Getting Started with Beancount

Martin Blais, July 2014

http://furius.ca/beancount/doc/getting-started

Introduction

This document is a gentle guide to creating your first Beancount file, initializing it with some options, some guidelines for how to organize your file, and instructions for declaring accounts and making sure their initial balance does not raise errors. It also contains some material on configuring the Emacs text editor, if you use that.

You will probably want to have read some of the User's Manual first in order to familiarize yourself with the syntax and kinds of available directives, or move on to the Cookbook if you've already setup a file or know how to do that. If you're familiar with Ledger, you may want to read up on the differences between Ledger and Beancount first.

Editor Support

Beancount ledgers are simple text files. You can use any text editor to compose your input file. However, a good text editor which understands enough of the Beancount syntax to offer focused facilities like syntax highlighting, autocompletion, and automatic indentation highly has the potential to greatly increase productivity in compiling and maintaining your ledger.

Emacs

Support for editing Beancount ledger files in Emacs was traditionally distributed with Beancount. It now lives as its own project in this Github repository.

Vim

Support for editing Beancount ledger files in Vim has been implemented by Nathan Grigg and is available in this Github repository.

Sublime

Support for editing with Sublime has been contributed by Martin Andreas Andersen and is available in this github repository or as a Sublime package here.

VSCode

There are a number of plugins for working with Beancount text files including VSCode-Beancount by Lencerf.

Creating your First Input File

To get started, let's create a minimal input file with two accounts and a single transaction. Enter or copy the following input to a text file:

```
2014-01-01 open Assets:Checking
2014-01-01 open Equity:Opening-Balances

2014-01-02 * "Deposit"
  Assets:Checking           100.00 USD
  Equity:Opening-Balances
```

Brief Syntax Overview

A few notes and an ultra brief overview of the Beancount syntax:

- Currencies must be entirely in capital letters (allowing numbers and some special characters, like "_" or "-"). Currency symbols (such as $ or €) are not supported.
- Account names do not admit spaces (though you can use dashes), and must have at least two components, separated by colons. Each component of an account name must begin with a capital letter or number.
- Description strings must be quoted, like this: `"AMEX PMNT"`.
- Dates are only parsed in ISO8601 format, that is, YYYY-MM-DD.
- Tags must begin with "#", and links with "^".

For a complete description of the syntax, visit the User's Manual.

Validating your File

The purpose of Beancount is to produce reports from your input file (either on the console or serve via its web interface). However, there is a tool that you can use to simply load its contents and make some validation checks on it, to ensure that your input does not contain errors. Beancount can be quite strict; this is a tool that you use while you're entering your data to ensure that your input file is legal. The tool is called "bean-check" and you invoke it like this:

```
bean-check /path/to/your/file.beancount
```

Try it now on the file you just created from the previous section. It should exit with no output. If there are errors, they will be printed on the console. The errors are printed out in a format that Emacs recognizes by default, so you can use Emacs' `next-error` and `previous-error` built-in functions to move the cursor to the location of the problem.

Viewing the Web Interface

A convenient way to view reports is to bring up the "bean-web" tool on your input file. Try it:

```
bean-web /path/to/your/file.beancount
```

You can then point a web browser to http://localhost:8080 and click your way around the various reports generated by Beancount. You can then modify the input file and reload the web page your browser is pointing to—bean-web will automatically reload the file contents.

At this point, you should probably read some of the Language Syntax document.

How to Organize your File

In this section we provide general guidelines for how to organize your file. This assumes you've read the Language Syntax document.

Preamble to your Input File

I recommend that you begin with just a single file[1]. My file has a header that tells Emacs what mode to open the file with, followed by some common options:

```
;; -*- mode: beancount; coding: utf-8; fill-column: 400; -*-
option "title" "My Personal Ledger"
option "operating_currency" "USD"
option "operating_currency" "CAD"
```

The title option is used in reports. The list of "operating currencies" identify those commodities which you use most commonly as "currencies" and which warrant rendering in their own dedicated columns in reports (this declaration has no other effect on the behavior of any of the calculations).

[1] It is tempting to want to break down a large file into many smaller ones, but especially at first, the convenience of having everything in a single place is great.

Sections & Declaring Accounts

I like to organize my input file in sections that correspond to each real-world account. Each section defines all the accounts related to this real-world account by using an Open directive. For example, this is a checking account:

```
2007-02-01 open Assets:US:BofA:Savings              USD
2007-02-01 open Income:US:BofA:Savings:Interest     USD
```

I like to declare the currency constraints as much as possible, to avoid mistakes. Also, note how I declare an income account specific to this account. This helps break down income in reporting for taxes, as you will likely receive a tax document in relation to that specific account's income (in the US this would be a 1099-INT form produced by your bank).

Here's what the opening accounts might look like for an investment account:

```
2012-03-01 open Assets:US:Etrade:Main:Cash            USD
2012-03-01 open Assets:US:Etrade:Main:ITOT            ITOT
2012-03-01 open Assets:US:Etrade:Main:IXUS            IXUS
2012-03-01 open Assets:US:Etrade:Main:IEFA            IEFA
2012-03-01 open Income:US:Etrade:Main:Interest        USD
2012-03-01 open Income:US:Etrade:Main:PnL             USD
2012-03-01 open Income:US:Etrade:Main:Dividend        USD
2012-03-01 open Income:US:Etrade:Main:DividendNoTax   USD
```

The point is that all these accounts are related somehow. The various sections of the cookbook will describe the set of accounts suggested to create for each section.

Not all sections have to be that way. For example, I have the following sections:

- **Eternal accounts.** I have a section at the top dedicated to contain special and "eternal" accounts, such as payables and receivables.
- **Daybook.** I have a "daybook" section at the bottom that contains all cash expenses, in chronological order.
- **Expense accounts.** All my expenses accounts (categories) are defined in their own section.
- **Employers.** For each employer I've defined a section where I put the entries for their direct deposits, and track vacations, stock vesting and other job-related transactions.
- **Taxes.** I have a section for taxes, organized by taxation year.

You can organize it any way you like, because Beancount doesn't care about the ordering of declarations.

Closing Accounts

If a real-world account has closed, or is never going to have any more transactions posted to it, you can declare it "closed" at a particular date by using a Close directive:

```
; Moving to another bank.
2013-06-13 close Assets:US:BofA:Savings
```

This tells Beancount not to show the account in reports that don't include any date where it was active. It also avoids errors by triggering an error if you do try to post to it at a later date.

De-duping

One problem that will occur frequently is that once you have some sort of code or process set up to automatically extract postings from downloaded files, you will end up importing postings which provide two separate sides of the same transaction. An example is the payment of a credit card balance via a transfer from a checking account. If you download the transactions for your checking account, you will extract something like this:

```
2014-06-08 * "ONLINE PAYMENT - THANK YOU"
  Assets:CA:BofA:Checking           -923.24 USD
```

The credit card download will yield you this:

```
2014-06-10 * "AMEX EPAYMENT    ACH PMT"
  Liabilities:US:Amex:Platinum       923.24 USD
```

Many times, transactions from these accounts need to be booked to an expense account, but in this case, these are two separate legs of the same transaction: a transfer. When you import one of these, you normally look for the other side and merge them together:

```
;2014-06-08 * "ONLINE PAYMENT - THANK YOU"
2014-06-10 * "AMEX EPAYMENT    ACH PMT"
  Liabilities:US:Amex:Platinum       923.24 USD
  Assets:CA:BofA:Checking           -923.24 USD
```

I often leave one of the description lines in comments—just my choice, Beancount ignores it. Also note that I had to choose one of the two dates. I just choose the one I prefer, as long as it does not break any balance assertion.

In the case that you would forget to merge those two imported transactions, worry not! That's what balance assertions are for. Regularly place a balance assertion in either of these accounts, e.g., every time you import, and you will get a nice error if you end up entering the transaction twice. This is pretty common and after a while it becomes second nature to interpret that compiler error and fix it in seconds.

Finally, when I know I import just one side of these, I select the other account manually and I mark the posting I know will be imported later with a flag, which tells me I haven't de-duped this transaction yet:

```
2014-06-10 * "AMEX EPAYMENT    ACH PMT"
  Liabilities:US:Amex:Platinum       923.24 USD
  ! Assets:CA:BofA:Checking
```

Later on, when I import the checking account's transactions and go fishing for the other side of this payment, I will find this and get a good feeling that the world is operating as it should.

(If you're interested in more of a discussion around de-duplicating and merging transactions, see this feature proposal. Also, you might be interested in the "effective_date" plugin external contribution, which splits transactions in two.)

Which Side?

So if you organize your account in sections the way I suggest above, which section of the file should you leave such "merged" transactions in, that is, transactions that involve two separate accounts? Well, it's your call. For example, in the case of a transfer between two accounts organized such that they have their own dedicated sections, it would be nice to be able to leave both transactions there so that when you edit your input file you see them in either section, but unfortunately, the transaction must occur in only one place in your document. You have to choose one.

Personally I'm a little careless about being consistent which of the section I choose to leave the transaction in; sometimes I choose one section of my input file, or that of the other account, for the same pair of accounts. It hasn't been a problem, as I use Emacs and i-search liberally which makes it easy to dig around my gigantic input file. If you want to keep your input more tidy and organized, you could come up with a rule for yourself, e.g. "credit card payments are always left in the paying account's section, not in the credit card account's section", or perhaps you could leave the transaction in both sections and comment one out[2].

Padding

If you're just starting out—and you probably are if you're reading this—you will have no historical data. This means that the balances of your Assets and Liabilities accounts in Beancount will all be zero. But the first thing you should want to do after defining some accounts is establish a balance sheet and bring those amounts to their actual current value.

Let's take your checking account as an example, say you opened it a while back. You don't remember exactly when, so let's use an approximate date:

```
2000-05-28 open Assets:CA:BofA:Checking  USD
```

The next thing you do is look up your current balance and put a balance assertion for the corresponding amount:

```
2014-07-01 balance Assets:CA:BofA:Checking    1256.35 USD
```

[2] Some people have suggested that Beancount automatically detect duplicated transactions based on a heuristic and automatically ignore (remove) one of the two, but this has not been tried out yet. In particular, this would lend itself well to organizing transactions not just per section, but in separate files, i.e., all files would contain all the transactions for the accounts they represent. If you're interested in adding this feature, you could easily implement this as a plugin, without disrupting the rest of the system.

Running Beancount on just this will correctly produce an error because Beancount assumes an implicit balance assertion of "empty" at the time you open an account. You will have to "pad" your account to today's balance by inserting a *balance adjustment* at some point in time between the opening and the balance, against some equity account, which is an arbitrary place to book "where you received the initial balance from." For this purpose, this is usually the "`Equity:Opening-Balances`" account. So let's include this padding transaction and recap what we have so far:

```
2000-05-28 open Assets:CA:BofA:Checking  USD

2000-05-28 * "Initialize account"
  Equity:Opening-Balances                    -1256.35 USD
  Assets:CA:BofA:Checking                     1256.35 USD

2014-07-01 balance Assets:CA:BofA:Checking    1256.35 USD
```

From here onwards, you would start adding entries reflecting everything that happened after 7/1. However, what if you wanted to go *back* in time? It is perfectly reasonable that once you've got your chart-of-accounts set up you might want to fill in the missing history until at least the beginning of this year.

Let's assume you had a single transaction in June 2014, and let's add it:

```
2000-05-28 open Assets:CA:BofA:Checking  USD

2000-05-28 * "Initialize account"
  Equity:Opening-Balances                    -1256.35 USD
  Assets:CA:BofA:Checking                     1256.35 USD

2014-06-28 * "Paid credit card bill"
  Assets:CA:BofA:Checking                     -700.00 USD
  Liabilities:US:Amex:Platinum                 700.00 USD

2014-07-01 balance Assets:CA:BofA:Checking    1256.35 USD
```

Now the balance assertion fails! You would need to adjust the initialization entry to fix this:

```
2000-05-28 open Assets:CA:BofA:Checking  USD

2000-05-28 * "Initialize account"
  Equity:Opening-Balances                    -1956.35 USD
  Assets:CA:BofA:Checking                     1956.35 USD

2014-06-28 * "Paid credit card bill"
  Assets:CA:BofA:Checking                     -700.00 USD
  Liabilities:US:Amex:Platinum                 700.00 USD

2014-07-01 balance Assets:CA:BofA:Checking    1256.35 USD
```

Now this works. So basically, every single time you insert an entry in the past, you would have to adjust the balance. Isn't this annoying? Well, yes.

Fortunately, we can provide some help: you can use a Pad directive to replace and automatically synthesize the balance adjustment to match the next balance check, like this:

```
2000-05-28 open Assets:CA:BofA:Checking  USD

2000-05-28 pad Assets:CA:BofA:Checking Equity:Opening-Balances

2014-06-28 * "Paid credit card bill"
  Assets:CA:BofA:Checking                    -700.00 USD
  Liabilities:US:Amex:Platinum                700.00 USD

2014-07-01 balance Assets:CA:BofA:Checking    1256.35 USD
```

Note that this is only needed for balance sheet accounts (Assets and Liabilities) because we don't care about the initial balances of the Income and Expenses accounts, we only care about their transitional value (the changes they post during a period). For example, it makes no sense to bring up the Expenses:Restaurant account to the sum total value of all the costs of the meals you consumed since you were born.

So you will probably want to get started with Open & Pad directives for each Assets and Liabilities accounts.

What's Next?

At this point you will probably move onwards to the Cookbook, or read the User's Manual if you haven't already done that.