column order, with a new column label inserted appropriately. Example:

```
column,name,[0],[1],,
column,insert,[0],[4],[1],
```

All operations of the form "column *" are specified by giving the resultant order of columns after that operation.

24.9 column delete

This operation specifies the removal of a single column. It has the same format as the "column move" row. The point of insertion is specified by giving the resulting column order, with the unwanted column removed. Example:

```
column,name,{3},[1],[0],[4]
column,delete,{3},[0],[4],
```

All operations of the form "column *" are specified by giving the resultant order of columns after that operation.

24.10 column rename

This operation specifies that a single column should be renamed. The specification is given by repeating the column names, with one name changed. For example:

```
column,name,number,digits
column,rename,number,digit
```

All operations of the form "column *" are specified by giving the resultant order of columns after that operation.

24.11 link name

To allow tidy descriptions of operations on rows in tabular form (see the "row *" operations), the DTBL format allows for flexibility in the **roles** assigned to diff columns. That role can be controlled using [link name](#) and [link act](#). The role of a diff column is:

- The **data column** it is associated with, if any. Data columns are columns in the table being transformed, whereas diff columns are columns in the diff file itself.
- The activity the diff column is associated with (such as matching or assigning data).

The first two diff columns are never associated with data columns; they instead contain operation tags such as "link name". The remaining diff columns may be associated with data columns as follows.

After any "column *" operation, the association of patch columns is set to match one-to-one where they appear in that row. For example, after:

```
column,move,digits,number
```

the third patch column is associated with the data column called "digits", and the fourth patch column is associated with the data column called "number".

It is possible to change the association using the "link name" command. For example, after:

```
link,name,number,digits,digits
```

the third patch column is associated with the data column called "number", and the fourth and fifth patch columns are associated with the data column called "digits". It is acceptable for multiple patch columns to be associated with a single data column, since the those patch columns may have distinct actions (see [link act](#)). It is also acceptable to have patch columns associated with non-existent/virtual data columns. The interpretation of such columns must be resolved with a [link act](#) operation.

24.12 link act

This operation controls the action associated with patch columns. It assumes that associations between patch columns and data columns have already been set up (see [link name](#)). For example:

```
link,name,number,digits,digits
link,act,*,*,=
```

This associates the action "match" (or "*") with the third and fourth column, and the action "assign" (or "=") with the fifth column. Here's another example:

```
link,name,number,digits
link,act,*,*=
```

This associates the action "match" (or "*") with the third column, and the actions "match" and "assign" (or "*=") with the fourth column.

See [row update](#) and [row select](#) to understand why this kind of flexibility is useful.

Possible actions are:

- "": none - no action, ignore this column.
- "*": match - match against the data in this column.
- "=": assign - assign data in this column.
- "*=": match and assign.
- "ROW": row - match a row identifier against the data in this column. The data column association is ignored in this case.
- "#": context - no action, ignore this column. However, the column will contain data just as for "*" for the benefit of a reader.

24.13 row update

Modifies the values of a row. If not preceded by a [row select](#), it will select a row to modify using the same rules as [row select](#). It then makes changes according to the assignment actions in effect (see [link act](#)). Here's an example which would change a row whose first column contains "Paul" such that the second column becomes "Fitzpatrick":

```
column,name,first,last,age
link,act,*,=,
row,update,Paul,Fitzpatrick,
```

Here's an alternate example using [row select](#).

```
column,name,first,last,age
link,act,*,=,
row,select,Paul,,
row,update,,Fitzpatrick,
```

Here's an example that would select a row whose first column is "Paul" and change "Paul" to be "Peter".

```
column,name,first,last,age
link,name,first,first,
link,act,*,=,
row,update,Paul,Peter,
```

Here's an alternate example to achieve the same result:

```
column,name,first,last,age
link,act,*,=,
row,select,Paul,,
row,assign,Peter,,
```

24.14 row select

Selects a row matching the given values. Matches are made based on the actions linked with diff columns (see [link act](#)). This example selects a row whose first column is "Paul":

```
column,name,first,last,age
link,act,*,,,
row,select,Paul,,
```

The selection is stored until the next [row update](#), [row insert](#), or [row delete](#).

24.15 row insert

Inserts a row with the given values. Some table formats allow control over row order. In this case, if an insertion is preceded by a [row select](#), the insertion is done before the selected row, otherwise it is done after the last existing row. Values are specified as for [row update](#).

24.16 row delete

Deletes a row matching the given values. Values are specified as for [row select](#).

24.17 CSV details

Since the CSV format has some variability, here are some notes on the variant expected. In summary, keep to [RFC4180](#) as closely as you can.

- If a cell begins with a double quote, the extent of the cell is determined by quoting rules. It is important to note that a cell may extend across line boundaries. The cell will extend to the next unpaired double quote in the patch. A pair of double quotes ("") found by scanning forwards through the file is interpreted as the double quote character, and not used to delimit the cell's extent. Once the extent of a cell is determined, if a double quote is present at the beginning of that cell, it is stripped, along with the matching quote at the end.
- Otherwise, a cell extends until a comma, CR, or LF character, or the end of the file.
- An unquoted CRLF sequence is treated as a single delimiter.
- Cells should be quoted if they contain any whitespace or the comma character.

24.18 Representing NULLs and other types

The DTBL format, as represented in CSV, needs a way to describe all cell types as text. So far, only the issue of representing "NULL" has been addressed. The rule adopted is as follows:

- the four-letter string "NULL" is interpreted as a cell whose value is NULL (in the database sense).
- the five-letter string "_NULL" is interpreted as a cell whose value is the four-letter string "NULL".
- the six-letter string "__NULL" is interpreted as a cell whose value is the five-letter string "_NULL".
- ...

There are other possibilities, such as distinguishing between a completely blank cell and a cell containing a pair of quotes, but such possibilities are unlikely to survive a round-trip through a graphical CSV editor and so could cause more problems than they solve.

24.19 Pending issues

- Control of row order (when possible) is not fully resolved.
- Typing of columns or cells is not resolved, especially for newly inserted columns. For existing columns/cells, it is not so big a problem, if the diff is interpreted in a type-preserving way.
- Comparing e.g. date/times is currently prone to differences in representation causing false mismatches.
- This list of pending issues has not been fully fleshed out.

25 Supported table formats

Programs in the COOPY toolbox can work with tables represented in the following formats:

- CSV (or other delimiter-separated formats).
- Sqlite tables.
- MySQL tables (partial support at the time of writing).

- Spreadsheet sheets (.xls, .xlsx, .ods, etc).

CSV support is most complete, followed by Sqlite, followed by Gnumeric, followed by MySQL. Good support for all four formats is targeted for version 1.0 of the toolbox.

25.1 Specifying sources

The COOPY tools expects to work with files as inputs and outputs. For example:

```
ssdiff ver1.sqlite ver2.sqlite
```

To use a database as an input or output, or to add configuration options on how a file should be interpreted, more information needs to be supplied. This can be done in two ways. One is to use a "dbi:*" string. For example:

```
ssdiff ver1.sqlite dbi:mysql:ver2:username=root
```

This would compare the sqlite database in "ver1.sqlite" with the mysql database called "ver2". Alternatively, the extra information can be placed in a "proxy" file with a .json extension, in the **JSON** format. Here is a file users.json to refer to a table called "users" in a mysql database called "db":

```
{
  "type": "mysql",
  "database": "db",
  "username": "testy",
  "password": "*****",
  "host": "127.0.0.1",
  "port": 3333,
  "table": "users"
}
```

More example dbi strings:

```
dbi:csv:file=hello          # file "hello" is in csv format
dbi:csv::file=hello:there    # file "hello:there" is in csv format
# the use of "::" means that the following key/value pair is the
# last one, and ":" characters should no longer be used as a divider.
```