LINUX VOICE
CODING TUTORIAL

GOOGLE SCRIPT YOUR GROCERY BUDGET

GRAHAM MORRISON

Forget boring accounting software. Code your own cloud-enabled budgeting script instead.

We're the first to admit we feel uncomfortable with the amount of data that Google is gathering on every aspect of our lives. Many of us on the team are making a concerted effort to move away from some of their services – especially when it comes to location tracking, context searches and personal information (facial recognition, social interaction and profile analysis). But we're also not the types to throw babies out with their bath water. Google has done, and still does, many good things for Free Software, and many of its services are genuinely useful.

And one of the most useful is its scripting engine, known colloquially as Google Apps Script, and there are two reasons why we think it's worth the effort of using. The first is that the scripts themselves are easy to write. The language is very similar to JavaScript, and while we accept that JavaScript is just as difficult as any other language if you want to become a master, for casual use it can be straightforward, quick and easy. It's widespread enough that many people will have come across it while hacking their own websites, and Google has also done a good job at documenting the various APIs that allow its scripting engine to access and process your data.

The second redemptive excuse we're offering is that you can schedule scripts to run automatically at any time, and unlike your network attached storage box, your Raspberry Pi or low-end-Linux machine, Google's servers rarely suffer outages and come for free.

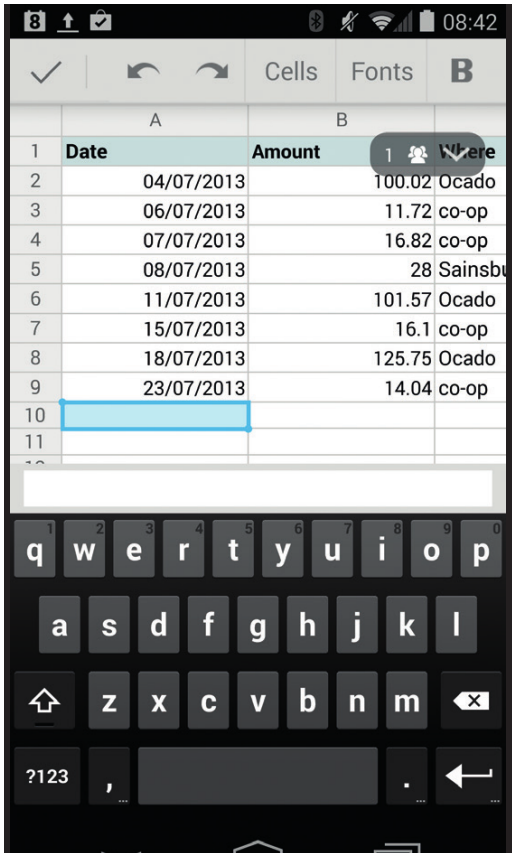
1 PROJECT ORIENTATION

To help illustrate what Google's Apps Script is capable of, and how you might best be able to use it, we're going to create a budgeting system for managing grocery expenditure. The idea is simple; set yourself a budget for each month, and whenever you go to the shops and buy something from your budget, you log the amount. The remaining budget is calculated and is sent via a weekly report telling you how much you've spent and how much you've got left to spend.

Thanks to Google, lots of the complexity is handled for us. To log spending we'll use a Google spreadsheet. These work extremely well from most smartphones, and from any Android device in particular, so it's no hassle adding totals as you go along. As it's a Google spreadsheet, you can also share it with other people, who will then be able to add and manage spending themselves. This is a great solution for a typical household.

We'll construct the spreadsheet in such a way that the data we place into it is easily accessible (through Google's APIl) to the script we'll write to tie everything together. We'll then write the script to take the important parts of this data, such as the total, the budget, when the cash was spent and how much you'd like to spend, and then write some simple logic around the calculation before outputting a verdict on your spending. The whole script can then be scheduled to run and email one or more people with the results at a specific time.

We feel bad writing this, but you'll need a Google account first. From there, you'll need to click on your

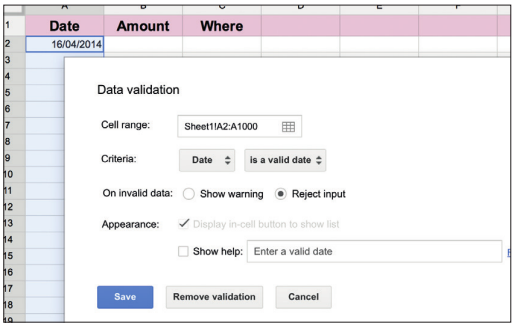


You can keep on top of your finances from your phone, your tablet or your laptop with equal ease.

Google Drive button or go to **http://drive.google.com**. This is Google's shared storage service that is now the central repository for Google Docs too. We imagine most readers will have a Google account already, so this shouldn't be too much of an issue, but we promise to revisit the subject if enough people would like to see a solution using an open source service, such as OwnCloud, rather than a Google service.

2 CREATING THE SPREADSHEET

From the Google Drive page, click on the large Create button at the top-left and select 'Spreadsheet'. A few moments later, a blank untitled spreadsheet will appear in your browser window. We've called ours simply **lv_groceries** by clicking on the unnamed value at the top. Our solution has two sheets – one for logging day-to-day expenditure and the other for making the various calculations and for holding our budget values. The first sheet is very easy to create, and the best place to start is by giving the first three columns a title each – 'Date', 'Amount' and 'Where'. You might also want to highlight these cells by changing the justification, using a bold font or perhaps a different background colour. This is the page you're going to use to enter your expenditure, sometimes from your laptop and sometimes from your phone or tablet, so a clear layout will help you to be accurate.



Use the 'show help' field to give your friends a little clue about what you want entered.

Arrange your data

As you can tell from the three column names, the first column is going to hold the date of the purchase. Google spreadsheets have a data validation feature that does two things for you.

1. It will only allow a valid date to be entered. This stops any rogue data creeping into our scripts, obviating the need to write code to handle the subsequent errors.
2. The convenience of date formatting as you get a pop-up calendar from which you can choose your date. This is much easier to use than typing in a date manually, and avoids any confusion over how a date should be formatted.

To enable data validation, Shift+select every cell in the first column beneath the title, and either right-click and select Data Validation or select the Validation option from the Data menu. A window will appear showing the cell range you've selected so you can make sure the selection is correct (the first column is 'A', and beneath this you need to select your criteria for validation). Click on the first pop-up menu button and select Date from the short number of input formats that can be validated. Secondly, in the second pop-up menu, make sure that 'Is A Valid Date' is the logical operation automatically set for you. On the following line, you can now choose to either show a warning if a value isn't a date, or reject the input. We went for the first option, as the second can be a little restrictive, especially if you just want to delete a date completely, as this isn't accepted as a standard date.

We don't make any formatting constraints for the other two columns, although theoretically we could for the second column, which is going to hold the value of each expenditure. We don't use the third column, 'Where', but we find this information is useful for monitoring where you spend the most money and for problem solving if you need to cross-check a purchase against a bank statement. This is only the first sheet, however, and we're going to create a lot more functionality in the second sheet, which you can create by clicking on the '+' symbol at the bottom of the page.

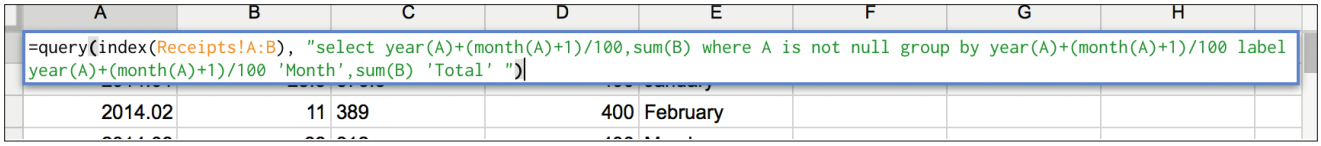
LV PRO TIP

We used the new version of Google's spreadsheet for this tutorial – released early March, but it should also work on the older version.

3 DATA PROCESSING

Before moving on to creating the second sheet, we need to give them both names that are going to make moving between them easier. We called our first sheet Receipts, as the values were mostly read off grocery receipts after buying something, and Budget for the second sheet, which is what we're going to explain now. You rename sheets by right-clicking on the tab at the bottom of the current spreadsheet.

We're going to create four columns in the second sheet, all of which are going to be for the convenience of our script rather than for direct use – although they'll also provide a good overview of your annual and monthly spending. To make sure everything works, we'd highly recommend creating some dummy data back on the first sheet so that when we add some calculations (and eventually the script), they'll have some real numbers to work on and you can judge on the feedback whether everything is working correctly. After you've done that, switch back to the second sheet. The first two columns will hold a reformatted month string and the total expenditure during that month, and we can create both using a formula. Double-click on the A1 cell (the first one on the sheet), and enter the following:



Google doesn't provide much error feedback, so you need to make sure all brackets and quotation marks are correct when entering a query.

```
=query(index(Receipts!A:B), "select year(A)+(month(A)+1)/100,sum(B) where A is not null group by year(A)+(month(A)+1)/100 label year(A)+(month(A)+1)/100 'Month',sum(B) 'Total' ")
```

If you're familiar with spreadsheets, and Google's in particular, you'll know that you can access data contained within its sheets using a 'select' statement, just as you would a database. And that's exactly what we're doing here. The reason why we're doing it this way is because it gives us greater flexibility in how we handle the return values. Here's what it does, broken down into chunks of functionality:

```
=query(index(Receipts!A:B),
```

This basically grabs an array of values from both the A and B columns of the 'Receipts' sheet on your spreadsheet. 'Receipts' need to be the same the name of your first sheet. The data from the two columns, A and B, is then passed on to the 'select' statement, for first part of which we'll tackle next:

```
select year(A)+(month(A)+1)/100,sum(B) where A is not null
```

LV PRO TIP

You can show all formulae running on a spreadsheet by selecting the option from the 'View' menu, making problem solving a little easier.

Date formatting

For our eventual script to work without any extra effort, we need the month to be formatted in a specific way: 2014.05, for May 2014, for example. Not only does this help with sorting, but it's easier to process as it appears as a floating point number.

The above command creates that formatting by taking the year and month from the first column (A), and pushing the numeric value for the month through a division by 100 to push the two digits to the right of the decimal place. We're also selecting the corresponding value in the adjacent column.

```
group by year(A)+(month(A)+1)/100 label year(A)+(month(A)+1)/100 'Month',sum(B) 'Total' ")
```

This is the remainder of the query. The **group by** makes sure that the same months are grouped together and with a label that's the same as the calculation – this will be the value itself. And to the right of this we place the total **sum(B)** for all the expenditure from that month, along with two titles for the two columns that are created. If you've created dummy data on the first sheet, you should see an entry in the first column for each month of expenditure, along with a total for that month in column B.

We now need to add three extra columns. In column 6, or 'E' on the sheet, we're going to type the word for each month starting with January in E2 and ending with December in E13. This is a cheat, so we can email the word for the month from the script. In the column to the left of this, 'D', enter the budget you want your spenders to adhere to for each month. We've done this for each month separately in case you wanted more budget for Christmas or birthdays. Finally, to the left of the budget column, we're going to enter a calculation to work out how much money you've got to spend in each month. This is as simple as subtracting the contents of the cell to its left (the total for the month), from the contents of the cell to its right (the overall budget for that month). To do that, double-click in the second cell in column C and type **=SUM(D2-B2)**. You can easily copy and paste the formula so that it changes to reflect the left and right cells of each new position by dragging the blue border surrounding the cell down the column.

4 WRITING THE CODE

The final step is to write the JavaScript-like code to take the data from our spreadsheet and email it to ourselves. To create a script from the spreadsheet, choose 'Script Editor' from the Tools menu. This opens a new editor window containing a simple template function called 'myFunction'. Here's the first bit of code – place all this code between the curly brackets of the function:

```
var sheet = SpreadsheetApp.openById("1dWqQha3E");
var budget = sheet.getSheetByName("Budget");
```

All this code is doing is opening the spreadsheet we opened earlier. The value within the double quotes is the reference to the spreadsheet, and you need to get this from its URL – it's the value that appears where the ******** is in the following line, but this might depend on the version of sheets that you're using. Either way, the unique identifier for your spreadsheet should be fairly obvious within your spreadsheet's URL:

```
https://docs.google.com/spreadsheets/d/****/edit#gid=1426067592
```

The next few lines of code are going to make a few assignments to get the current date and implement an offset. We're assuming your budgeting starts in January, but if it doesn't, change the first **startmonth** value to your start month number. You'll also need to offset the word list on the spreadsheet.

```
var startmonth = 0;
var date = new Date();
var month = date.getMonth();
month = (month - startmonth) + 2;
```

We're now going to grab some data from the spreadsheet, first by using the **getRange** method with the **month** variable to specify the row and '3' for the 'Remaining' column. This value will then be appended

to a string we'll use as the subject line in the email, as well as within the body of the email later:

```
var dataRange = budget.getRange(month,3);
var data = dataRange.getValues();
var remaining = parseFloat(data);
remaining = remaining.toFixed(2);
var subject = "Grocery Budget Remaining: " + remaining;
```

We'll cheekily use the same trick to add the text string for the month, taken from the fifth column in the spreadsheet:

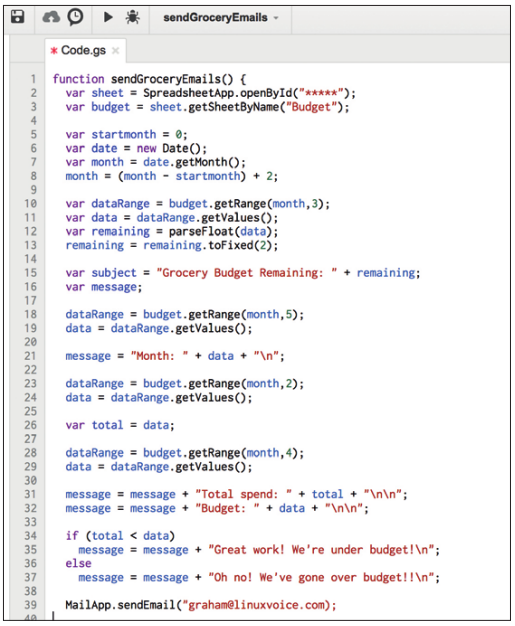
```
dataRange = budget.getRange(month,5);
data = dataRange.getValues();
var message = "Month: " + data + "\n";
```

Add to the body of the message by grabbing the total spend value and putting this in along within the message before adding the total budget for the month:

```
dataRange = budget.getRange(month,2);
data = dataRange.getValues();
var total = data;
dataRange = budget.getRange(month,4);
data = dataRange.getValues();
message = message + "Total spend: " + total + "\n\n";
message = message + "Budget: " + data + "\n\n";
```

Now we've got all the variables together, we can write a quick conditional expression that changes the text of the message depending on whether you're under or over budget, leaving the final step to be the sending of the email itself. This is remarkably simple from Google App Script, as you simply call the **sendEmail** method from MailApp, using an email address with both the subject and message variables to handle everything else. Obviously, you'll want the email address to be your own, entered carefully, because the nightmare of being blacklisted for mail bombing your budgets from Google's servers isn't worth the potential embarrassment:

```
if (total < data)
 message = message + "Great work! We're under budget!\n";
else
 message = message + "Oh no! We've gone over budget!!\n";
MailApp.sendEmail("graham@linuxvoice.com",subject,
message);
```



Google's script editor has syntax highlighting and an effective debugger, which can help if you find any errors.

LV PRO TIP

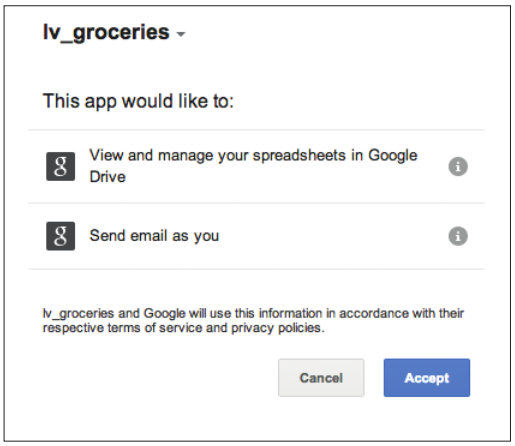
Use the 'Share' button to allow other people to add expenditure, and don't forget to add their emails to the script if you want them to get a notification.

5 RUNNING THE SCRIPT

You're now at the point where you can run the script. To do this, just click on the small black 'Play' button in the script editor toolbar. The first time you run the script, you'll be asked to authorise its access to the spreadsheet and to your email account, which is where the email will appear to originate from. With a bit of luck, a few moments after validation the script will execute and you should see an email like this:

```
Subject: Grocery Budget Remaining: 217.50
Month: April
Total spend: 182.5
Budget: 400
Great work! We're under budget!
```

Congratulations! It works! All that's now left to do is schedule the script to run at a time that makes best sense for you. This is accomplished through Google's trigger system, which can be enabled by going to the script editor, clicking on the 'Resources' menu and selecting 'Current Project's Triggers'. A wide window will include the text 'Click Here To Add One Now', and when you click on that, you can select a 'Time-driven' event to run on a 'Week Timer', 'Every Sunday' at a specific time, or whatever day/time work best for you.



You'll need to give your script permission to access your spreadsheet and to use your email account.

You can even use a trigger to send an email whenever the spreadsheet is opened or changed, giving you the awesome cloud control for your budget, and ultimately, more money to spend on beer.

Graham Morrison left eBay off this budget spreadsheet to hide the amount he spends on vintage synthesizers.