PRISM Scripts Collaboration Instructions

Or: How to use GitHub like a (child support) rock star

# Introduction

The BlueZone Scripts project (for PRISM scripts) aims to increase efficiency and consistency by using technology already present in BlueZone (the state’s mainframe emulation software) to its fullest extent. This document aims to provide technical details for scriptwriters (who already know how to write scripts) to share and collaborate on script projects using the free and open-source GitHub software/service.

Note: If you are looking for instructions for how to write BlueZone Scripts in general, you may appreciate the MAXIS scriptwriter wiki at <https://github.com/MN-Script-Team/DHS-MAXIS-Scripts/wiki>.

# How GitHub works

GitHub is what’s referred to as a “repository hosting service”. Our repository (or place where we keep script code) is hosted on GitHub.com, which is similar in function to a “cloud storage” service such as Google Drive or Dropbox. What makes GitHub different, though, is that it utilizes a central administrator (in our case, Veronica Cary in her part-time employment with Anoka County) to decide which files/updates are included in the repository (or “repo” for short), and which aren’t. In this way, GitHub is significantly more stable and secure than sharing script files via email or traditional cloud storage.

In addition to the storage capabilities, GitHub keeps track of each version committed (or saved) and can be regressed at any time. This allows an administrator to “roll back” a change if it’s found to be difficult/non-functional.

GitHub also includes “branches” for scripts-work to be done, as well as for our end users to be able to access. For PRISM scripts, in addition to any “working branches” (explained a bit later), we also use “beta” and “release” branches (which are utilized live by our end users).

Finally, one of the best features of GitHub is its “client software”, called “GitHub for Windows”. GitHub for Windows allows you to easily sync a folder on your computer with your GitHub fork (and vice versa). That way, you can edit your script files on your computer, just like you’re used to, and syncing them to your fork is just a few clicks away!

# Before we begin

Before you do anything with script collaboration, you must sign up for a free GitHub account at [www.github.com](http://www.github.com). Please note that, while there are paid options for accounts with “private repositories” (places where nobody can see what code you’re creating), we are not using them. Follow the screen guidance at github.com to set up your free account. We’ll need that account for the rest of the instructions.

At this point, it may be wise to email your user ID information to [Veronica.Cary@co.anoka.mn.us](mailto:Veronica.Cary@co.anoka.mn.us), as she will probably want to add you to an “external collaborator” group (for reporting/tracking/taking issues).

# Setting up GitHub for Windows

The easiest way to set up and use GitHub is by downloading the GitHub for Windows client software. It is free and is available at <http://windows.github.com>. After downloading and installing the software, log in using your GitHub user ID (you may need to click the gear icon in the upper right to log in). Once your software is set up, follow these instructions.

1. Go to the main repo page at <https://github.com/MN-CS-Script-Team/PRISM-Scripts>.
2. In the right-hand column, click the button marked “Clone in Desktop”.
3. It will ask you details about cloning the repository in your desktop. The important detail is **where**. It usually defaults to your shared drive, or perhaps the “my documents” folder, but for our purposes, **we want to clone it to the C:\ drive**. Don’t use a subfolder, just save it to C:\. We use “C:\PRISM-Scripts\Script Files\” as the default script directory for BlueZone Child Support scriptwriters, so matching this directory is important for making sure your changes are easy to test for other users.
4. Now that that’s all done, you may find it helpful to go to C:\ PRISM-Scripts\Script Files\SETTINGS - GLOBAL VARIABLES.vbs, and updating the county name/number to match your own information, which defaults to Anoka County.
5. Finally, when testing your script, you may find it convenient to update BlueZone to point to this directory, in Script>Properties and View>Properties>Power Pads.

# What about command-line Git?

Some users may be more comfortable with the text driven “command line Git” than using the mouse-driven “GitHub for Windows” software. That’s fine! Command-line Git is very powerful, and in some cases, easier to use. However, it isn’t recommended for people who don’t remember command prompts (or DOS).

To bring up command-line Git, simply open up your script repository in GitHub for Windows, select the gear icon (upper right corner), then select “Open in Git Shell”. That will load up a Windows PowerShell session with Git preloaded, ready to use. Instructions below also include “how-to” information for command-line Git. If you aren’t using command-line Git, feel free to skip those instructions.

# Basic outline of how we share script files using GitHub

1. [Pull request from the main repository into your own fork.](#_Pull_request_from) This brings you in sync with the main scripts.
2. In the GitHub for Windows software, [sync your repository](#_Create_a_branch). This keeps your work in sync on your computer.
3. [Create a branch for your change](#_Create_a_branch_1). For example, if you are updating “closed programs”, you create a branch called “closed programs”. This lets you work on multiple scripts at once, without needing to send (pull request) all of your changes to the main repository at the same time. This is what we refer to as a “working branch”.
4. Using the script editor of your choice, [build/update the script](#_Using_your_script). Test it thoroughly in BlueZone. Save often.
5. Once you’re sure it’s working, [commit the script](#_Once_you’re_sure) to your repository.
6. In GitHub for Windows, [sync your repository](#_In_GitHub_for) with your fork on GitHub.
7. Back on GitHub.com, [initiate a pull request](#_From_your_fork’s) from your new branch to the “master” branch of PRISM-Scripts.  
   Note: if you are fixing a critical bug, you may be asked to also initiate a pull request into the “beta” or “release” branches of PRISM-Scripts.

From here, an administrator will either approve or close your pull request. If they close it, they will provide details as to why (and whether-or-not you have to fix something), and you’d start again from step 2. If they approve it, you can go ahead and remove the branch you created in step 2.

Now, we’ll go into detail for the above steps.

# How to share files using GitHub

## Pull request from PRISM-Scripts into your fork

This process is relatively confusing, and is often forgot when creating a new script or writing an update, but is absolutely vital. If you’re comfortable with command line navigation, you may want to skip this step and jump to [the next section](#_Optional:_pull_changes).

1. Navigate to your fork of the BlueZone Scripts (or create a new one). You can do this by going to the main BlueZone Scripts repository (which is located at <https://github.com/MN-CS-Script-Team/PRISM-Scripts>) and selecting the “fork” option in the top-right corner of the window (Figure 1).



Figure 1: a screenshot of the "fork" button in a GitHub repository.

1. Depending on your settings and other repositories available, it will either directly navigate to your fork, or ask you which fork to visit. Usually, you will visit the fork of your personal user. If you did not yet have a fork, this will create one for you.



Figure 2: the "pull request" button on GitHub.

1. If you created a brand new fork in step 2, you can stop here, because you’re already in sync. If you did not create a brand new fork, or aren’t sure, click on the green icon next to the branch selector on your repository page (Figure 2).
2. For some reason, it defaults to sending a pull request to the administrator, but we need to flip the order. To do that, follow these steps:
   1. Click on “Edit” in the upper-right hand portion of the window.
   2. For the “base fork”, select your repository from the list of repositories (should be <USERNAME>/PRISM-Scripts).
   3. The window should switch to a “there isn’t anything to compare” message. This is annoying, but not unexpected. To get past it, click “Edit” again, and this time, select “compare across forks”, then change the “head fork” to “MN-CS-Script-Team/PRISM-Scripts” (should be the top choice).  
      Pro tip: bookmark the page once you get to this step. Then, when you want to pull changes into your fork, you can just navigate directly to this one page.
3. If there’s any changes, click the green button that says “create pull request”.
4. Add a description if one isn’t already in there, then press the green “Create pull request” button.
5. Click on the button that says “Merge pull request”, followed by “Confirm merge”.

#### Optional/advanced: pull changes from PRISM-Scripts using the command line

Users who are comfortable with command line navigation may find the below series of commands easier than navigating the GitHub website. Otherwise, if you followed along with [the last section](#_Pull_request_from), go ahead and go to the next section.

1. Fetch the latest updates from our main repository.  
   git fetch upstream
2. Checkout the master branch from our main repository.  
   git checkout master
3. Merge the fetched master branch into your main repository.  
   git merge upstream/master
4. Pull all changes into your repository.  
   git pull
5. Push all of your changes out to GitHub to be stored.  
   git push

## Sync your repository

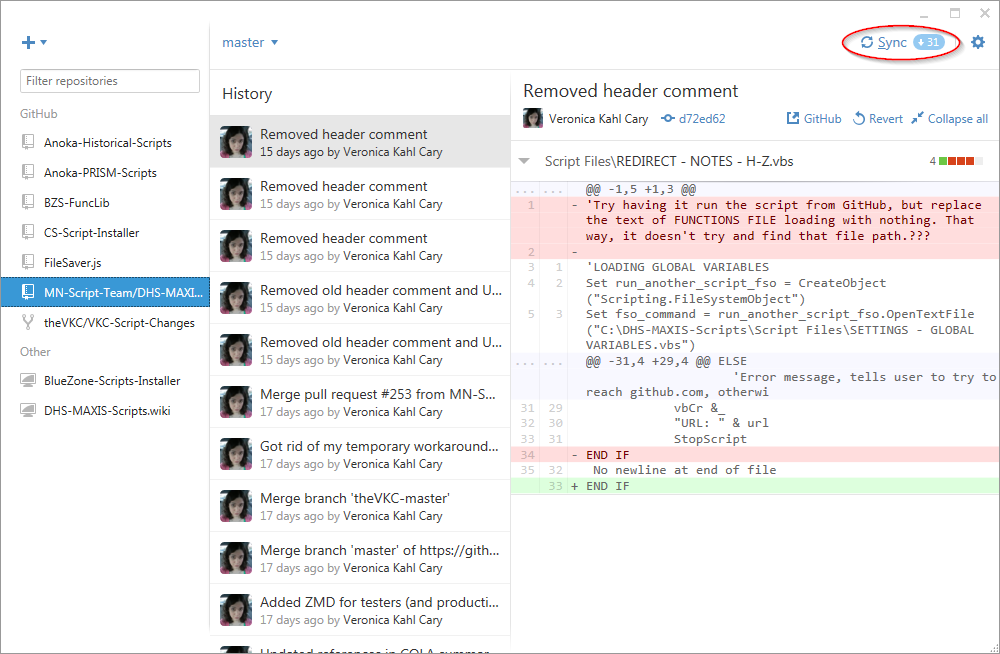


Figure 3: a screenshot of the location of the "sync" button in GitHub for Windows.

This part syncs what you’ve done online with what you’ll be using on your computer. It is one of the easiest parts of the GitHub process. To sync, simply select the “sync” button on the upper-right-hand side of the GitHub for Windows software (Figure 3).

If using command-line Git, simply use the command git pull.

After syncing your repository, we then have to create a working branch for your change to go into.

## Create a branch for your change

There are three ways to create a branch. On the website, in the client software, and via command line. We will not be discussing how to do so on the website (although there’s very good documentation available on GitHub.com to learn how to do so). We’ll be discussing how to do this via the client software or command line.

To do this with the GitHub for Windows software, select that dropdown with the current branch on it, right above the repo history (it usually defaults to “master”. After selecting the dropdown, enter the name of the new branch you’ll be working on in the “filter or create new” edit box. Select the “create <branch name>” checkbox, and your new branch is made (Figure 4). This allows your work on this issue to be “kept separate” from the other script files which may be changed, and is very useful for collaborative projects like this.

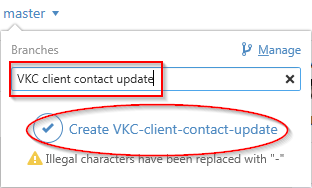


Figure 4: how to create a branch.

#### Optional/advanced: create a new branch using the command line

To create a branch from the command line, instead of using GitHub for Windows, simply use the following command:

git checkout –b VKC-client-contact-update

This checks out a new branch in your repo, called “VKC-client-contact-update”. Obviously, you’d call it whatever you’d call it in the real world.

### Notes about branches

Branches can (and should) be deleted if they are no longer being used. We will cover deleting branches later on. The primary repository uses three branches: “master” as a “working” directory for scriptwriters, “beta” for the beta agencies, and “release” for everyone else. **“Beta” and “release” are live and used by script users statewide, so it’s important that they remain functional at all times!**

## Build/update the script

This part, you should know how to do! Either create your new script, or update an existing script. An important thing to note is that you should focus on the one issue or idea in this branch. If you decide to take a break and work on another script project in the middle of this one, commit your changes ([see below](#_Once_you’re_sure)) and create a new branch from master, [following the instructions in the previous section](#_Create_a_branch_1). Test it well, and make sure it’s working before moving on to the next part.

## Commit the script to your repository

So you’ve tested your script, and you’re comfortable that it’s working exactly as we’d hope. Wonderful! Now you have to **commit** your script files to your repository. **Committing** a change is like taking a snapshot of your scripts, exactly the way they look right now. This is important, as we use these commits to track changes to our files, as well as to “regress” them if something doesn’t go right. It’s also very helpful for “versioning”, or “checking the difference between different versions of the same file”. A commit is like a placeholder to be used in comparisons.

Committing is easy. In the GitHub for Windows software, after you’ve saved your work from the previous step, it should show you the changed file in the right-hand side of the software (under “Files to Commit”). Check over the file (or files) and make sure it looks like the file you’ve changed. Then, when you’re ready, in the “uncommitted changes” window (center column of the GitHub for Windows software) enter a short summary of the change followed by a description.

Pro tip: you can tag other GitHub users using the “@” symbol, followed by their username.

#### Optional/advanced: commit a file using the command line

Committing a file with the command line isn’t tricky. Just use the following commands.

1. Add all files you’ve changed to the list-to-be-committed. (The period sends all files, and is needed.)  
   git add .
2. Commit the changes to the Git record, with a short message explaining what changed.  
   git commit -m “This is a change”

## In GitHub for Windows, sync your repository with your fork on GitHub.

Once you’ve committed your file (or files), we need to sync the repository on your computer with GitHub.com. That’s easy. Simply select the “sync” (or “publish” if it’s a new branch) option in the GitHub software, exactly where we found it [earlier in the instructions](#_Sync_your_repository) (Figure 3).

#### Optional/advanced: sync using the command line

To sync a new branch, simply use git push –set-upstream origin <branch name>. To sync an existing branch, it’s a simple as git push.

## Initiate a pull request from your new branch to the “master” branch of PRISM-Scripts

This is the part which makes GitHub so special. Up until now, we haven’t done much that you can’t do via standard file/save methods. In fact, you may have been asking yourself “why are we going to all of this extra work?” Here is where that work pays off, in the “pull request” process. In this step, you will submit your script enhancement to the administrator as a **pull request**, meaning **a request to pull your change into the main repository**. This process is relatively complicated, compared to the other processes we’ve learned, but it greatly simplifies the process of administering BlueZone Scripts across multiple counties.



Figure 5: the "pull request" button on GitHub.

Follow these steps to initiate a pull request on GitHub.com.

1. Navigate to the main page of your GitHub repository. (If you lost it, you can find it again by following steps 1-2 on [page 3](#_Pull_request_from).)  
   Note: if you just synced a change with your own fork, you may see a handy “compare & pull request” button. If you do, select it and skip to step 5.
2. Click on the green “pull request” button (Figure 5).
3. Click “compare across forks” near the top of the window, and change the “head fork” to your own repository.
4. Like before, it will automatically show you a list of changes, and your list may or may not have any changes in it at all. In any case, click on “Edit” (upper right of window) and change the “compare” branch to the branch you created earlier.
5. You should see a green “Create pull request” button, similar to what we saw on [page 3](#_Pull_request_from). Look over the changes indicated near the bottom of the window. If everything looks good, go ahead and click “Create pull request”, then insert a comment/description, and press “Create pull request” again. That will send it to the administrators for consideration!

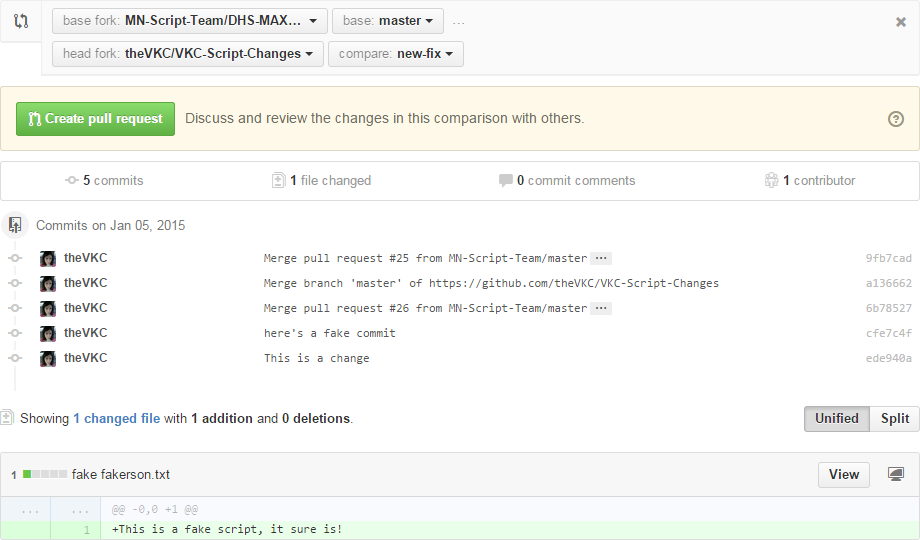


Figure 6: a sample of the pull request screen, showing a base fork and head fork, with a branch called "new-fix" slated to be pulled into the main scripts. It also showcases the list of commits as well as the contents of the new file.

# Other important GitHub concepts

## Deleting unused branches

It’s important to delete an unused branch when you’re done with it. Script files change frequently: if you’re working off of an older branch, you may inadvertently be basing your changes off of very old script data, and that can result in commit conflicts which administrators have to work out manually. Yuck!

To delete a branch in GitHub for Windows, click on the branch selector above the history column (it usually says “master”), then click “manage”. From here, you should see your list of branches, and can use the trash can icon to delete an old one.

Pro tip: If you’re in the branch-to-be-deleted, you may need to select another branch in the list first. You cannot delete the branch you’re currently working in.

## Issue tracking

We use GitHub’s issue tracking features to assign/track/resolve issues, ideas, questions, and enhancements with BlueZone Scripts. Using these features is relatively simple: go to the main page at <https://github.com/MN-CS-Script-Team/PRISM-Scripts>, and click on “Issues” in the right-hand column. From here, you can report a new issue (please, please, please search first to see if the issue was already reported), scan existing issues by about a dozen different subjects, and “claim” issues/ideas to try and take on yourself.

To “claim” an issue or enhancement, simply comment on the issue suggesting that you want to take it on. You may want to check first to see if there’s an assignee in the assignee area of the right column. If there’s no assignee, and you feel comfortable taking on the task, go ahead and comment on it suggesting you’d like to give it a try! You may want to wait until an administrator adds you as the “assignee” before taking it on, just to make sure you don’t end up doing something someone else already started.

## Common GitHub-related terms

* Branch – different versions of the same script code, in the same repository.
* Commit – a saved change to a file (or group of files).
* Fork – a “copied” version of the script code, made by a specific user. Users make changes to scripts in forks, then they pull request them to the main scripts.
* Pull request – a request to merge a fork into the main branch, or vice versa.
* Repository – place where we keep script code. Often called a “repo” for short.