Wins -
Admin Document



WHAT IS THIS DOCUMENT FOR?

This document is to guide current and future HackForLA website team leads on how the Wins Google Apps Script works, how it uses the API token, and who should get access to it.

WHO IS THIS DOCUMENT FOR?

This guide is specifically for administrative purposes and should only be shared with the website team leads, website merge team, and any other individual who has admin-level privileges. Please **do not** share this guide with anyone that does not match the criteria written above.

HOW DOES THE WINS PAGE WORK?

The Wins page is a page on the website that displays the accomplishments of current and former volunteers in Hack For LA. To get featured on this page, a volunteer fills out a Google Form to submit their information. The responses are then saved in a Google Sheet. Currently, the Wins Form and Wins Responses Sheet are both stored in the website drive.

There is a Google Apps Script attached to the Wins Responses Sheet. This script has three main functionalities: first, it will trigger once every month to pull data from the Wins Responses Sheet, and make a Pull Request in the website's GitHub repository to merge any new data. Second, everytime a user submits a new form, it creates an issue in the website's repository to alert project managers about the new submission. Lastly, as a continuation of the second functionality, it updates the review sheet in the Wins Responses Sheet to reflect the changes. The review sheet acts as a "user-friendly" way of reading the data.

On the GitHub repository, a member of the website-merge team reviews and merges the Pull Request, which updates a wins-data.json file. JavaScript code deciphers this json file, which outputs what we see in the Wins page.

This document will mostly cover the Google Apps Script part of the workflow.

Google Apps Script Workflow:

There are two scripts that work in conjunction to successfully integrate with GitHub. The first one is the script mentioned in the overview of the previous page. The second script is a library that utilizes a GitHub token to handle all API requests with GitHub. The first script uses this library with no knowledge of how the API requests work. We separate the two scripts to protect the API key and ensure that the first script (which will generally get more footprint from future volunteer developers, whereas the library will be limited to website leads and the merge team) cannot do anything malicious to the codebase. This idea is inspired from this article.

The Google App Script: winsFormResponse

As mentioned in the overview, this script has three main functions: pulling data from the responses sheet and creating a Pull Request in the website repository, creating a new issue in the repository about a new submission, and updating the review sheet. This script also utilizes the gh-requests library (the second script) to make API requests to GitHub.

Since this is a container-bound script, it is attached to the Wins Responses SpreadSheet. Permission levels assigned to the Wins Responses SpreadSheet propagate to this script as well. So if a user has read/write access to the Wins Responses SpreadSheet, they will also have read/write access to the script.

It should also be noted that triggers for this script are set up using the Elizabeth Honest bot account. When triggers are created in Google App Scripts, only the creator of the trigger can modify and delete them. Therefore, if existing triggers need to be modified, or new triggers need to be added, they should be established using the Elizabeth Honest bot account.

1. **main()**

This function takes the data from the Responses Sheet, creates an array of objects, and encodes it in base64. Then, it gets the SHA of the wins-data.json file from GitHub. Afterwards, it uses the encoded data and retrieved SHA to update wins-data.json on a fork of the website repository. This fork belongs to a bot (an account called Elizabeth Honest Bot, or Beth) that is dedicated to this functionality only. Lastly, a Pull Request is created to merge the branch from Beth's fork into the website's gh-pages branch.

This function triggers once a month, so the website-merge team has a month to review and merge the data. If for some reason the Pull Request is still open after a month, the script will attempt to create a Pull Request and fail, an error will appear in the logs, but the trigger will still complete.

2. **createIssue()**

This function triggers during a new form submission. It creates an issue in the website repository to inform Project Managers that a new form was submitted. This is generally done to ensure that submissions do not contain any inappropriate messages.

This also creates a new card under the “New Issue Approval” column of the Project Board, and links the newly created issue.

3. **updateReviewSheet(formSubmitEvent)**
This function triggers during a new form submission. It updates the Review Sheet of the Wins Form Responses Spreadsheet and provides a user-friendly way of viewing the data.

The API Library: gh-requests

This library acts as the liaison between the first script and GitHub. It retrieves the GitHub token and sets it for GET requests, POST requests, and PUT requests. This library is also a container-bound script, attached to a placeholder document in the Secret folder of the admin drive. This is done to hide the script. This library (and the document it is bound to) should only be shared to website leads and the website-merge team. Anyone else should not get access.

It should also be observed that there are two kinds of functions: normal functions (i.e. main()) and underscored functions (i.e. main_()). Normal functions are usable inside the library as well as any script that calls the library externally. Underscored functions are protected functions that can only be used by methods inside the library. This means that the first script can only use normal functions, and not underscored functions. Generally speaking, underscored functions in this library are the API methods (GET, POST, PUT), and the token retrieval function. The normal functions instead use the underscored functions and return data as needed. This way, the first script does not need to interact with any tokens, nor API methods directly, ultimately securing the repository from any possible malicious attacks.

1. **getSHA(fileName)**
This function retrieves the SHA of a specified file. The SHA is needed for future requests to update that file.

2. **updateWinsFile(fileName, content, sha)**
This function updates a specified file given a specified payload.

3. **createPR()**
This function creates a Pull Request in Beth’s fork.

4. **createIssue()**
This function creates an issue in the website repository.

5. **addIssueToProjectBoardColumn(issueID, columnID)**

This function creates a new project card in a project board column, and links a specified issue in that newly created card.

6. **setToken_()**
This function retrieves the GitHub API token from a Google Doc. The token is encoded in base64.

7. **decode(value)**
This function decodes a base64 input.

8. **getRequest_(url, acceptHeader)**
This initiates a GET request to GitHub based on an API endpoint and an Accept Header.

9. **putRequest_(url, acceptHeader, payload)**
This initiates a PUT request to GitHub based on an API endpoint, an Accept Header, and a payload.

10. **postRequest_(url, acceptHeader, payload)**
This initiates a POST request to GitHub based on an API endpoint, an Accept Header, and a payload.

It should be observed that all API requests to GitHub of the same method have the same structure. The only difference is the Accept Headers, which is described in GitHub's documentation. This is why the Accept Headers are stored as a part of a constant object at the top, ACCEPT_HEADER.

Security:

Elizabeth Honest Bot

This account was created solely for GitHub interaction. The API token used in the gh-requests library is generated from Beth. Website team leads should be the only ones to have access to this account.

Once a day, the script will create a new pull request to merge any new data from the wins submission to the wins-data.json file. Leads will need to review this data and approve the pull request.

Admin Drive

The admin drive currently hosts two important directories: the Secret folder and the Wins folder. The Secret folder hosts the GitHub API token and the document that the gh-requests library is attached to, while the Wins folder hosts the Wins Submission Form and the Responses SpreadSheet.

It should be noted that the Admin Drive should only be shared with the website merge team, and anyone else who would need special admin privileges. The Secret folder should only be shared with the merge team. If developers need to work on either the gh-requests library or the wins script, then they should only receive view permissions on those individual files. They would need to create a similar environment with their own google drive, implement new changes in their drive, demo it to the leads, and get approval before changes are implemented. PMs who need to approve new wins submission should only get access to the Wins folder.

Secret Folder

As mentioned previously, this folder currently hosts the API token and the document that the gh-requests library is attached to. Therefore, it is important to ensure that access to this folder is given only to the merge team, and other admin-level individuals. Developers should not get access to this entire folder.

API Token

The token is currently kept in the gh-key file, where it is encoded in base64. The key is currently set to not expire. (Before, the key was set to expire monthly and the website leads needed to update and refresh this key once a month.) To refresh the key, go to the Elizabeth Honest GitHub account, refresh the token, and encode it in base64 (https://www.base64encode.org/) keeping the default options, and paste the new encoded key in the gh-key file. (For detailed step by step instructions, see How to Update Elizabeth Honest Bot Token.)

How to Update Elizabeth Honest Bot Token

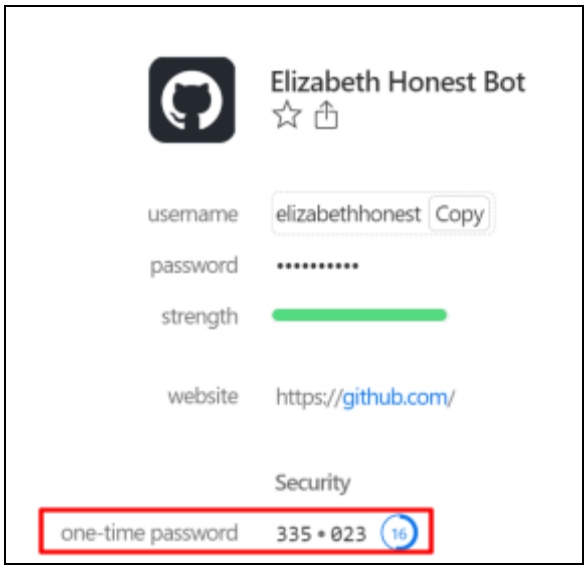
Prerequisite

1. You have a 1Password account with Hack for LA.
2. You have access to the “Hackforla.org website” vault in 1Password.

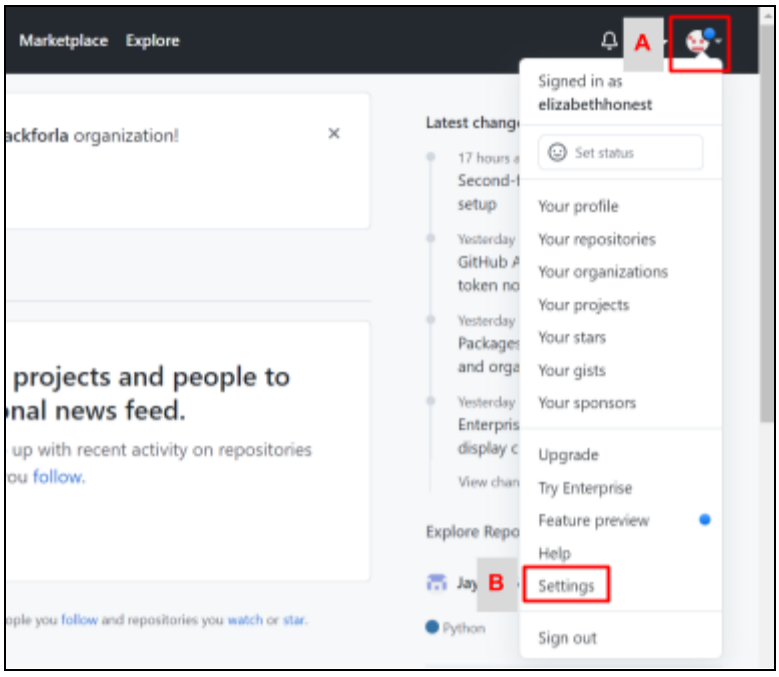
If you don’t have one or both of the prerequisites, ask a Product Manager on the Website Team.

Instructions

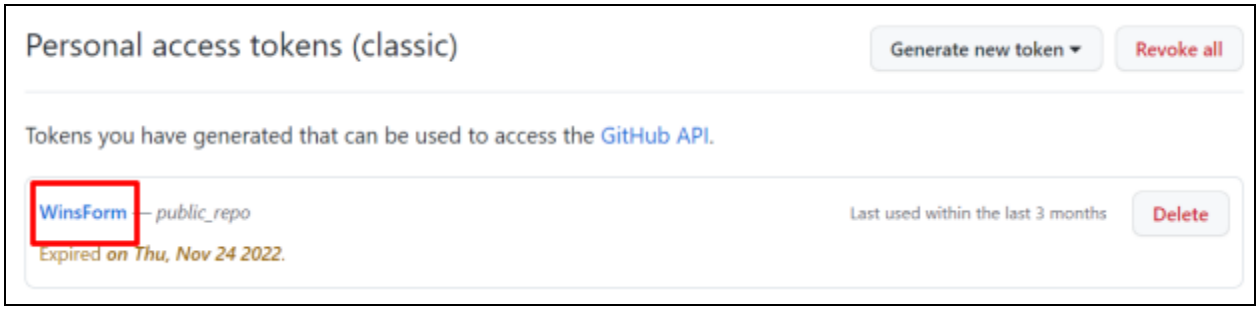
1. In your own Google Account, navigate to the **HfLA website Admin** drive.
2. Select **HfLA website Admin** folder **> Secret** folder **> gh-key** Google Doc.
3. In a new tab, log into 1Password
4. Navigate to the “Hackforla.org website” vault
5. Click on “Elizabeth Honest Bot” to access the GitHub account information for the bot.
6. Login to the GitHub account using the information from step 5.
7. When asked for an “Authentication Code” on GitHub, copy and use the “one-time password” in the account information in 1Password under the Security section.
    a. If you enter the one-time password and it says, “Two-factor authentication failed.”, check the time on your computer or device is accurate using How to Check and Sync Your Device's Time Guide.
    If your time is ahead or behind, you will need to fix it. If there is less than a 1 second difference from the time in Time.is, you should be okay. If the difference is over 1 second, you can still use 1Password’s one-time password but you may have to wait till after a few seconds have passed on the new countdown, or use it before the last few seconds of the countdown are up.



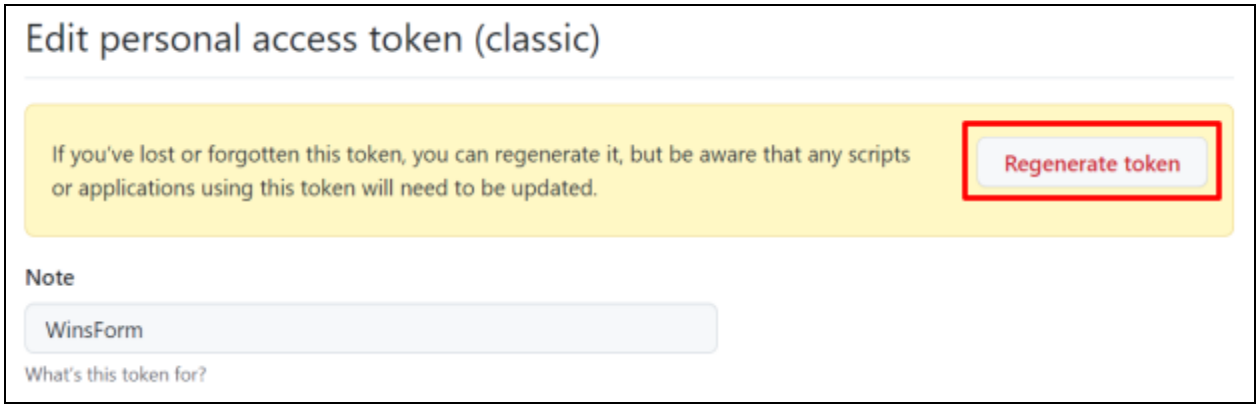
8. In the top right corner,
    a. click on the profile photo and
    b. then click “Settings”.



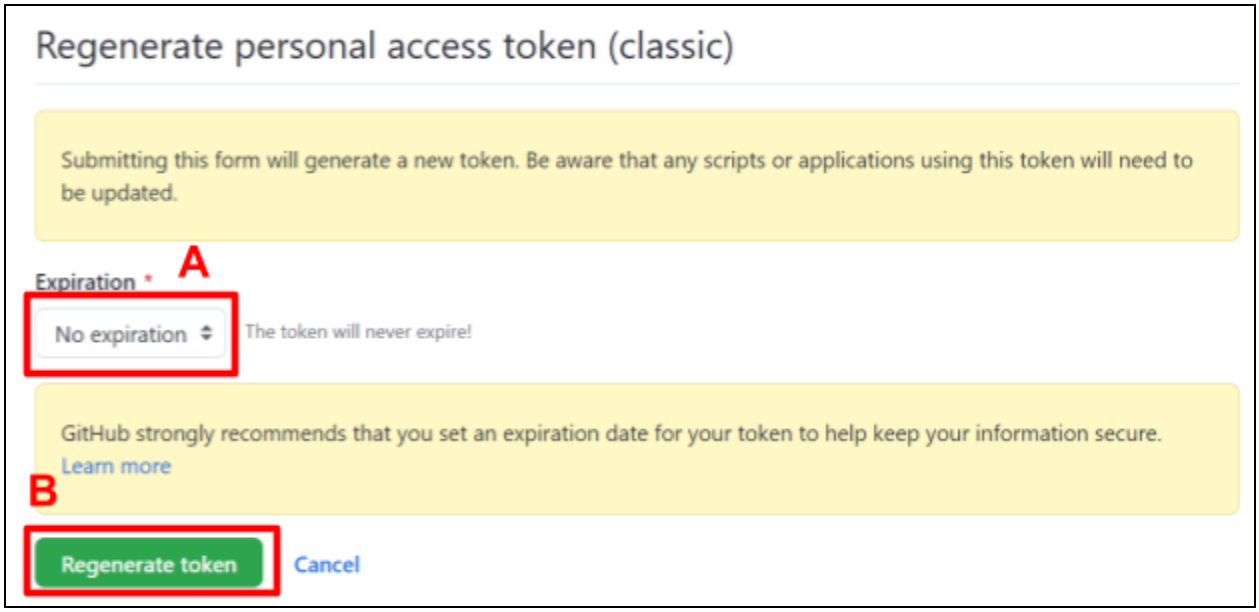
9. In the left menu, scroll down and click on “Developer settings”.
10. In the left menu, click on **Personal access tokens > Tokens (classic)**.
11. Click on “WinsForm”



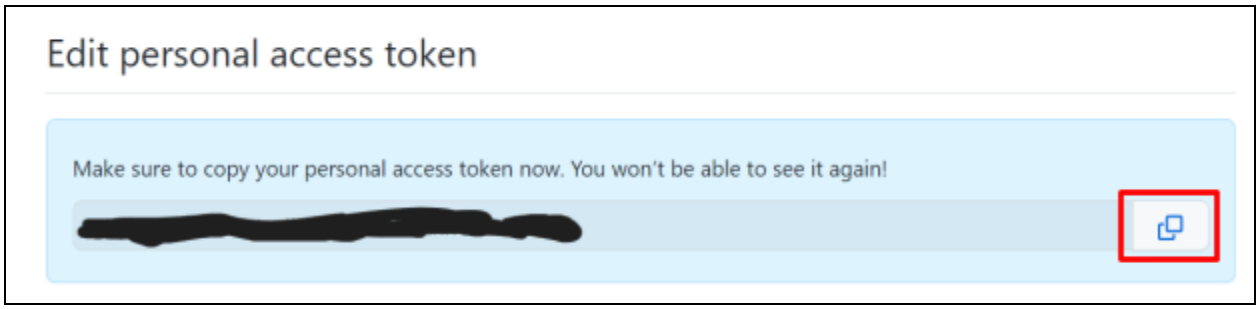
12. Click on “Regenerate token”



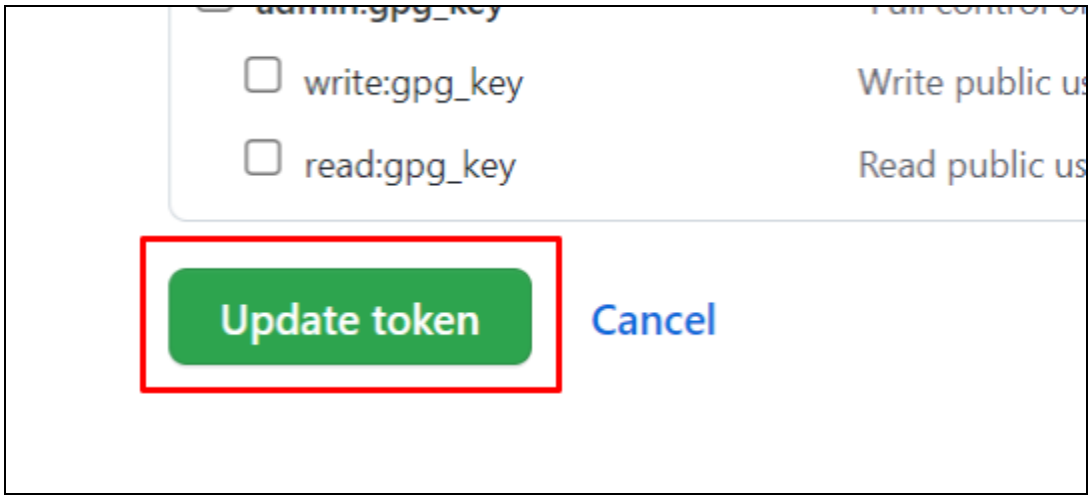
13. Under “Expiration”,
    a. select “No expiration” and
    b. click “Regenerate token” button



14. Click on the copy button, which is the button that looks like 2 rectangles overlapping each other.



15. Scroll down and click “Update token”.



16. Encode it in base64 (https://www.base64encode.org/) keeping the default options
17. In the **gh-key** file, delete the old key and paste the new encoded key.

gh-requests Placeholder Document

This is a placeholder document for the gh-requests library. The document itself does not do anything; it is simply used so the gh-requests library is kept and stored in the admin drive. If the library needs to be shared to a developer, give only read access to the document. Any proposed changes should be demonstrated and approved by the leads.

Wins Folder

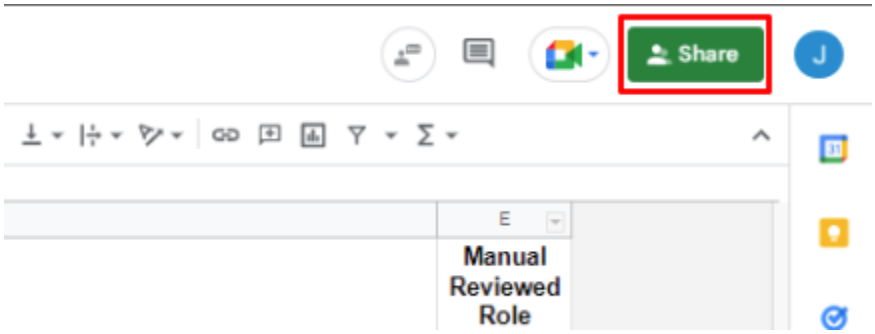
As mentioned previously, this folder currently hosts the Responses SpreadSheet and the Wins Submission Form. PMs will need access to this folder so they can review and approve new wins submission. Ensure that PMs only get access to this folder, and nothing else outside of it.

Responses SpreadSheet

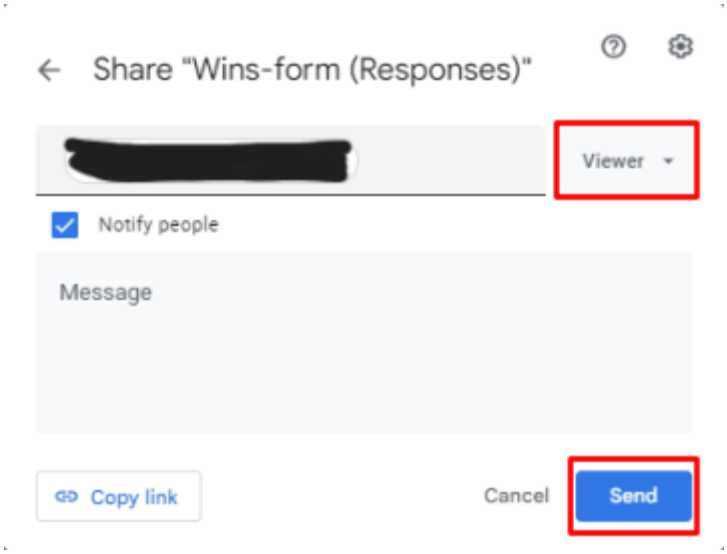
This SpreadSheet is where the responses are stored for each wins submission. It is also the container for the wins google app script. Permission levels given to this document also propagate to the google app script, so ensure that developers only get read access to this. Unfortunately, PMs will get write access to the google app script. However, they generally do not need to touch the script.

How to share files with developers working on an issue

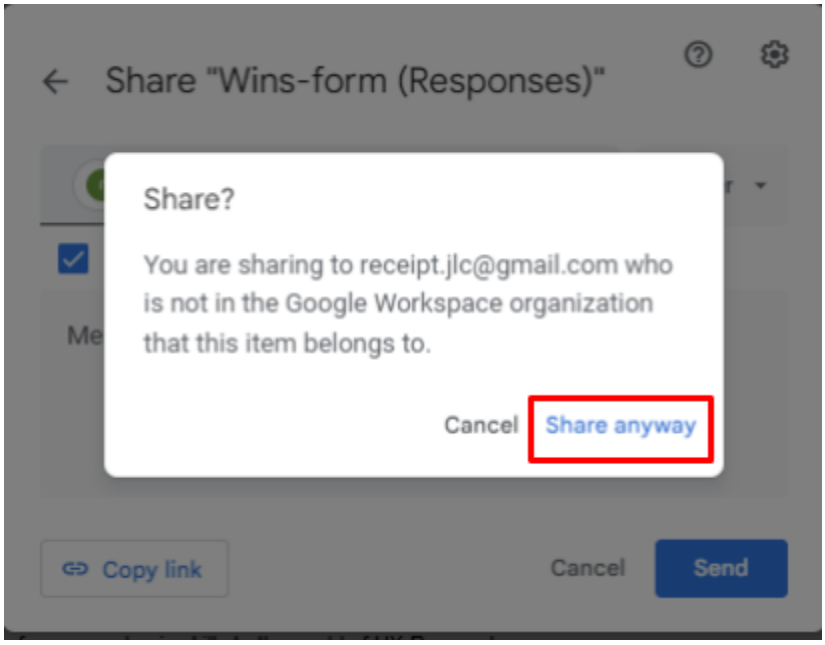
1. Read the Security section above, including all subsections, to help you understand what files to share.
2. **Share only the necessary files** and associated Google Apps Script to work on the issue.
3. Navigate to and open the file you want to share. As an example, I will use “Wins-form (Responses)” file.
4. Click on the “Share” button in the top right.



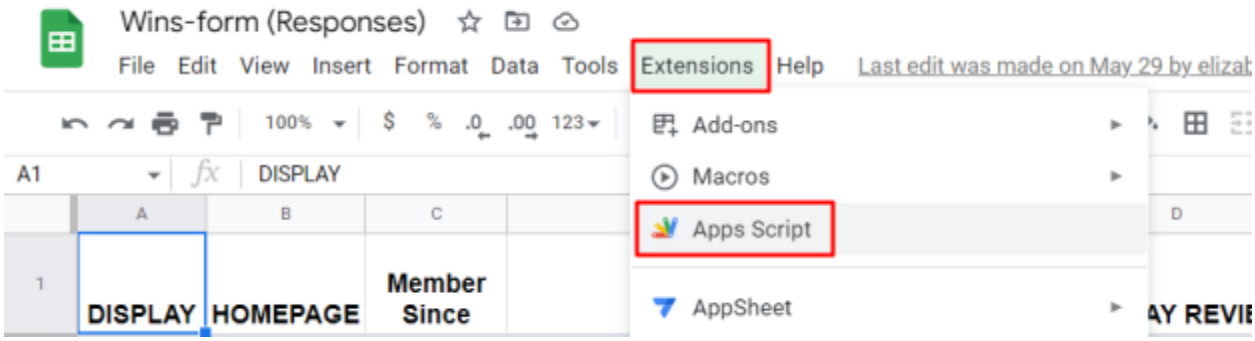
5. Input their email address and then select “Viewer”. Then click the “Send” button.



6. Then click “Share anyway”.



7. To share the Google Apps Script associated with the file, “Extensions” and then click “Apps Script”.



8. Once the Apps Script is open, copy the web address for it and share it with the developer.
9. The developer will need to
    a. Copy and paste the contents of the file **into their own file in their own Google Drive**. This may also include editing the tab names and other aspects of their own file so it matches the original file.
    b. Then, **in their own file**, navigate to the Google Apps Script by clicking “Extensions” and then “Apps Script”, and copy and paste script from the original file **into their own file** and click “Save project”.
        i. The developer will also need to set up their own Personal Access Token to use with the Google Apps Script in your own Google Drive. (Note: Personal Access Tokens are like passwords, so be very careful with them.)

Peer created guides are an important part of Hack for LA’s Culture. Help us to make it better by posting any comments directly next to the content.

This guide was created and contributed to by: Matthew A. Cacho
Additional contributors:
Jessica Cheng