leas manual

Zach Flynn

This is the manual for leas, a personal account manager.

Table of Contents

1 Introduction

leas is an interactive, command-line program for managing personal finances. Its goal is to be a simple but easily-extensible tool for helping you keep track of your spending and an aid in making future plans. **leas** was conceived from the need to track my own personal finances leading up to my wedding and finding that most free software personal finance packages were large, complicated GUI programs that were difficult to hack to make work the way I wanted them to. **leas** is a command-line program, but it is possible to write a GUI on top of the program. It is possible because **leas** can be extended arbitrarily using the Guile Scheme programming language. If you make a neat extension like this, let me know and I will reference it in this manual.

No knowledge of Scheme or programming in general is required to use **leas**. I have included almost all the functions I use personally in the **leas** distribution itself. In other words, I use **leas** without any programming in Scheme on a day-to-day basis. The only commands I have written that are not included in the main **leas** distribution are some commands to fetch the price of various stocks I own and update their value automatically, but I will show how to write such a command later in this document as an example.

The other extreme in the accounting software world were programs that required the user to edit text files to add transactions (the best example being **ledger**). I found this to be difficult to use as a user and wanted something more interactive. Of course, you may have your own preference. Like **ledger**, **leas** stores data in human-readable text files. **leas**'s save files are particularly easy to analyze with statistical software, like **R** or **Libreoffice**. The **leas** save file is simply a **tar** archive of comma-separated data files with data about your various transactions. You can open these files in **R**, for example, and analyze your past spending that way.

All commands that ship with **leas** are small Guile Scheme commands which call a few functions exported by **leas** from C and a built-in function `leas/call` which lets the user enter the function arguments interactively (it fills the role of `interactive` in Emacs, if you are familiar with it). These functions are documented in the reference manual to help you construct your own commands. The **leas** prompt, in fact, is a Scheme interpreter. Any Scheme expression can be written there.

This document has two main sections. In the first, I describe the basic workflow for using **leas**. This section is more in-depth than the manpage and gives some examples of how to add accounts and make transactions. It is a tutorial that will help get you up-and-running using **leas**. In the second section, I describe all the Scheme commands exported by **leas** which is more useful if you are trying to write your own commands.

2 Tutorial

2.1 Adding a cash asset account

A **leas** file is made up of several "accounts". To start using **leas**, you will want to add some asset accounts. Assets can be wherever you store your money. Asset accounts can be Checking accounts, Savings accounts, Stocks, Bonds, Cash, and anything else you own. To see how this works in **leas**, let's look at an example of adding a checking account to a fresh **leas** file.

```
:> aa
Account: Checking
0: Expense
1: Income
2: Asset
3: Liability
Type: 2
Opening Balance: 10000
(Checking) :>
```

This opens up an asset account called "Checking" with 10000 units of currency in it as an opening balance. We can use asset accounts to pay for our expenses, to pay back our loans, and as a place to store the income we earn from working. I discuss how to do each of those things in the following sections.

2.2 Paying for expenses

Now that we have an asset account, we can use it to pay for our expenses.

First, we add an expense account, say an account for our Rent.

```
(Checking) :> aa
Account: Rent
0: Expense
1: Income
2: Asset
3: Liability
Type: 0
Opening Balance: 0
(Rent) :>
```

Now, to pay the Rent. The main command you use in **leas** after getting your accounts set up is the "transfer" command, `t`. `t` transfers money from one account to another account. Making a payment is transferring money from your asset accounts to your expense accounts.

```
(Rent) :> t
 0: Checking
 1: Rent
To Account: 1
 0: Checking
 1: Rent
```

```
From Account: 0
Amount: 2000
Description: The rent
Day:
Year [2019]:
Month [4]:
Day [30]:
(Checking) :>
```

By default, the prompt tells you what the *current account* is and, when you add an account, the current account is set to the account you just created. **leas** has several commands that act on the current account. You can change the current account with the `sa` ("switch account") command,

```
(Checking) :> sa
 0: Checking
 1: Rent
Account: 1
(Rent) :>
```

Another command you commonly use in **leas** is the "list transactions" command, `lt`. `lt` lists the transactions in the current account,

```
(Rent) :> lt
2019-04-30 The rent          2000.00
```

To see how much money is in your accounts, type `la` (for "list account").

```
(Rent) :> la
Checking          8000.00    8000.00
Rent              2000.00    2000.00
```

The output has two columns. The first gives the balance in your account on the *current day*, and the second gives the balance in your account in the future.

To see the current day, use the command `cd`. To set the current day, use `sd`.

```
(Rent) :> sd
Current Day:
Year [2019]: 2018
Month [4]: 12
Day [30]: 31
(Rent) :> cd
2018-12-31
```

Now, if we type `la`, we can see what the account balance would have looked like on 2018-12-31 and what it would be in the future.

```
(Rent) :> la
Checking         10000.00    8000.00
Rent                 0.00    2000.00
```

To change the day back to the current day, type `sd` and take all the default options,

```
(Rent) :> sd
Current Day:
Year [2019]:
```

```
Month [4]:
Day [30]:
(Rent) :>
```

To list only expense accounts (useful for getting an understanding of where you are spending your money),

```
(Rent) :> lae
Rent          2000.00     2000.00
```

2.3 Earning income

It is a good idea to earn income to pay for your expenses. Highly recommended. Like with paying expenses, to add income start by adding an income account. Let's start with adding a salary account,

```
(Rent) :> aa
Account: Salary
0: Expense
1: Income
2: Asset
3: Liability
Type: 1
Opening Balance: 0
(Salary) :>
```

To receive a salary, do the opposite of paying for expenses: transfer money from the salary account to an asset account.

```
(Salary) :> t
 0: Checking
 1: Rent
 2: Salary
To Account: 0
 0: Checking
 1: Rent
 2: Salary
From Account: 2
Amount: 6000
Description: Salary
Day:
Year [2019]:
Month [4]:
Day [30]:
(Salary) :> lt
2019-04-30 Salary        -6000.00
(Salary) :> la
Checking       14000.00    14000.00
Rent            2000.00     2000.00
Salary         -6000.00    -6000.00
```

2.4 Managing debt

Adding debt follows the same pattern as paying expenses and receiving income.

```
(Salary) :> aa
Account: Loan
0: Expense
1: Income
2: Asset
3: Liability
Type: 3
Opening Balance: 0
(Loan) :> t
 0: Checking
 1: Rent
 2: Salary
 3: Loan
To Account: 0
 0: Checking
 1: Rent
 2: Salary
 3: Loan
From Account: 3
Amount: 10000
Description: Personal Loan
Day:
Year [2019]:
Month [4]:
Day [30]:
(Loan) :> la
Checking          24000.00    24000.00
Rent               2000.00     2000.00
Salary            -6000.00    -6000.00
Loan             -10000.00   -10000.00
(Loan) :> lt
2019-04-30 Personal Loan        -10000.00
```

Usually, people do not give you interest-free loans. So you will also need an *expense* account for paying interest.

```
(Loan) :> aa
Account: Interest
0: Expense
1: Income
2: Asset
3: Liability
Type: 0
Opening Balance: 0
(Interest) :>
```

To pay back loans, use the command `pl`. This command allows you to split your payment on the loan between interest and principal.

```
(Interest) :> pl
 0: Checking
 1: Rent
 2: Salary
 3: Loan
 4: Interest
Loan Account: 3
 0: Checking
 1: Rent
 2: Salary
 3: Loan
 4: Interest
Interest Account: 4
 0: Checking
 1: Rent
 2: Salary
 3: Loan
 4: Interest
Pay from Account: 0
Principal: 70
Interest: 30
Description: Loan Payment
Day:
Year [2019]:
Month [4]:
Day [30]:
(Checking) :> la
Checking          23900.00    23900.00
Rent               2000.00     2000.00
Salary            -6000.00    -6000.00
Loan              -9930.00    -9930.00
Interest             30.00       30.00
```

It is useful to see broadly how much we are spending, how much we are making, and how in debt we are. To do so, we can use the command `bt`.

```
(Checking) :> bt
Expense            2030.00     2030.00
Income            -6000.00    -6000.00
Asset             23900.00    23900.00
Liability         -9930.00    -9930.00
Worth             13970.00    13970.00
Balances          10000.00
```

Note that income is measured as a *negative* number as are *liabilities*. *Worth* is *Assets - Liabilities*. Balances gives the total of the opening balances.

2.5 Non-cash assets

You may own some non-cash assets, like stocks, mutual funds, or bonds. The value of these
assets in terms of currency changes over time. **leas** provides a command for updating the
currency value of these assets.

First, let's add our mutual fund,

```
(Checking) :> aa
Account: Mutual Fund
0: Expense
1: Income
2: Asset
3: Liability
Type: 2
Opening Balance: 10000
(Mutual Fund) :>
```

Then, let's add an income account for our fund.

```
(Mutual Fund) :> aa
Account: Mutual Fund Income
0: Expense
1: Income
2: Asset
3: Liability
Type: 1
Opening Balance: 0
(Mutual Fund Income) :>
```

Now, say we own 500 shares of the mutual fund each worth 20 currency units. Say the
value of a share in the mutual fund increased to 21 currency units. We can then use the
`csp` ("change share price") command to change the value of the shares in our mutual fund,

```
(Mutual Fund Income) :> sa
 0: Checking
 1: Rent
 2: Salary
 3: Loan
 4: Interest
 5: Mutual Fund
 6: Mutual Fund Income
Account: 5
(Mutual Fund) :> csp
 0: Checking
 1: Rent
 2: Salary
 3: Loan
 4: Interest
 5: Mutual Fund
 6: Mutual Fund Income
From Account: 6
```

```
Stock Price: 21
Number of Shares: 500
Day:
Year [2019]:
Month [4]:
Day [30]:
(Mutual Fund Income) :> lt
2019-04-30 Stock Price Change          -500.00
(Mutual Fund Income) :> la
Checking                    23900.00    23900.00
Rent                         2000.00     2000.00
Salary                      -6000.00    -6000.00
Loan                        -9930.00    -9930.00
Interest                       30.00       30.00
Mutual Fund                 10500.00    10500.00
Mutual Fund Income           -500.00     -500.00
(Mutual Fund Income) :>
```

2.6 Saving, loading, and quitting

To save your **leas** accounts, use the `w` command.

```
(Mutual Fund Income) :> w
File: example.leas
(Mutual Fund Income) :>
```

To do so non-interactively, type `leas/write "example.leas"`.

To quit **leas**, use the `q` command,

```
(Mutual Fund Income) :> q
Save file? (y/n) n
```

To load the file you just saved, you can use the interactive command `r`,

```
$ leas
:> r
File: example.leas
(Checking) :> la
Checking                    23900.00    23900.00
Rent                         2000.00     2000.00
Salary                      -6000.00    -6000.00
Loan                        -9930.00    -9930.00
Interest                       30.00       30.00
Mutual Fund                 10500.00    10500.00
Mutual Fund Income           -500.00     -500.00
(Checking) :>
```

To load the file non-interactively, you can use the command `leas/read`
`"example.leas"`. This command is particularly useful to include in **leas**'s init file
`~/.leasrc.scm`. Usually, this file will include the line,

```
(leas/read "leasfile.leas")
```

2.7 Scheme code

The prompt is a Scheme interpreter and can execute arbitrary code. The only difference is that the outer expression should not be enclosed in parenthesis. For example,

```
(Checking) :> display (string-append (number->string (+ 1 2)) "\n")
3
(Checking) :>
```

leas provides a useful function `p` for displaying expressions and then adding a newline like the above,

```
(Checking) :> p (+ 1 2)
3
(Checking) :>
```

In interactive functions, the prompt for the various arguments allows Scheme expressions as well. For example, you can enter the value of a transaction as `(- 15.29 13.99)`.

2.8 Common customizations

You can add customizations that are loaded automatically in `~/.leasrc.scm`. This section gives examples for a few common customizations.

2.8.1 Set default file

You will probably want to load the same file almost everytime you load **leas**. To do so, add the following to `~/.leasrc.scm`:

```
(leas/read "/path/to/file/myfinances.leas")
```

2.8.2 Customize prompt

The **leas** prompt is generated by calling the Scheme function `(leas/prompt)`. If you change this function, you can customize the prompt. For example, try adding the following to see the current date,

```
(define leas/prompt
  (lambda ()
    (if (= (leas/get-number-of-accounts) 0)
        ":> "
        (let* ((day (leas/get-current-day))
               (mday (list-ref day 0))
               (month (list-ref day 1))
               (year (list-ref day 2)))
          (string-append
           "("
           (leas/get-current-account)
           " "
           (number->string year)
           "-"
           (number->string month)
           "-"
           (number->string mday)
           ") :> ")))))
```

3 Btar file format

The **leas** save file is a **tar** archive containing the following files in a directory named after the save file's filename (e.g. a save file named *book.leas* would, when un-tarred, be a directory called *book*):

- accounts - a CSV file containing the metadata for each account, one line for each account. There is no header line giving column names. The fields are (in this order):
  - Account Type - one of expense, income, asset, liability.
  - Account Name - the name of the account
  - Opening Balance - the opening balance of the account.
- *account_name*.csv - for each account, there is a separate CSV file (the format of the filename for this CSV file implies that account names in leas need to be valid filenames). There is no header line giving column names. Each row in the file is a transaction. The fields are (in this order):
  - Account Name - will be the same for all transactions in the file, just the account name.
  - Amount - the amount of money added to or subtracted from the account.
  - Day - the day of the transaction (YYYY-MM-DD)
  - Description - a description of the transaction

4 Programming leas

This chapter will be more of a reference manual for the various Scheme functions available in **leas**.